Platinencomputer - Handbuch

Alexander Wersching und Simon Walter

2021

Inhaltsverzeichnis

1 Einführung

1.1 Ziel diese Dokuments

Dieses Dokument soll die Funktionsweise unseres Platinencomputers beschreiben. Dabei soll nicht auf die tatsächliche auf Implementation wert gelegt werden, aber viel mehr auf die einzelnen Features und Funktionen. Diese Spezifikation wurde vor dem Bau des Computers bzw. des Emulators geschrieben, um eine Grundlage für alle weiteren Konstruktionsschritte zu haben.

1.2 Geschichte

Dieses Dokument hat sehr viel Änderung durchleben müssen, diese können unter `https://github.com/jufo-ufo/Breadboard-Computer` eingesehen werden. Zudem hatte diese Spezifikation zwei, wenn auch nicht vollständige, Vorgänger: `https://github.com/jufo-ufo/Breadboard-Computer/blob/master/specification_v0.1.txt` und `https://github.com/jufo-ufo/Breadboard-Computer/blob/master/specification_v0.2.txt`. Wobei die zweite Fassung als Basis dieses Dokuments diente.

2 Spezifikation des Computers

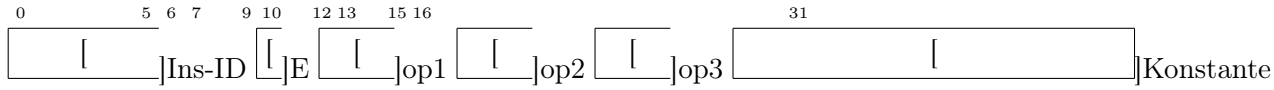
Ein Computer hat eine Reihe verschiedener "high-level" Aufgaben und Features, die es ihm erlauben, seiner Tätigkeiten nach zu gehen! Diese Features können wie folgt auf gegliedert werden: Register, Memory, Input/Output, Interrupts, Supervisor-/Usermode und der Instruktionssatz.

Aber um zuerst ein paar Standards festzulegen: Der Computer hat eine Daten-/Adressenlänge von 16 Bit, d.h. er kann maximal 16 Bit auf einmal operieren bzw. können maximal 16 Bit an Daten/Adressen im Computer auf einmal bewegt werden. Mehr dazu in Unterabschnitt 3.1

Unsere Computer ist eine von Neumann-Maschine, d.h. sowohl Daten als auch Programm während in einem Hauptspeicher (Memory) gespeichert

2.1 Instruktions-Satz

Das Programm, welches den Computer vorschreibt, was er abzuarbeiten hat, besteht aus Instruktionen. Jede Instruktion enthält die 6 Bit langen Ins-ID, ein 16 Bit Konstante und 3 gewählte Register in Form einer 3 Bit Register Nummer (RN). Damit ergibt sich eine 32 Bit, also 2 words, lange Instruktion. Die Codierung ist dabei in Unterabschnitt 2.1 festgehalten.



Jeder zur Verfügung stehenden Instruktion ist dabei exakt einer oben erwähnten Ins-ID zugeordnet. Alle Ins-ID und dazugehörige Instruktion ist in Tabelle 1

Jede Instruktion hat 3 Operatanten (op), welche Register sind. Im Falle des dritten Operatanten kann auch eine Konstante verwendet werden. Das E Bit gibt dabei an, welches von beiden genutzt werden soll. Bei gesetztem E Bit wird die Konstante verwendet, bei nicht gesetztem E Bit das Register.

2.2 Supervisor-/Usermode

Der Computer hatte ein eingebautes Rechte-System. Er kann dabei in zwei Modi operieren: dem Supervisormode und Usermode. Im Supervisormode kann der Computer alle Funktion ohne Einschränkungen nutzen. Im Usermode hingegen ist der Computer in einigen Funktion eingeschränkt. Folgendes kann der Computer im Usermode *nicht* machen:

- Setzen der SUP Flagge
- Änderung der OINT Flagge
- Lesen und/oder schreiben im Bereich von `0x0000` - `0x7fff`
- Änderung der Banken
- Die Modifikation der IO-Berechtigungsliste
- Zugriff auf IO-Geräte, welche nicht in der Berechtigungsliste gespeichert sind (siehe Unterunterabschnitt 2.5.4)

Tabelle 1: Instruktions-Satz

Ins-ID	Name	op1	op2	op3	Beschreibung
0x00	NOP				Kein Effekt
0x01	MOV	reg		reg/c	op1 = op3
0x02	ADD	reg	reg	reg/c	op1 = op2 + op3
0x03	SUB	reg	reg	reg/c	op1 = op2 − op3
0x04	MUL?	reg	reg	reg/c	op1 = op2 ∗ op3
0x05	MULOF	reg	reg	reg/c	op1 = Overflow op2 ∗ op3
0x05	DIV?	reg	reg	reg/c	op1 = op2 / op3
0x06	XOR	reg	reg	reg/c	op1 = op2 ⊕ op3
0x07	AND	reg	reg	reg/c	op1 = op2 ∧ op3
0x08	OR	reg	reg	reg/c	op1 = op2 ∨ op3
0x09	NOT	reg		reg/c	op1 = ¬op3
0x0a	STR	reg	reg	reg/c	memory[op2 + op3] = op1
0x0b	LD	reg	reg	reg/c	op1 = memory[op2 + op3]
0x0c	BNK1			reg/c	Setzt BID für Bank-Slot1
0x0d	BNK2			reg/c	Setzt BID für Bank-Slot2
0x0e	PUSH			reg/c	SP–; memory[SP] = op1
0x0f	POP			reg	op3 = memory[SP]; SP++
0x10	CALL			reg/c	memory[SP] = IP; SP++; IP = op3
0x11	RET				IP = memory[SP]; SP++
0x12	TEST	reg		reg/c	Vergleicht op1 und op3
0x13	ME	reg		reg/c	op1 = op3 if E
0x14	MG	reg		reg/c	op1 = op3 if G
0x15	ML	reg		reg/c	op1 = op3 if L
0x16	MS	reg		reg/c	op1 = op3 if SUP
0x17	MI	reg		reg/c	op1 = op3 if OINT
0x18	MOFadd	reg		reg/c	op1 = op3 if O
0x19	MOFsub	reg		reg/c	op1 = op3 if O
0x1a	IOUT	reg		reg/c	IIO[op1] = op3/c
0x1b	DOUT	reg		reg/c	DIO[op1] = op3/c
0x1c	DIN	reg		reg	op3 = DIO[op1]
0x1d	AIO	reg			Aktiviert IO[op1]
0x1e	DIO	reg			Deaktivieren IO[op1]
0x1f	INT				Löst einen Interrupt aus
0x20	TM1			reg/c	Modi Timer1 = op3
0x21	TM2			reg/c	Modi Timer2 = op3
0x22	SSTL1			reg/c	lower Stop Timer1 = op3
0x23	SSTL2			reg/c	lower Stop Timer2 = op3
0x24	SSTH1			reg/c	higher Stop Timer1 = op3
0x25	SSTH2			reg/c	higher Stop Timer2 = op3
0x26	CFL	reg			op1 = lower Clock Freq
0x27	CFH	reg			op1 = higher Clock Freq

Der Modus des Computers (Supervisor or User) wird dabei in der SUP Flagge gespeichert. Eine gesetzte SUP Flagge entspricht dem Supervisormode und eine nicht gesetzte SUP dem Usermode Der Computer started dabei mit gesetzter SUP Flagge.

Zudem ändert sich je nach Modus des Computer die Start position des Stacks wie in Unterunterabschnitt 2.4.2 erläutert.

Mehr zu der Motivation hinter diesem System in Unterabschnitt 3.3

2.3 Registers

Register sind kleine, sehr schnelle Speichereinheiten, die genutzt werden können, um Zwischenergebnisse, lokal Variablen, Argumenten für Funktionen zu speichern oder um Daten im Computer zwischen den einzelnen Komponenten zu verschieben. Aus jedem Register kann *gelesen* (*Kopierten* des alten Wertes an einen anderen Ort) oder *geschrieben* werden (den alten Wert mit einem neuen Wert *überschreiben*)

Der Computer besitzt 8 verschiedene, jeweils 16 Bit lange Register. Jedes Register hat dabei eine eigene Registernummer (RN). Machen Register erfüllen zudem noch spezielle Aufgaben und sollten auch nur so verwendet werden. Diese Funktionen sind in Tabelle 2 zu sehen.

Mehr zur Registeranzahl in Unterabschnitt 3.2

Tabelle 2: Registeraufteilung

RN	Name	Funktion
0	ZERO	Konstante 0x0000; Schreiben hat keinen Effekt
1	IP	Instruction Pointer, zeigt auf aktuell auszuführende Instruktion
2	SP	Stack Pointer, zeigt auf obersten Wert des Stacks, siehe 2.4.2
3	A	Allzweck Register ohne besonderen Aufgaben
4	B	Allzweck "
5	C	Allzweck "
6	D	Allzweck "
7	FLAG	Enthält alle wichtigen Flaggen, siehe Unterunterabschnitt 2.3.2

Die Register A-D sind dabei Register, welche frei für jede Nutzung zur Verfügung stehen. Es ist *stark zu empfehlen*, die Spezialregister (ZERO, IP, SP und FLAG) *nicht* für die Speicherung von Daten zu verwenden, da dies evtl. Programmabläufe stark stören könnte

oder im Falle von ZERO oder den letzten 8 bit des Flaggen Registers im Usermode nicht möglich ist. Sie sind ausdrücklich nur für ihre zugewiesenen Aufgaben da.

2.3.1 Kopieren und Bewegen von Daten

Das Kopierten eines Wertes in ein Register ist mit der MOV-Instruktion möglich. Die MOV-Instruktion nimmt dabei als ersten Operanden das Ziel Register und als zweiten Operanden das Quellenregister oder eine Konstante an. MOV kopiert nun den Wert des zweiten Operanden in den des Ersten. Die Notation ist dabei wie folgt:

MOV <Ziel-Register> <Quellen-Register/Konstante>

2.3.2 Flaggen

Das Register FLAG enthält alle Statusflaggen. Eine Flagge kann entweder gesetzt oder nicht gesetzt sein, daher kann jede Flagge als ein Bit repräsentiert werden. Jedes Bit in FLAG stellt daher eine Flagge da. Flaggen werden bei unterschiedlichen Ereignissen geschaltet. Dies und welche Flaggen es gibt, ist in Tabelle 3 zusehen.

Tabelle 3: Flaggenaufteilung in FLAG

Bit	Name	Funktion	Schaltung
0			
1	E	Gleichheit zweier Werte	TEST op1 = op2
2	G	Ungleichheit (>) zweier Werte	TEST op1 > op2
3	L	Ungleichheit (<) zweier Werte	TEST op1 < op2
4	OFadd	Integer Overflow bei einer Addition	TEST op1 + op2 > 65535
5	OFsub	Integer Overflow bei einer Subtraktion	TEST op1 - op2 < 0
6			
7			
8	OVTI1	Overwrite Timed-Interrupt1, siehe Unterabschnitt 2.6	
9	OVTI2	Overwrite Timed-Interrupt2, siehe Unterabschnitt 2.6	
10	OVIOI	Overwrite IO-Interrupt, siehe Unterabschnitt 2.6	
11			
12			
13			
14			
15	OINT	Overwrite Interruption	
15	SUP	Supervisor-/Usermode, siehe Unterabschnitt 2.2	

Das FLAG Register verhält sich ebenfalls wie ein Register, daher können Flagge auch per Hand gesetzt oder nicht gesetzt werden. Eine Ausnahme machen die letzten 8 Flaggen darunter SUP und OINT die nur geschaltet werden können, wenn der Computer sich im Supervisormode befindet, siehe Unterabschnitt 2.2. Ein Schalten im Usermode hat keinen Effekt.

Flaggen 0-7 werden bei dem Ausführen der TEST-instruktion gesetzt, die restlichen nur manuel.

Neben der MOV-Instruktion gibt es auch die sogenannten konditionalen MOV-Instruktionen. Diese funktionieren genauso wie die MOV-Instruktion, indem sie das Quellen-Register/Konstante in das Ziel-Register kopiert, aber dies jedoch nur tun, wenn die entsprechende Flagge gesetzt ist. Die Notation dafür ist wie folgt:

M<Flagge> <Ziel-Register> <Quellen-Register/Konstante>

Daher ergeben sich folgende konditionalen MOV-Instruktionen: ME (E Flagge), MG (G Flagge), ML (L Flagge), MI (ONIT Flagge), MS (SUP Flagge), MOFadd (OFadd Flagge), MOFsub (OFsub Flagge).

2.3.3 Arithmetische und Logische Operationen

Der Computer hat zudem die Fähigkeit, arithmetische und logische Operation durchzuführen. Alle Operationen sind dabei binär daher Kombinieren zwei Eingangswerte oder Operanden zu einem Ausgangswert. Die Ausnahme macht dabei die NOT-Instruktion, die nur einen Eingangswert annimmt und auf diesen ein Bitweises Nicht durchführt.

Alle arithmetischen und logische Operationen funktionieren dabei immer gleich: Sie kombinieren das erste Quelle-Register mit dem zweiten Quellen-Register und schreiben das Ergebnis in das Ziel-Register. Dabei ist zu beachten, dass das zweite Quellen-Register durch einen konstanten Wert ersetzt werden kann. Die Notation dafür sie wie folgt aus:

<Operation> <Ziel-Register> <Quellen-Register 1> <Quellen-Register 2/Konstante>

Der Computer unterstützt dabei folgende Operationen: Addition (ADD), Subtraktion (SUB), Multiplikation (MUL), Division (DIV), Bitweises exklusives Oder (XOR), Bitweise Und (AND), Bitweises Oder (OR), Bitweises Nicht (NOT).

Wichtig zu sagen ist, dass bei alle Operationen ($f(x_1, x_2) \equiv y \mod 65536$) gilt $65535 \geq$

$y \geq 0$ ist wo bei gilt $y \in \mathbb{N}$. Der springende Punk ist, dass bei einer Rechnung wie 65535+1 das Ergebnis nicht 65536 ist, sondern 0. So ein "Overflow" kann erkannt werden durch die Ausführung der TEST-Instruktion für Addition und Subtraktion. Diese Setzt bzw löscht bei Subtraktion und Addition, die entsprechenden Flaggen OFadd und OFsub.

Bei Addition ist das schlimmste was passieren kann ein Overflow von 1 ($65535 + 65535 = 131070 \equiv 65534 \mod 65536$). Bei Subtraktion werden negative Zahl im Zweierkomplement System dargestellt. Das Schlimmst was passieren kann ist daher ein Overflow, oder vielmehr Underflow von 1 ($0 - 65535 = -65535 \equiv 1 \mod 65536$)

Bei der Multiplikation von zwei Zahlen ist dies jedoch komplexer, da in diesem Fall ein Overflow zwischen 0 und 65534 haben kann ($65535 * 65535 = 4294836225 \equiv 1 \mod 65536$). Daher gibt es eine MULOF um den bei der Multiplikation entstanden Overflow aufzufangen. Die Notation ist dabei wie folgt:

MULOF <Ziel-Register> <Quellen-Register 1> <Quellen-Register 2/Konstante>

2.4 Memory

Während die Register für lokale Variablen und Zwischenergebnisse gedacht sind, ist Memory (Arbeitsspeicher) für die Speicherung von größeren Datenmengen gedacht.

Es ist in 65536 einzelne und unabhängige Speicherzellen (Jede 16 Bit große) aufgeteilt. Jeder Zelle ist einer Adresse, welche von 0 bis 65535 reicht, zugeordnet. Der Computer kann damit auf 65536 words oder 131072 bytes ($\approx$ 128 kib) zugreifen [1]. Wichtig ist, dass die gesamt Größe, dank Banking siehe Unterunterabschnitt 2.4.1 theoretisch weitaus größer sein kann.

Memory wird dabei über LD- und STR-Instruktionen gesteuert. Die LD-Instruktion kopiert einen Wert aus Memory und schreibt ihn in ein Ziel-Register, während STR an der angegebenen Adresse den Wert mit dem Wert des Quellen-Register überschreibt. Die Adresse wird zusammen gesetzt durch die Addition des Address-Register1 und entweder in dem Address-Register2 oder einer Konstante. Die Notation sieht dabei wie folgt aus:

LD <Ziel-Register> <Address-Register1> <Address-Register2/Konstante>

STR <Quellen-Register> <Address-Register1> <Address-Register2/Konstante>

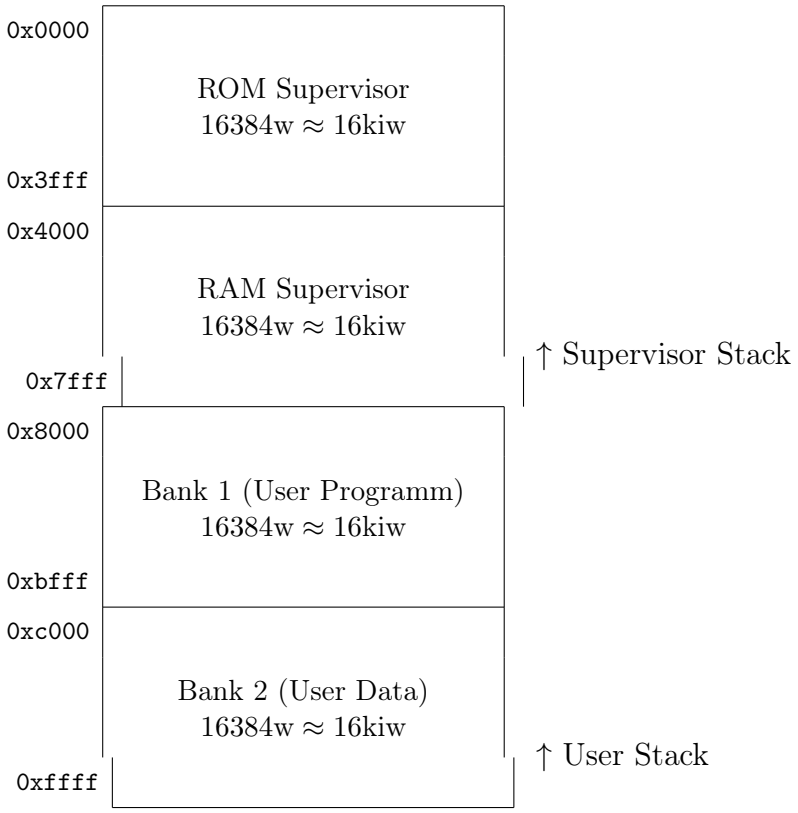
Zudem wurden bestimmte Bereichen bestimmte Aufgaben bzw. Funktionen zugewiesen, wie in Abbildung 1 zu sehen. Die zwei Hardware Stacks sind dort gekennzeichnet mit ↑,

[1] Die Größe messen wir in words (w) oder in kilowords (kiw), wobei 1 word = 2 bytes = 16 Bit.

mehr dazu Stacks in Unterunterabschnitt 2.4.2

Die mit Supervisor gekennzeichneten Bereich können nur dann gelesen oder geschrieben werden, wenn sich der Computer im Supervisormode befindet. Mehr zu Supervisormode in Unterabschnitt 2.2. Der Bereich, der mit ROM versehen wurde, ist dabei nicht beschreibbar, sondern kann nur gelesen werden. Eine STR-Instruktion hat keine Wirkung. Eine Änderung des Inhalts der ROM Bereiche ist nur durch den Austausch/Externes Programmieren des darunterliegen physikalischen ROM-Chips möglich.

Abbildung 1: Memory Karte



2.4.1 Banking

Banken sind externe Memory Module, die eingesteckt werden können. Alle Banken sind gleich groß (16kiw = 32 kib) und der Computer unterstützt maximal 65536 unterschiedliche Banken. Jede Bank hat dabei eine ID (BID), die von 0 bis 65535 reicht.

Der Computer hat zwei Bank-Slots (Slot 1 `0x8000 - 0xbfff` und Slot 2 `0xc000 -`

`0xffff`). Er kann nun auf jede dieser Slots eine der theoretisch 65536 Bank legen [2] Das Banken legen lässt sich über die SBK1- (Select Bank Slot 1) und SBK2- (Select Bank Slot 2) Instruktion erreichen. Die BIN kann dabei als Konstante oder als Register vorliegen. Die Notation ist dabei wie folgt:

SBK<1/2> <BIN-Register/Konstante>

Somit führt das Beschreiben/Lesen an einer Adresse welche einem Bank-Slot zugeordnet ist, zum Beschreiben/Lesen der ausgewählte Bank.

Durch dieses System lassen sich $65536 \cdot 16\text{kiw} = 1\ \text{Giw} = 2\ \text{Gib}$ an Arbeitsspeicher ansprechen.

Das Bankensystem dient zudem dazu, Prozesse voneinander zu isolieren, da der Computer im Usermode die Bank nicht verändern kann, siehe Unterabschnitt 2.2.

2.4.2 Hardware Stack

Der Stack (oder, auch Stapelspeicher) ist eine LIFO (Last In First Out) Datenstruktur. Sie dient hauptsächlich dazu, Subroutinen zu verwalten.

Er funktioniert dabei wie ein Papierstapel. Es kann ein Wert auf den Stack gelegt, werden (PUSH-Instruktion) oder ein Wert von oben abgehoben werden (POP-Instruktion). Wobei der Wert, der auf den Stack gelegt bzw. abgehoben werden soll, entweder im Quellen-Register bzw. Zeil-Register oder als Konstante vorliegt. Die Notation ist dabei wie folgt:

PUSH <Quellen-Register/Konstante>

POP <Ziel-Register>

Der Stack liegt dabei physisch in Memory vor. Jedes Element des Stacks okkupiert dabei eine Memory-Zelle (16 Bit). Das Stack-Point Register-(SP) hält die Adresse, welche die nächste freie Memory-Zelle zeigt, die eins über dem obersten Wert des Stacks liegt. Der Stack wächst dabei von hohen Adressen zu niedrigen Adressen, wie in Abbildung 2 zusehen.

Bei einer PUSH Instruktion wird zuerst an die Stelle auf die SP zeigt der Wert des Quellen-Register/Konstante geschrieben und darauf der SP um 1 dekrementiert. POP

[2] Es ist dabei möglich, die gleiche Bank auf beide Slots zu legen, auch wenn das nur bedingt nützlich ist.

dagegen inkriminiert zuerst den SP um 1 und kopiert dann den Wert von der Stelle auf welche SP zeigt in das Ziel-Register.

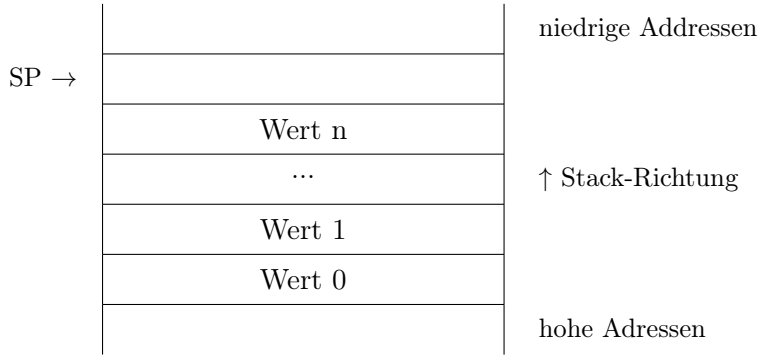
Zudem gibt es noch die RET- und CALL-Instruktionen. RET ist äquivalent zu (POP IP) und CALL ist äquivalent zu PUSH IP; MOV IP <Quellen-Register/Konstante>. Die Notation ist dabei wie folgt:

RET

CALL <Quellen-Register/Konstante>

Fundamental hat der Computer zwei Stacks. Der Supervisor-Stack beginnt bei `0xc7fff` (kleiner werdend) und wird genutzt, wenn die SUP Flagge gesetzt ist. Der User-Stack beginnt bei `0xffff`, also am Ende des Bank Slots 2 und wird genutzt, wenn die SUP Flagge nicht gesetzt ist.

Abbildung 2: Stack Aufbau



2.5 Input/Output

Der Computer hat zudem die Möglichkeit mit verschieden IO-Geräte zu interagieren. Die Kommunikation läuft über die sogenannten IO-Ports. Der Computer hat physische Anschlüsse an denen IO-Geräte angesteckt werden können. Die Struktur dieser Anschlüsse sieht wir folgt aus:

Tabelle 4:

#	Funktion
00	GND
01	Vcc
02-17	16Bit Daten
18	Iin-Enable
19	Din-Enable
20	Dout-Request
21	Dout-Accept
22	CLK_R

2.5.1 Kommunikation zwischen IO-Geräten und dem Controller

Die Interaktion zwischen IO-Controller und dem IO-Gerät funktioniert dabei über drei Instruktionen, wie in Tabelle 5 zusehen ist.

Tabelle 5:

kurzform	Ausgeschrieben	funktion
Iin	Instruktion Eingang	Computer schreibt eine Instruktion ans I/O-Gerät
Din	Daten Eingang	Computer schreibt 16-Bit Daten ans I/O-Gerät
Dout	Daten Output	Computer liest 16-Bit Daten vom I/O-Gerät

Jede Interaktion benötigt dabei mehrere Schritte, die genaue Definition der Kommunikationsschritte wird dabei durch jede instruktion eigen festgelegt. Ein allgemeiner standard Ablauf (wie z.B. der der später definierten IO-Instruktion "rmn") ist in Tabelle 6 zusehen.

Tabelle 6:

Bereich	Clock 1	Clock 2	...	Clock n	Clock $n+1$	...
Iin	1	0		0		
Din	0	1		0		
Dout	0	0		1		
Data$0-7$	InstructionID	Variable			Rückgabewert	
Data$8-15$	8b Variable	Variable			Rückgabewert	

Die Pins Iin und Din werden dabei durch den Computer für einen Clock-Tick gesetzt, sobald die Daten/Instruktionsinformation am Dateneingang an liegen; Wenn das IO-Gerät etwas zu dem Computer senden will, dann setzt es Dout-Request und der Computer nimmt darauf mit dem Setzten von Dout-Accept an, worauf hin das IO-Gerät die Daten zu dem Computer übermittelt.

2.5.2 Ansprechen des Controllers im Code

das Oben genannte verhalten ist die Kommunikation nachdem der IO-Controller die Instruktionen übersetzt hat, dafür muss der Controller wie folgt angesprochen werden:

Um Etwas zu dem IO-Gerät zu schreiben müssen folgende schritte im Programm stehen: Tabelle 7

Tabelle 7:

Bereich	Clock 1	Clock 2	...	Clock n	Clock $n+1$	...
InsName	Iout	Dout		Dout	Dout	
Variable 1	Geräte-ID	Geräte-ID		Geräte-ID	Geräte-ID	
Var2 / Konstante b0-7	IO-Befehl	8b Variable		8b Variable	8b Variable	
Var2 / Konstante b8-15	8b Variable	8b Variable		8b Variable	8b Variable	

Der InsName ist hier der Name aus den Allgemeinen Instrukionen, wie in Tabelle 1 definiert, Iout/Dout definiert, welcher der obigen Iin/Din pins gesetzt wird. Die Geräte-ID ist die ID, mit der sich das IO-Gerät bei dem Controller meldet und die vom Controller auf den richtigen Anschluss aufgelöst wird. Die Variable 2 oder Konstante wird vom IO-Controller komplett unverändert an das IO-Gerät weiter gegeben.

2.5.3 Standart IO-Instruktionen

Die Momentan bestehenden I/O-Instruktionen sind in Tabelle 5, aber diese sind im Programm frei veränderlich, solange das externe Gerät die gleiche Interpretation der Anweisungen hat.

Tabelle 8: I/O-Instruktionen

ID	Gerätetyp	Name	Funktion	Verhalten
0x00	All	ls	gibt die ID des Gerätes zurück	Iin(Instruktion); n∗Dout(Antworten)
0x01	Drive	rsn (read singel normal)	Lesen eines einzelnen 16-Bit werts von einer 24-Bit Addresse	Iin(Instruktion); Din(Addresse); 1∗Dout(Antwort)
0x02	Drive	rmn (read multiple normal)	Lesen einer reihe von bis zu 65535 16 Bit Werten von 24-Bit Addressen aus	Iin(Instruktion); Din(startAddresse); Din(Länge); n∗Dout(Antworten)
0x03	Drive	wsn (write singel normal)	Schreiben eines einzelnen 16 Bit werts von einer 24-Bit Addresse	Iin(Instruktion); Din(Addresse); Din(Daten); Dout(bestätigung)
0x04	Drive	wmn (write multiple normal)	Schreiben einer reihe von bis zu 65535 Werten an 24-Bit Addressen	Iin(Instruktion); Din(startAddresse); n∗Din(Daten); Dout(bestätigung)
0x05	Drive	rsb (singel read big)	Lesen eines einzelnen 16-Bit werts von einer 40-Bit Addresse	Iin(Instruktion); Din(Addresse); Din(Addresse); 1∗Dout(Antwort)
0x06	Drive	rmb (multiple write big)	Lesen einer reihe an von bis zu 65535 Werten von 40-Bit Addressen aus	Iin(Instruktion); Din(startAddresse); Din(startAddresse2); Din(Länge); n∗Dout(Antworten)
0x07	Drive	wmb (singel write big)	Schreiben eines einzelnen 16-Bit werts von einer 40-Bit Addresse	Iin(Instruktion); Din(Addresse); Din(Addresse);Din(Daten); 1∗Dout(bestätigung)
0x08	Drive	wmb (write multiple big)	Schreiben einer reihe von bis zu 65535 Werten 16 Bit werten an 40-Bit Addressen	Iin(Instruktion); Din(startAddresse); n∗Din(Daten); Dout(bestätigung)
0x09	Drive	rfn (read file normal)	Lesen einer gesamten Datei (24-bit Namen)	Iin(Instruction); Din(Name); n∗Dout(Antwort)

Tabelle 8: I/O-Instruktionen

0x0a	Drive	wfn (write file normal)	Schreiben einer neuen Datei (24-bit Namen)	Iin(instruction); Din(Name); n∗Din(daten); Dout(bestätigung)
0x0b	Drive	rfpb (read file-part big)	Lesen eines teils einer Datei (24-bit Namen)	Iin(instruction); Din(Name); 2∗Din(läge); n∗Dout(Daten)
0x0c	Drive	wfpb (write file-part big)	Anhängen von Daten an eine Datei(24-bit Namen)	Iin(instruction); Din(Name); Din(länge); n∗Din(Daten); Dout(bestätigung)

2.5.4 Berechtigungsliste

Der Supervisor-mode hat die berechtigungen zu allen I/O-Interaktionen, für normale Prozesse wird die Berechtigungsliste in einer Reihe an RAM-Chips gespeichert, deren Adressen sich mit dem wechseln zwischen den verschiedenen Banks ändern. Gesetzt werden können sie nur durch den Supervisor, wobei auch dieser die RAM-Adressen nicht manuel ändern kann.

2.6 Interrupts

Interrupts sind Unterbrechungen im Programmablauf. Wenn ein Interrupt gemeldet wird, wird dieser im nächsten Instruktions-Zyklus bearbeitet. Für diesen Zyklus, wird, je nach Interrupt-Quelle, nicht die Instruktion an der Stelle von IP sondern von einem IV (Interrupt Vektor) geladen. Es gibt im Computer vier verschieden Interrupt Quellen, wobei jede ein fest-verbauten IV hat, dies ist in Tabelle 9 zusehen.

Interrupts können generell (sowohl im Usermode als auch im Supervisormode) deaktiviert werden, mit Ausnahme von Software-Interrupts, durch das setzen der OINT Flagge (nur vom Supervisor). Für ein feinere Einstellung gibt es die OVIOI, OVTI1, OVTI2 Flaggen welche wie in Tabelle 9 zusehen, einen speziellen Interrupt deaktivieren. Diese Flaggen, übernehmen dies aber nur im Supervisor. Alle diese Flaggen befinden sich übrigens in

den letzten 8 Bit des FLAG Register, womit sie nur im Supervisormode verändert werden kann.

Die INT Instruktion löst ein Software-Interrupt aus. Dieser kann nicht deaktiviert werden. Diese Instruktion kann sowohl im Supervisor und im Usermode ausgeführt werden. Die Instruktion nimmt keine Operntanten an. Die Notation ist dabei wie folgt:

INT

Tabelle 9:

Interrupt-Quelle	IV Name	IV Standard Wert	Flagge zum Deaktivieren
IO-Interrupts	IV-IO	`0x3fff8`	OVIOI
INT-Instruktion	IV-INT	`0x3fffa`	
Timed-Interrupts	IV-TI1	`0x3fffc`	OVTI1
Timed-Interrupts	IV-TI2	`0x3fffe`	OVTI2

2.6.1 Timed-Interrupts

Der Computer besitzt zwei interne Timer je 32 Bit groß. Jeder Timer started bei 0 und zählt (bei jedem Clock Schlag) um 1 bis zu seinem eingestellten End Wert. Bei erreichen dieses Werts löst der Timer einen dementsprechenden Interrupt aus.

Jeder Timer kann in vier Modi Arbeit, wie in Tabelle 10 zusehen. Der Timer Modus kann über die TM1 und TM2 (steht für Timer1 Mode, Timer2 Mode) Instruktion gesetzt werden, welcher entweder ein Quellen Register oder eine Konstante an nimmt, wobei die Notation wie folgt aus sieht:

TM<1/2> <Quellen-Register/Konstante>

Der Endwert wird über die SSTL1, SSTL2 und SSTH1, SSTH2 erledigt. Wobei SSTL<1/2> die ersten 16 Bit und SSTH<1/2> die letzten 8 Bit des Endwert-Registers setzt. Der Endwert kann dabei als Konstante oder als Quellen-Register vorliegen. Die Notation ist dabei wie folgt:

SSTL<1/2> <Quellen-Register/Konstante>

SSTH<1/2> <Quellen-Register/Konstante>

Tabelle 10:

Timer Modus	Modus ID	Funktion
Suspended	`0x0`	Timer zählt nicht hoch, behält aber sein aktuellen Wert. Das Hochzählen wird fortgesetzt sobalt der Timer in Continues oder Singel Mode gesetzt wird
Reset	`0x1`	Timer ist Inaktive und der Zähler wird auf 0 gesetzt
Continues	`0x2`	Timer zählt hoch; bei ereichen des Ziels, löst Interrupt aus und fängt von vorne an zuzählen.
Singel	`0x3`	Timer zählt hoch; bei ereichen des Ziels, löst Interrupt aus und schalte in Reset Modus um

Die aktuell Clock Geschwindigkeit, ist in mit CFL und CFH instruktion zu bekommen. CFL lädt die ersten 16 bit der Clock Geschwindigkeit in das Ziel-Register und CFH die letzten 8 bit in das Ziel-Register. Die Frequenz ist dabei In Herz (Schläge pro Sekunde) angegeben. Die Notation ist wie folgt:

CFL <Ziel-Register>

CFH <Ziel-Register>

2.6.2

Auch IO-Geräte können als Interrupt-Quelle fungieren. Dabei gibt es keine Möglichkeit, vom Computer her Interrupt eines IO-Geräts zu deaktivieren oder aktivieren. Dies muss gegebenenfalls über das IO-Protokoll, siehe Unterabschnitt 2.5, gemacht werden. Es können natürlich alle IO-Interrupts auf einmal mit der OVIOI-Flagge gemacht werden.

3 Kommentar zur Specification

3.1 16 bit Daten/Address Länge

Die Address- und Datenlänge besagt, wie viele Binäre Ziffern (bits) für eine Address und Daten verwendet werden. Je mehr Ziffern man verwendet desto größere Zahlen können in einer Operation verarbeitet werden wobei die maximal repräsentierbare Zahle $2^n - 1$

ist wenn n die Anzahl der Binären Ziffern (= Bits) ist und wenn man mit 0 anfängt zu zählen. Siehe Tabelle 11

Ein eine n stellige Binär Zahl wobei n teilbar durch 8 ist, ist dabei relative angenehm, 8 bit = 1 byte. Das Byte ist dabei die basis Einheit für so gut wie alle informationstechnischen Standards. Daher ist es gute eine solches n zu wählen. Zudem kommen einige elementar ICs, z.B. Buffer oder Register, immer mit 8 bits.

Jedoch gilt je mehr Bits man verwendet, desto mehr Schaltung brauchen man, desto mehr Stromverbrauch. Da wir wie zwar nicht speziell auf Platz und Stromverbrauch optimieren, aber trotzdem ein limitiertes Budget haben, können wir auch nicht eine beliebig große Daten/Address Länge wählen.

Wir haben uns daher für 16 bit entschieden, der laut Tabelle 11 Zahlen bis zwischen 0 und 65535 darstellen kann. Diese Menge ist gerade große genug, um für die meisten Programme große genüge Zahl abzubilden [3].

Die Entscheidung zur größe der Daten/Address Länge ist sehr ausschlaggebend, daher haben wir diese zuerst festgelegt. Sie wird für viele weitere Entscheidung eine wichtige Rolle spielen. Wichtig ist nur das wir mit unseren 16 bit maximal 65536 Zustände oder Zahlen von 0 bis 65535 darstellen können.

Tabelle 11: Maximale repräsentierbare Zahlen (startend mit 0)

n	$n^2 - 1$
2	3
4	15
8	255
16	65535
32	4294967295
64	18446744073709551615

3.2 Register Anzahl

Der Computer hat 8 Register. Wir haben uns für Acht aus folgenden Gründen entschieden: Zu erst die minimale Anzahl an Registern die wirklich gebraucht würde, ist 5! Man

[3] Für Programme die Zahlen welche > 65535 darstellen wohlen ist es ratsam immer zwei oder mehr Speichereinheiten zunehmen oder bei Berechnungen, selbige auf zwei oder mehr Schritte auf zuteilen

braucht mindestens zwei allzweck Register um binäre Rechenoperation durch zu führen, einen Stackpointer, einen Instruktion Pointer und einen Ort wo man die Flaggen speichern kann. [4].

Fünf lässt sich nicht mehr mit 2 bits repräsentieren ($2^2 = 4$), daher muss man 3 bit ($2^2 = 8$) nehmen. Der Grund warum wir nicht mehr genomen haben hat mit der Encodierung von Instruktionen zutun siehe **??**. Somit haben sich 8 Register als das Optimum ergeben

3.3 Gedanken Hinter dem Supervisormode

Wir haben uns entschieden eine Art Berechtigungssystem in der Computer einzubauen. Eine Rechner der nur einfache Programm ausführt um zu Beispiel eine LED oder ein LCD-Display zusteueren, wie es z.B. ein ESP32 oder ein ATMega328p macht, ist so ein System nicht nötig. Wir hatten jedoch vor mit unserem Computer die Möglichkeit zu schaffen ein Betriebssystem ähnliches Programm (Supervisor) laufen zulassen.

Hier für hat man ein mal den Supervisor, der bei Start des Computer anfängt zulaufen, und Programme, welche im Usermode ausgeführt werden, sogenannte Prozesse. Dabei sieht ein typischer Programmablauf ist dabei in gezeigt.

Wichtig ist, das sobald ein Prozess beendet wurde, der Supervisor an dieser Stelle wieder anfängt und den nächsten Prozess started.

Ein Prozess kann auch Funktionen des Supervisor nutzten über Software Interrupts, da bei einem Interrupt, der Computer in den Supervisormode wechselt. So kann der Prozess, bevor er den Interrupt auslegt, ein oder mehrere Register nutzten um Werte an den Supervisor zu übergeben. Z.B. kann er eine Register nutzten um dem Supervisor zusagen was er tun soll, z.B. dem Prozess zugang zu einem IO-Gerät zugeben oder etwas in eine Datei zuschreiben. Diese Art von Kommunikation werden auch System-Calls genannt und sind die Basis für jedes moderne Betriebssystem.

Ein solcher Supervisor kann sehr mächtig sein:

1. Abstraktion der Hardware: Oft ist die IO-Configuration von Setup zu Setup sehr verschieden. Ein Prozess was IO-Nutzen möchte hat oft den Nachteil dass es für ein bestimmte Hardware setup geschrieben wurde. So z.B. ein Prozess was Daten einer SD-Karte ausließt und diese anzeigt. Wenn jedoch man jetzt eine Floppy-Laufwerk anschließt, so kann man das Prozess zum Auslesen nicht verwenden, da es nur die

[4] Theoretisch ist es möglich das Flaggen Register wegzulassen, wir haben uns aber für eine solches Register entschieden siehe **??**

Kommuntiation mit einer SD-Karte kennt. Der Supervisor könnte an dieser Stelle ein Interface über System-Calls bieten welches unabhänige von dem Gerät ist. So im diesem Beispiel ein Dateisystem, auf welches Prozesse zugreifen können. Somit übernimmt der Supervisor die Kommuntiation mit der Hardware.

2. Teilung der Rechenzeit: Der Computer so viele Features er doch hat, ist stark limitiert vor allem was Rechengeschwindigkeit betrifft. Wenn man nun mehrer Programm hat, welche ausgeführt werden sollen, so wäre es natürlich sehr einfach sie der Reihe nach auszuführen. Jedoch gibt es bessere Wege, so z.B. das der Computer jedem Prozess nur eine Bestimmte Rechenzeit gibt und dann den nächsten Prozess ausführt. Das kann erreicht werden durch folgenden Ablauf: Der Supervisor setellt vor dem ausführen des Prozesse eine Timed-Interrupt. Dann übergibt der Supervisor den Computer dem Prozess, welcher dann seine Programm ausführt. Dies machter bis der Timed-Interrupt ausgelöst wird und der Computer wieder in den Supervisormode wechselt. Dieser speichert nun alle Register und Banken Configuration des Prozesse und übergibt einem anderen Prozesse den Computer. Wenn nun ein schon angefanger Prozess wieder gestarted werden soll läd der Supervisor wieder die Register und Banken des Prozesse und übergibt die Außführung wieder an den Prozess. Somit kann der Computer zwischen Prozessen hin und herschallten, was die Illusion gibt, das der Computer diese gleichzeitig ausführt.

3. Isolation von Prozessen: Programme sind meist nie perfekt. Vor allem sowas wie Bufferoverflows und andere Fehler dieser Art tretten in Low-Level Systeme gerne auf. Aus diesem Grund ist es wichtig Prozesse von einander zu isolieren. Nehmen wir an Prozess A hat ein Fehler welcher dafür sorgt das er den gesammten RAM anfängt voll zuschreiben. Um jetzt Prozess A daran zuhinern Prozess B zu überschreiben, ist es dem Prozess nicht gestattet auf den Arbeitsspeicher von Prozess B zuzugreifen. Nur der Supervisor kann zwischen den Prozessen wechseln [5].

Wenn man so etwas will kann dass auch auf einem ATMega328p gemacht werden, jedoch nicht ohne massive Leistungsverluste. Wenn so ein Berechtigungssystem jedoch direkt in die Hardware des Computers einfließt sorgt das für eine viel effektiveren Computer.

Wenn man tatsächliche nur Programme schreiben will welche ürëin LED oder LCD-Display steuern soll, kann man das mit unserem Computer auch machen, indem man einfach diese Programm im Supervisor laufen lässt.

[5] Natürlich kann auch der Supervisor fehlerhaft sein. Jedoch hatten wir bei dieser Maßnahme nicht nur fehlerhaft sondern auch schlicht böswillige Prozesse im Kopf. Natürlich ist dies unwahrscheinlich dass so etwas jemals eintreten wird, jedoch war unsere Idee von Anfang an eine Computer zu bauen der im Internet stehen könnte und die dementsprechenden Sicherheitsmaßnahmen hat

3.4 Zur fehlenden Deaktivierung der Software Interrupts

Die INT-Instruktion löst wie in Unterabschnitt 2.6 einen Software Interrupt aus welcher nicht durch die OINT- Flagge deaktiviert werden kann.

Um die Motivation hinter dieser Entscheidung zu verstehen ist zu wissen, dass der Software Interrupt dazu gedacht ist um aus dem Usermode in den Supervisormode zu wechseln. Und zwar in einem kontrollieren Weg, sodass es nicht möglich ist beliebigen Programm-Code als User mit Supervisor Privilege auszuführen. Daher wird auch nicht direkt nachdem Ausführen der INT-Instruktion an dieser Stelle weiter ausgeführt sonder Programm-Code welcher an der Position von einem IV sich befindet, welcher nicht verändert werden kann und schluss endlich irgendwelchen Supervisor Code ausführt.

Dabei ist dies der einzige Weg, nebst den Computer zu resetten um Code im Supervisormode auszuführen aus der Position des Usermodes. Wenn es nun die Möglichkeit gäbe Software Interrupts zu deaktivieren und man in den Usermode wechseln würde, kann man nicht mehr zurück in den Supervisormode kommen, da im Usermode keine Interrupts aktivieren oder deaktiviert werden können. Von daher ist es Sinnbefreit dem Computer die Möglichkeit zugeben Software Interrupts zu deaktivieren.

3.5 Zuordnung des IO-Slot zu IO-Type

Bei den IO-Geräten wird momentan als ID die ID des Anschlusses verwendet und der Supervisor hat die Aufgabe über die ls-funktion die Anschluss-IDs auf die Geräte-IDs zu mappen. Idealer weise würde dies un dem IO-Controller umgesetzt werden, aber da würde es für uns momentan zu kompliziert, weshalb wir uns entschieden haben das in der Software zu machen.