Project 8Bit

Fully functional and programmable 8 Bit Computer out of Logic Chips

0x4d/Martin Loretz

Maker & Software Developer

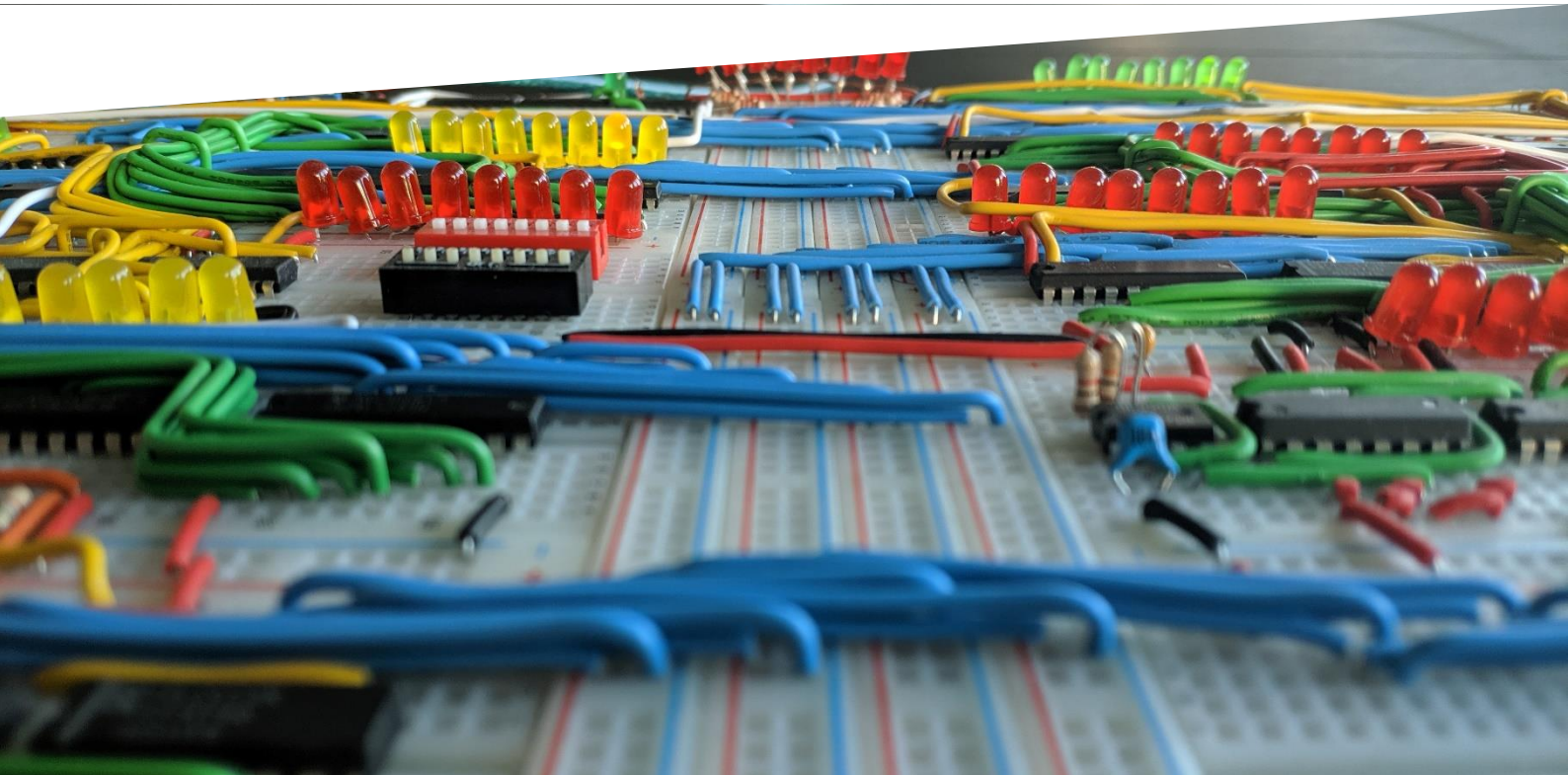
Introduction

Project 8Bit ist ein vollständig funktionstüchtiger, speicherprogrammierbarer Computer aus 74LSXX Logic Chips mit einer Busbreite von 8 Bit. Dieser Computer ist Turing Complete, was bedeutet, dass er alles machen kann, was auch ein Computer berechenbar ist. Aus diversen Register, Counter und anderen Logik-Gatter ist er auf 14 Breadboards aufgebaut und durch LED's an allen Register ist genau ersichtlich, welche Daten wo intern übertragen werden.

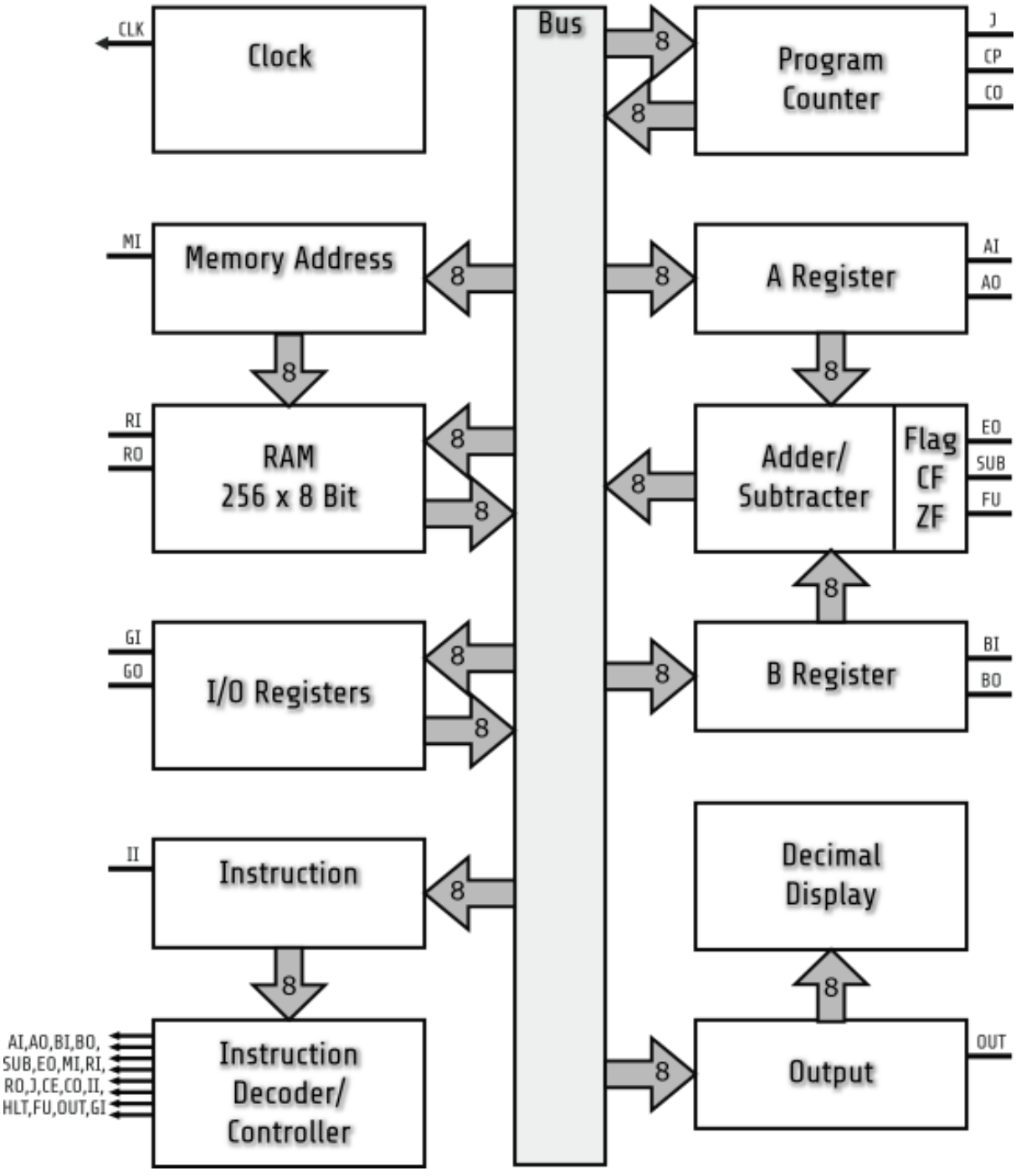
Der prinzipielle Aufbau basiert auf dem vom Youtuber Ben Eater, welcher eine 44 Teilige Videoserie zum Bau des Computers gemacht hat. Meine Version ist modifiziert, sie hat 16 mal mehr Speicher, I/O Register, verwendet zum Teil andere Chips und hat ein anderes Instruction Set.

Features

- 8 Bit Busbreite
- 14 Breadboards
- 54 Chips
- 57 LED's
- 60+ Meter Kabel
- 256 Byte RAM
- 400h+ Arbeitszeit

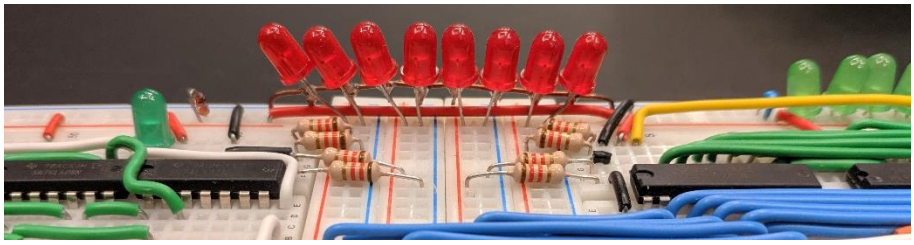


Architecture



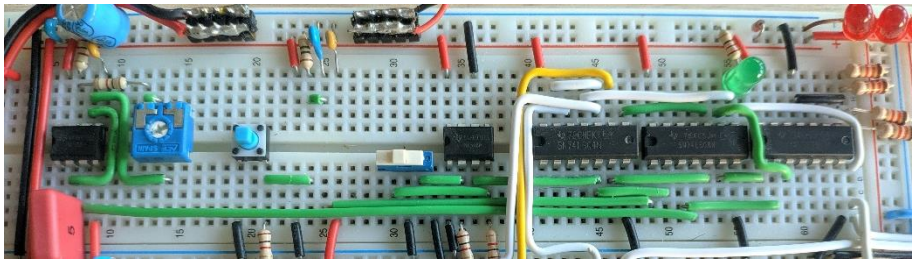
Main Bus

Alle Baugruppen können miteinander über diesen Bus kommunizieren, welcher eine Busbreite von 8 Bit hat. Zu jedem Zeitpunkt kann nur eine Baugruppe auf den Bus Daten senden, doch diese können von mehreren empfangen werden.



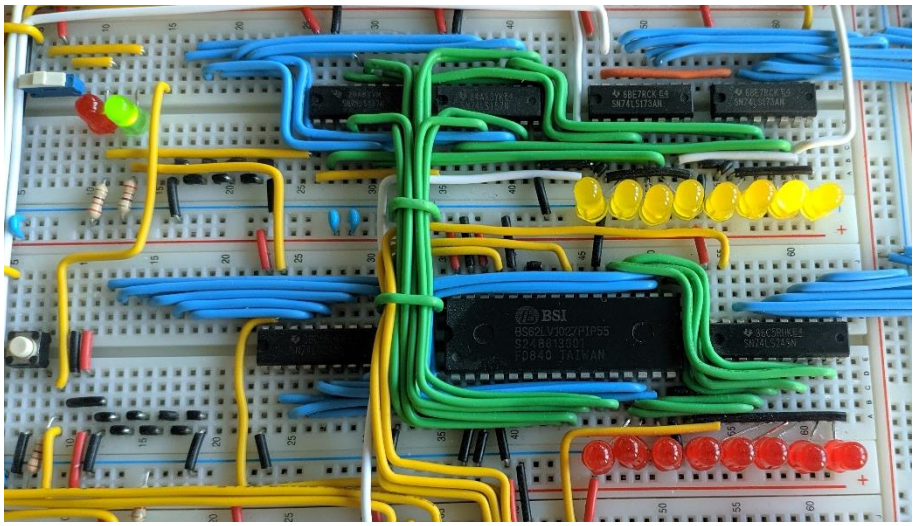
Clock

Die Clock gibt für den gesamten Computer das Taktsignal vor. Sie hat 2 Betriebsarten, einen Automatik-Modus in dem sie ein Taktsignal mit verstellbarer Frequenz ausgibt [1-400Hz] und ein Manueller Modus, bei dem jeweils um einen Taktimpuls weitergesprungen wird.



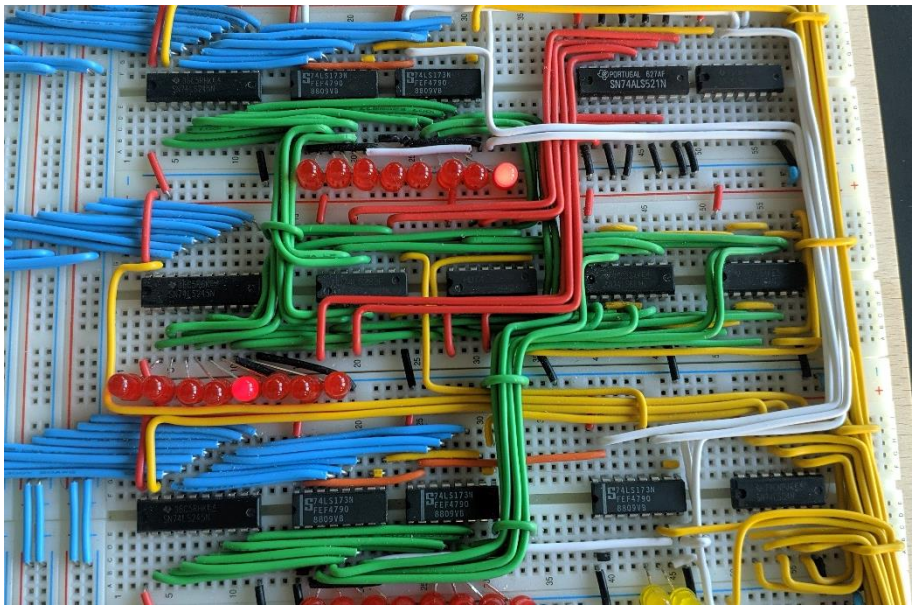
RAM + Memory Address Register

Im RAM [Random Access Memory] wird sowohl das Programm, als auch die Daten/Variablen gespeichert. Der RAM hat eine adressierbare Größe von 256 Byte [Chip könnet 2MByte] und ist als einziges Bauelement in CMOS Technologie ausgeführt. Um den Computer zu programmieren, muss das ganze Programm in Machinecode direkt in den RAM geschrieben werden.



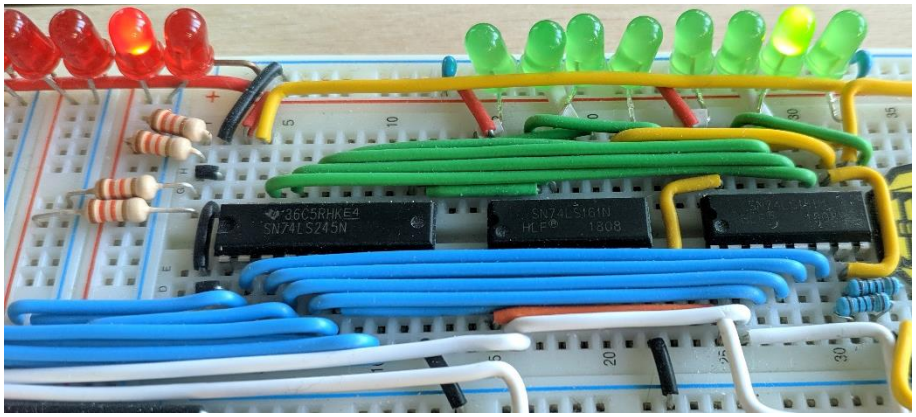
ALU + A/B Register

Im ALU(Arithmetic Logic Unit) kann der Inhalt des A und B Registers addieren und subtrahieren. Das ALU liefert auch ein Meldesignal, wenn das Ergebnis 0 oder größer als 255 ist, womit IF Anweisungen realisiert werden können. Im A and B Register werden die beiden Zahlen vor der Addition/Subtraktion zwischengespeichert.



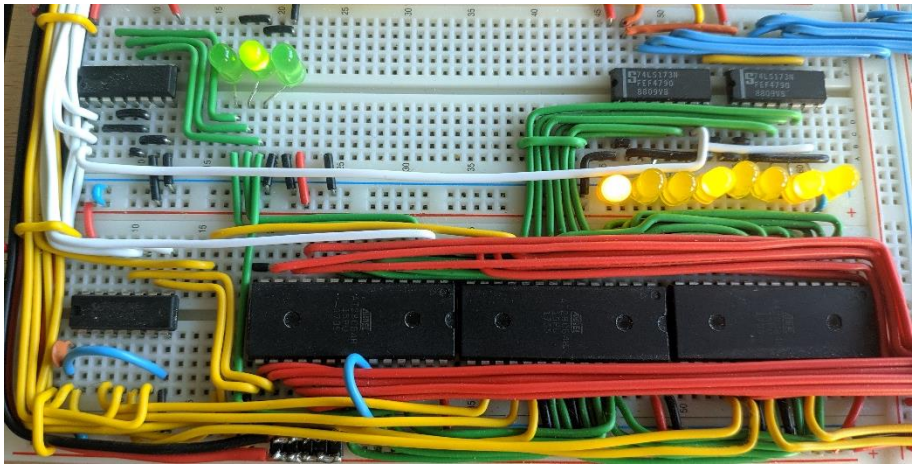
Programm Counter

Im Programm Counter wird jene RAM Adresse zwischengespeichert, welche abgefragt werden muss, um den nächsten Befehl aus dem RAM zu hohlen. Es ist auch möglich, Werte in den Programm Counter zu schreiben, womit Jump/GOTO Befehle und in weiterer Folge Schleifen und IF Anweisungen realisiert werden.



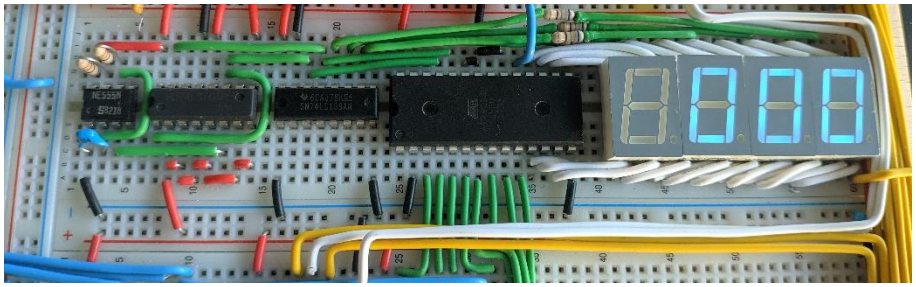
Instruktion Decoder

Der Instruktion Decoder dekodiert die Befehle und setzt alle Steuerungsleitungen so, damit bei den nächsten Taktimpulsen der Computer eine bestimmte Funktion ausführt.



Output Display + I/O Register

Zur Ausgabe von Werten kann das Zahlendisplay aus vier 7-Segmentanzeigen verwendet werden. Desweitern gibt es noch ein binäres Input und ein binäres Output Register, mit dem externe Elektronik wie mit einem Microcontroller angesteuert werden können.



Fetch Execute Cycle

Zum Abarbeiten eines Befehls müssen folgende Dinge gemacht werden:

1. Programm Counter in das Memory Address Register übertragen [Fetch]
2. Befehl vom RAM zum Instruction Decoder übermitteln
3. Befehl decodieren und Steuerungsleitungen aktivieren [Decode]
4. Befehl ausführen [Execute]
5. Programm Counter inkrementieren

Programming

Der Computer ist komplett Speicherprogrammierbar, indem das Programm in Machinecode in den RAM gespeichert wird. Zur einfachen Programmierung wurde ein Arduino Programmer gebaut. Das Programm startet nach drücken des Reset Knopfes. Die Umwandlung der Befehle in Machinecode kann händisch mit der Instruction Set Tabelle gemacht werden, nur da das recht aufwendig ist, wurde ein kleines JAVA Programm für diese Aufgabe geschrieben.

Instruction Set

Instruction Set

0	0x00	00000000	NOP	-	-
1	0x01	00000001	LDA	FA	A = RAM[FA]
2	0x02	00000010	LIA	NU	A = NU
3	0x03	00000011	STA	TA	RAM[TA] = A
4	0x04	00000100	LDB	FA	B = RAM[FA]
5	0x05	00000101	LIB	NU	B = NU
6	0x06	00000110	STB	TA	RAM[TA] = B
7	0x07	00000111	OUT	FA	OUT = RAM[FA]
8	0x08	00001000	OUTI	NU	OUT = NU
9	0x09	00001001	JA	AD	PC = AD
10	0x0A	00001010	JI	NU	PC = NU
11	0x0B	00001011	SET	N,A	RAM[AD] = N
12	0x0C	00001100	CPY	A;A	RAM[AD] = RAM[AD]
13	0x0D	00001101	ADD	TA	RAM[TA] = A + B
14	0x0E	00001110	SUB	TA	RAM[TA] = A - B
15	0x0F	00001111	INV	AD	RAM[AB] = ! RAM[AD]
16	0x10	00010000	SHL	AD	RAM[AB] = << RAM[AD]
17	0x11	00010001	SHR	AD	RAM[AB] = >> RAM[AD]
18	0x12	00010010	HLT	-	CLK = 0
19	0x13	00010011	JZ	AD	(ZF == 1) ? PC = AD
20	0x14	00010100	JC	AD	(CF == 1) ? PC = AD
21	0x15	00010101	INT1	1	(INT1 == 1) ? PC++
22	0x16	00010110	INT2	1	(INT2 == 1) ? PC++
23	0x17	00010111	EXP	FA	EXP = RAM[FA]
24	0x18	00011000	EXPI	NU	EXP = NU
25	0x19	00011001	IMP	TA	RAM[TA] = IMP
26	0x1A	00011010	NOP	-	-
27	0x1B	00011011	NOP	-	-
28	0x1C	00011100	NOP	-	-
29	0x1D	00011101	NOP	-	-
30	0x1E	00011110	NOP	-	-
31	0x1F	00011111	NOP	-	-

Instruction →Microcode [Sequencing]

0x00	NOP		CO|MI	RO|II|CE	ICR	0	0	0	0	0
0x01	LDA	FA	CO|MI	RO|II|CE	CO|MI	RO|MI|CE	RO|AI	ICR	0	0
0x02	LIA	NU	CO|MI	RO|II|CE	CO|MI	RO|AI|CE	ICR	0	0	0
0x03	STA	TA	CO|MI	RO|II|CE	CO|MI	RO|MI|CE	AO|RI	ICR	0	0
0x04	LDB	FA	CO|MI	RO|II|CE	CO|MI	RO|MI|CE	RO|BI	ICR	0	0
0x05	LIB	NU	CO|MI	RO|II|CE	CO|MI	RO|BI|CE	ICR	0	0	0
0x06	STB	TA	CO|MI	RO|II|CE	CO|MI	RO|MI|CE	BO|RI	ICR	0	0
0x07	OUT	FA	CO|MI	RO|II|CE	CO|MI	RO|MI|CE	RO|OUT	ICR	0	0
0x08	OUTI	NU	CO|MI	RO|II|CE	CO|MI	RO|OUT|CE	ICR	0	0	0
0x09	J	AD	CO|MI	RO|II|CE	CO|MI	RO|MI|CE	RO|J	ICR	0	0
0x0A	JI	NU	CO|MI	RO|II|CE	CO|MI	RO|J	ICR	0	0	0
0x0B	SET	N,A	CO|MI	RO|II|CE	CO|MI	RO|AI|CE				
0x0C	CPY	A,A	CO|MI	RO|II|CE	CO|MI	RO|MI|CE				
0x0D	ADD	TA	CO|MI	RO|II|CE	CO|MI	RO|MI|CE				
0x0E	SUB	TA	CO|MI	RO|II|CE	CO|MI	RO|MI|CE				
0x0F	INV	AD	CO|MI	RO|II|CE	CO|MI	AI				
0x10	SHL	AD	CO|MI	RO|II|CE	CO|MI	RO|AI|BI				
0x11	SHR	AD	CO|MI	RO|II|CE	CO|MI	RO|AI|BI				
0x12	HLT	-	CO|MI	RO|II|CE	HLT	0				
0x13	JZ	AD	CO|MI	RO|II|CE	CO|MI	RO|MI|CE				
0x14	JC	AD	CO|MI	RO|II|CE	CO|MI	RO|MI|CE				
0x15	INT1	-	CO|MI	RO|II|CE	CO|MI|AI	RO|BI				
0x16	INT2	-	CO|MI	RO|II|CE	CO|MI|AI	RO|BI				
0x17	EXP	FA	CO|MI	RO|II|CE	CO|MI	RO|MI|CE				
0x18	EXPI	NU	CO|MI	RO|II|CE	CO|MI	RO|GO|CE				
0x19	IMP	TA	CO|MI	RO|II|CE	CO|MI	RO|MI|CE				
0x1A	NOP	-	MI|CO	RO|II|CE	ICR	0				
0x1B	NOP	-	MI|CO	RO|II|CE	ICR	0				
0x1C	NOP	-	MI|CO	RO|II|CE	ICR	0				
0x1D	NOP	-	MI|CO	RO|II|CE	ICR	0				
0x1E	NOP	-	MI|CO	RO|II|CE	ICR	0				
0x1F	NOP	-	MI|CO	RO|II|CE	ICR	0				

MicroCode Set

1	0x00	AI	Bus → A	A
2	0x01	AO	A → Bus	
3	0x02	BI	Bus → B	B
4	0x03	BO	B → Bus	
5	0x04	SUB	SUM = A - B	ALU
6	0x05	EO	SUM → Bus	
7	0x06	MI	Bus → M	
8	0x07	RI	Bus → RAM[M]	RAM
9	0x08	RO	RAM[M] → Bus	
10	0x09	J	Bus → C	
11	0x0A	CE	C++	PC
12	0x0B	CO	C → Bus	
13	0x0C	II	Bus → I	
14	0x0D	ICE	IC++	CONT
15	0x0E	ICR	IC = 0	
16	0x0F	HLT	CLK = 0	CLK
17	0x10	FU	CF = SUM.CF	ALU
18	0x11	OUT	Bus → OUT	
19	0x12	GI	IO → Bus	
20	0x13	GO	Bus → IO	IO
21	0x14	NI	N → Bus	
22	0x15	NO	Bus → N	
23	0x16	R	-	-
24	0x17	R	-	

Programm Examples

Ein Programm zum Addieren von den Zahlen 0x17 und 0x23 kann beispielsweise wie unten abgebildet aufgebaut sein, rechts ist ein etwas komplexeres Programm, welches die Fibonacci Folge berechnet. [1. Spalte: Programmzeile, 2.Spalte: Programmbefehl, 3.Spalte: Kommentar, 4. Spalte: Machinecode]

Add

0x00: JI	J 0x05	0x0A
0x01: 0x05		0x05
0x02: V 0x17	x	0x17
0x03: V 0x23	y	0x23
0x04: V 0x00	z	0x00
0x05: LDA	LDA x	0x01
0x06: 0x02		0x02
0x07: LDB	LDB y	0x04
0x08: 0x03		0x03
0x09: ADD	z = x + y	0x0D
0x0A: 0x04		0x04
0x0B: OUT	OUT z	0x07
0x0C: 0x04		0x04
0x0D: HLT	HLT	0x12

In den Addressen 0x02, 0x03, 0x04 des RAM'S werden die Variablen x, y, und z gespeichert. Damit diese nicht als Programmcode interpretiert werden, wird mit dem ersten Befehl JI 0x05 dieser Teil des Speichers übersprungen. Anschließend wird zuerst x in das A Register und y in das B Register gespeichert. Das ALU addiert diese beiden Zahlen und speichert das Ergebnis wird wieder zurück in die RAM Addresse der z Variable. Diese Variable wird am Display ausgegeben und das Programm/die Clock wird angehalten.

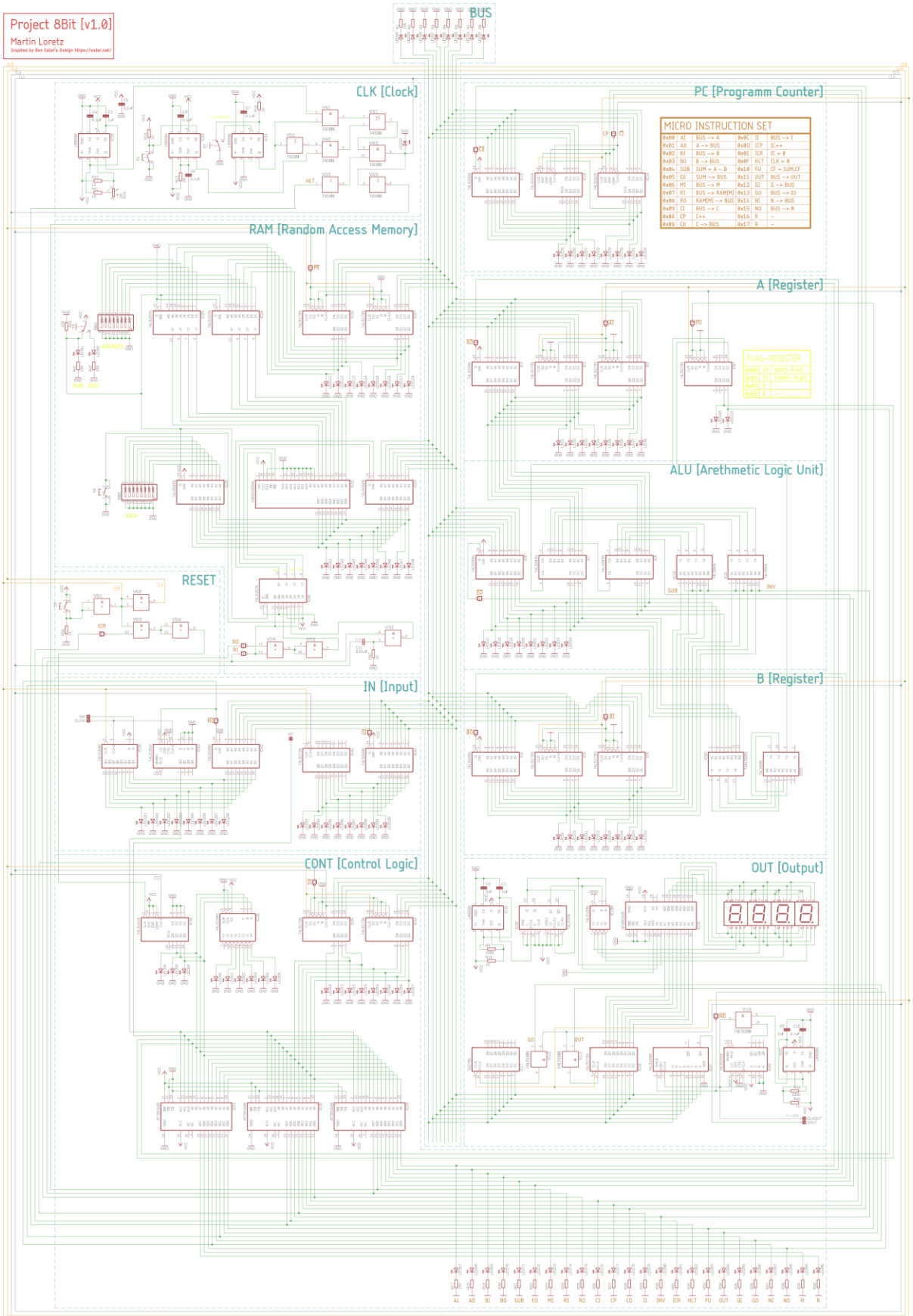
Das Fibonacci Programm funktioniert ähnlich, ist aber etwas komplexierter und enthält eine while Schleife, mit einer IF Abfrage als break Bedingung.

Fibonacci

0x00: JI	J 0x05	0x0A
0x01: 0x05		0x05
0x02: V 0x00	x	0x00
0x03: V 0x00	y	0x00
0x04: V 0x00		0x00
0x05: LIA		0x02
0x06: 0x00	x = 0	0x00
0x07: STA		0x03
0x08: 0x02		0x02
0x09: LIB		0x05
0x0A: 0x01	y = 1	0x01
0x0B: STB		0x06
0x0C: 0x03		0x03
0x0D: LDA	LDA x	0x01
0x0E: 0x02		0x02
0x0F: LDB	LDB y	0x04
0x10: 0x03		0x03
0x11: ADD	z = x + y	0x0D
0x12: 0x04		0x04
0x13: OUT	OUT z	0x07
0x14: 0x04		0x04
0x15: LDA		0x01
0x16: 0x04		0x04
0x17: LIB		0x05
0x18: 0xE9	(z == 233) ? J 0x05	0xE9
0x19: SUB		0x0E
0x1A: 0x30		0x30
0x1B: JZ		0x13
0x1C: 0x01		0x01
0x1D: LDA		0x01
0x1E: 0x03	x = y	0x03
0x1F: STA		0x03
0x20: 0x02		0x02
0x21: LDB		0x04
0x22: 0x04	y = z	0x04
0x23: STB		0x06
0x24: 0x03		0x03
0x25: JI	J 0x0D	0x0A
0x26: 0x0D		0x0D


Mutterschiff
AI AO BI BO SU EO MI RI RO JMP CE CI INV ICR HLT FU OUT G0 R1 R2 R3 R4
BSI
AI AO BI BO SU EO MI RI RO JMP CE

Schematic


Project 8Bit [v1.0]
Martin Loretz
BUS
CLK [Clock]
PC [Programm Counter]
MICRO INSTRUCTION SET
RAM [Random Access Memory]
A [Register]
FLAG-REGISTER
ALU [Arethmetic Logic Unit]
RESET
B [Register]
IN [Input]
CONT [Control Logic]
OUT [Output]