

Platinencomputer - Handbuch

Alexander Wersching und Simon Walter

2021

Inhaltsverzeichnis

1	Einführung	3
1.1	Ziel diese Dokuments	3
1.2	Geschichte	3
2	Spezifikation des Computers	3
2.1	Registers	4
2.1.1	Kopieren und Bewegen von Daten	4
2.1.2	Flaggen	5
2.1.3	Arithmetische Operationen und Bitmanipulation	6
2.2	Memory	7
2.2.1	Banking	7
2.2.2	Hardware Stack	9
2.3	Input/Output	10
2.3.1	Berechtigungsliste	13
2.4	Interrupts	13
2.4.1	Timed-Interrupts	14
2.4.2	IO-Interrupts	14
2.5	Supervisor-/Usermode	14
2.6	Instruktions-Satz	15
3	Kommentar zur Specification	15
3.1	16 bit Daten/Address Länge	15
3.2	Register Anzahl	17
3.3	Alternative Instruktion Encodierung	18
3.4	Gedanken Hinter dem Supervisormode	18
4	Quellenverzeichnis	18

1 Einführung

1.1 Ziel diese Dokuments

Dieses Dokument soll die Funktionsweise unseres Platinencomputers beschreiben. Dabei soll nicht auf die tatsächliche auf Implementation wertgelegt 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 kann 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 diene.

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 Neuman-Maschine, d.h. sowol Daten als auch Programm während in einem Hauptspeicher (Memory) gespeichert

2.1 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* (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 1 zu sehen.

Mehr zur Registeranzahl in Unterabschnitt 3.2

Tabelle 1: 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 2.2.2
3	A	Allzweck Register ohne besonderen Aufgaben
4	B	Allzweck "
5	C	Allzweck "
6	D	Allzweck "
7	FLAG	Enthält alle wichtigen Flaggen, siehe Unterunterabschnitt 2.1.2

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.

2.1.1 Kopieren und Bewegen von Daten

Das Kopieren 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> <Qullen-Register/Konstante>

Eine Übersicht über alle Instruktion ist in Unterabschnitt 2.6 vermerkt.

2.1.2 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 2 zusehen.

Tabelle 2: Flaggenaufteilung in FLAG

Bit	Name	Funktion	Schaltung
0	OF	Intager Overflow einer Rechenoperation	Bei jeder ALU Operation
1	E	Gleichheit zweier Werte	TEST arg[0] = arg[1]
2	G	Ungleichheit (>) zweier Werte	TEST arg[0] > arg[1]
3	L	Ungleichheit (<) zweier Werte	TEST arg[0] < arg[0]
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15	OINT	Override Interruption, siehe Unterabschnitt 2.4	
15	SUP	Supervisor-/Usermode, siehe Unterabschnitt 2.5	

Alle Flaggen, die mit TEST als Schaltung gekennzeichnet sind, werden bei der Ausführung der TEST Operation geschaltet.

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 Supervisor-Mode befindet, siehe Unterabschnitt 2.5. Ein Schalten im User-Mode hat keinen Effekt.

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), MOF (OF Flagge), MG (G Flagge), ML (L Flagge), MI (ONIT Flagge), MS (SUP Flagge).

Eine Übersicht über alle Instruktionen ist in Unterabschnitt 2.6 vermerkt.

2.1.3 Arithmetische Operationen und Bitmanipulation

Der Computer hat zudem die Fähigkeit, arithmetische Operation und Bit-Manipulation 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 Operationen und Bit-Manipulation funktionieren dabei immer gleich: Sie kombinieren das erste Quell-Register mit dem zweiten Quell-Register und schreiben das Ergebnis in das Ziel-Register. Dabei ist zu beachten, dass das zweite Quell-Register durch einen konstanten Wert ersetzt werden kann. Die Notation dafür sieht 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)

Eine Übersicht über alle Instruktionen ist in Unterabschnitt 2.6 vermerkt.

Wichtig zuzusagen ist bei allen Operationen ($f(x_1, x_2) = y \bmod 65536$) gilt $65535 \geq y \geq 0$ ist wo bei gilt $y \in \mathbb{N}$. Der springende Punkt ist bei, dass bei einer Rechnung wie $65535 + 1$ das Ergebnis nicht 65536 ist, sondern 0. Bei so einem "Overflow" wird die OF-Flagge gesetzt. Dieses Problem tritt auf da wir nur 16 Bit als Datenlänge zu Verfügung haben, mehr dazu in Unterabschnitt 3.1

2.2 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 ($\approx 131.0\text{kb}$) zugreifen.¹ Wichtig ist das die Gesamt Größe theoretisch weitaus größer sein kann, dank Banking siehe Unterunterabschnitt 2.2.1.

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>

Eine Übersicht über alle Instruktion ist in Unterabschnitt 2.6 vermerkt.

Zudem wurden bestimmte Bereichen bestimmte Aufgaben bzw. Funktionen zugewiesen, wie in Abbildung 1 zu sehen. Die zwei Hardware Stacks sind dort gekennzeichnet mit \uparrow , mehr dazu Stacks in Unterunterabschnitt 2.2.2

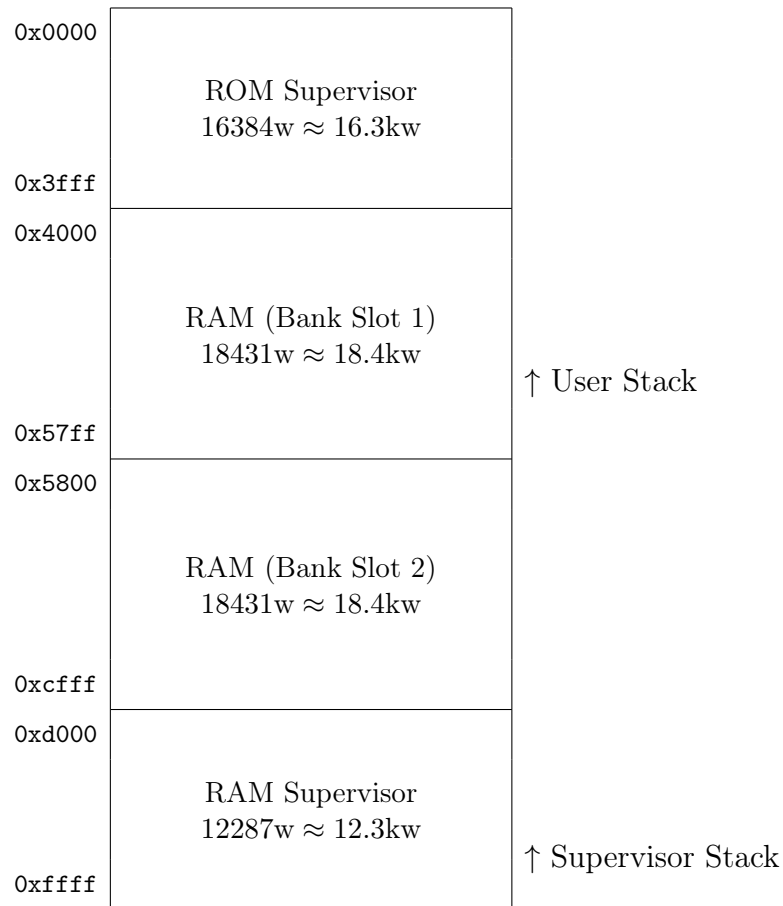
Die mit Supervisor gekennzeichneten Bereich können nur dann gelesen oder geschrieben werden, wenn sich der Computer im Supervisormode befindet. Mehr zu Supervisormode in Unterabschnitt 2.5. Der Bereich, der mit ROM versehen wurde, ist dabei nicht beschreibbar, sondern kann nur gelesen werden. Eine STR-Instruktion hat keine Wirkung. Eine Änderung des Inhalts der ROM Bereiche ist nur durch den Austausch des darunterliegenden physikalischen ROM-Chips möglich.

2.2.1 Banking

Banken sind externe Memory Module, die eingesteckt werden können. Alle Banken sind gleich groß ($18.4\text{kw} = 36.8\text{kb}$) und der Computer unterstützt maximal 65536 unterschiedliche Banken. Jede Bank hat dabei eine ID (BID), die von 0 bis 65535 reicht.

¹Die Größe messen wir in words (w) oder in kilowords (kw), wobei $1\text{ word} = 2\text{ bytes} = 16\text{ Bit}$.

Abbildung 1: Memory Karte



Der Computer hat zwei Bank-Slots (Slot 1 0x4000 - 0x3fff und Slot 2 0x4000 - 0x57ff). Er kann nun auf jede dieser Slots eine der theoretisch 65536 Bank legen ². Das Banken legen lässt sich über die SBK1- (Select Bank 1) und SBK2- (Select Bank 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>

Eine Übersicht über alle Instruktion ist in Unterabschnitt 2.6 vermerkt.

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

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

Durch dieses System lassen sich $65536 * 18.4\text{kw} = 1.2\text{Gw} = 2.4\text{GB}$ an Arbeitsspeicher ansprechen.

Das Bankensystem dient zudem dazu, Prozesse voneinander zu isolieren, da im Usermode der Computer die Bank nicht verändern kann, siehe Unterabschnitt 2.5.

2.2.2 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. Ziel-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 inkriminiert. POP dagegen dekrementiert 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:

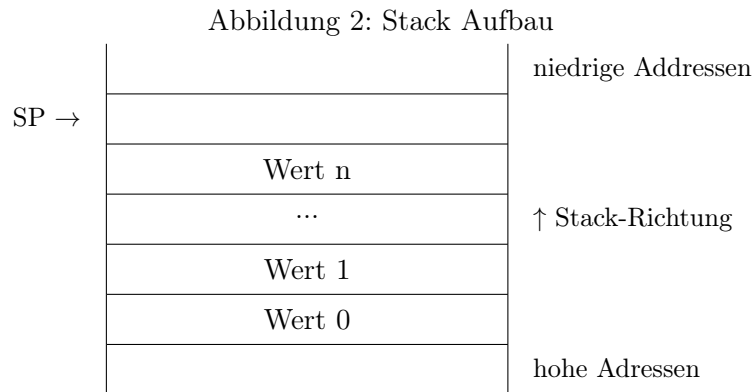
RET

CALL <Quellen-Register/Konstante>

Fundamental hat der Computer zwei Stacks. Der Supervisor-Stack beginnt bei `0xffff` und wird genutzt, wenn die SUP Flagge gesetzt ist. Der User-Stack beginnt bei `0x57ff`,

also am Ende des Bank-Slots 1 und wird genutzt, wenn die SUP Flagge nicht gesetzt ist.

POP, PUSH, CALL und RET schalten automatisch zwischen Supervisor-Stack und User-Stack bei Änderung der SUP-Flagge.



2.3 Input/Output

Der Computer hat zudem die Möglichkeit mit verschiedenen 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 3:

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

Dabei bekommt jedes IO-Gerät eine eigene ID (IO-ID), welche im Bereich von 0 bis 65535 liegen darf, wobei davon aufgrund von physikalischen beschränkungen vermutlich nur die ersten -zig verwendet werden. Die Interaktion funktioniert dabei über drei Instruktionen:

Tabelle 4:

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") sieht wir folgt aus:

Tabelle 5:

Bereich	Instr 1	Instr 2	...	Instr n	Instr $n + 1$...
Iin	1	0		0		
Din	0	1		0		
Dout	0	0		1		
Data0 – 7	InstructionID	Variable		Rückgabewert	Rückgabewert	
Data8 – 15	8b Variable	Variable		Rückgabewert	Rückgabewert	

Die Pins Iin und Din werden dabei durch den Computer für einen Clock-Tick gesetzt, sobald die Daten/Instruktionsinformationen am Dateneingang an liegen; der Dout Pin wird durch das I/O-Gerät selbst gesetzt, sobald dieses seinerseits die Rückgabeinformation auf die Daten-pins anlegt. Das setzen des Dout-pins wird vom Computer als Interrupt(Unterabschnitt 2.4) bearbeitet. Der fall, dass gleichzeitig der Computer und das I/O-Gerät die Datenpins versuchen zu beschreiben wird dadurch gelöst, dass meistens die instruktionen definieren, wann das I/O-Gerät schreiben darf. Bei geräten, wie z.B. Peripherie-Geräten, die standartmäßig schreiben dürfen sollen ist von dem I/O-Gerät der Dout-pin schon eine Instruktion vor beginn des beschreibens des auf den Daten-pins zu setzen.

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

Tabelle 6: I/O-Instruktionen

ID	Gerätetyp	Name	Funktion	Verhalten
0x00	All	ls	gibt alle normalen Funktionen des Geräts zurück	Iin; n*Dout
0x01	All	ll	gibt ALLE (auch die, die man nicht verwenden sollte) Funktionen zurück	Iin(Instruktion); n*Dout(Antworten)
0x02	Drive	rsn (read singel normal)	Lesen eines einzelnen 16-Bit werts von einer 24-Bit Adresse	Iin(Instruktion); Din(Adresse); 1*Dout(Antwort)
0x03	Drive	rmn (read multiple normal)	Lesen einer reihe an 16-Bit werten von 24-Bit Adressen aus	Iin(Instruktion); Din(startAdresse); Din(Länge); n*Dout(Antworten)
0x04	Drive	wsn (write singel normal)	Schreiben eines einzelnen 16-Bit werts von einer 24-Bit Adresse	Iin(Instruktion); Din(Adresse); Din(Daten); Dout(bestätigung)
0x05	Drive	wmn (write multiple normal)	Schreiben einer reihe an 16-Bit werten an 24-Bit Adressen	Iin(Instruktion); Din(startAdresse); n*Din(Daten); Dout(bestätigung)
0x06	Drive	rsb (singel read big)	Lesen eines einzelnen 16-Bit werts von einer 40-Bit Adresse	Iin(Instruktion); Din(Adresse); Din(Adresse); 1*Dout(Antwort)
0x07	Drive	rmb (multiple write big)	Lesen einer reihe an 16-Bit werten von 40-Bit Adressen aus	Iin(Instruktion); Din(startAdresse); Din(startAdresse2); Din(Länge); n*Dout(Antworten)
0x08	Drive	wmb (singel write big)	Schreiben eines einzelnen 16-Bit werts von einer 40-Bit Adresse	Iin(Instruktion); Din(Adresse); Din(Adresse);Din(Daten); 1*Dout(bestätigung)
0x09	Drive	wmb (write multiple big)	Schreiben einer reihe an 16-Bit werten an 40-Bit Adressen	Iin(Instruktion); Din(startAdresse); n*Din(Daten); Dout(bestätigung)

Tabelle 6: I/O-Instruktionen

0x0a	Drive	rfn (read file normal)	Lesen einer gesamten Datei (24-bit Namen)	Iin(Instruction); Din(Name); n*Dout(Antwort)
0x0b	Drive	wfn (write file normal)	Schreiben einer neuen Datei (24-bit Namen)	Iin(instruction); Din(Name); n*Din(daten); Dout(bestätigung)
0x0c	Drive	rfpb (read file-part big)	Lesen eines teils einer Datei (24-bit Namen)	Iin(instruction); Din(Name); 2*Din(läge); n*Dout(Daten)
0x0d	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)
0x0e				
0x0f				

2.3.1 Berechtigungsliste

Berechtigungen: 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 (Unterabschnitt 2.6)

2.4 Interrupts

Interrupts sind Unterbrechungen im Programmablauf. Wenn ein Interrupt meldet wird. Wird dieser im nächsten Instruktions-Zyklus bearbeitet. Für diesen Zyklus, wird, je nach Interrupt-Quelle, nicht die Instruktion ander Stelle von IP sondern von IV (Interrupt Vektor) geladen.

Alle Interrupts werden blockiert wenn die OINT Flagge gesetzt ist. Blockiert heißt dabei das alle Interrupts nicht behandelt werden. Diese Flagge kann nur gesetzt werden wenn

sicher der Computer im Supervisormode befindet. siehe 2.5

Wir unterscheiden dabei zwischen drei verschiedenen Interrupt-Quelle, wobei jede Quelle einen eigenen IV hat, wie in Tabelle 7

Tabelle 7:

Interrupt-Quelle	IV Name	IV Standard Wert
IO-Interrupts	IV-IO	0x0000
INT-Instruktion	IV-INT	0x0000
Timed-Interrupts	IV-TI	0x0000

2.4.1 Timed-Interrupts

Der Timed-Intrrupt funktioniert wie ein Wecker. Nachdem Ausführen der STI (Set Timed Interrupt) Instruktion, wird nach angegebener Anzahl an ausgeführten Instruktion ein Intrrupt ausgelöst. So würde STI 0x00ff in 256 Instruktion einen Interrupt auslösen. Wenn ein Timed-Interrupt mit der Zeit 0x0000 gesetzt wird, so wird der Timed-Intrrupt deaktiviert.

2.4.2 IO-Interrupts

2.5 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 einschreckung, nutzen. Im Usermode hingegen ist der Computer in einigen Funktion eingeschnrenckt. Folgendes kann der Computer im Usermode *nicht* machen:

- Setzen der SUP Flagge
- Änderung der OINT Flagge
- Lesen und/oder schreiben in den Bereichen von 0x0000 - 0x3fff und 0xd000 - 0xffff
- Änderung der Banken

- Die IO-Berechtigungsliste zu modifizieren
- Zugriff auf IO-Geräte, welche nicht in der Berechtigungsliste gespeichert sind (siehe Unterunterabschnitt 2.3.1)

Der Modes 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 startet dabei mit gesetzter SUP Flagge.

Zudem ändert sich je nach Modus des Computer die Start position des Stacks wie in Unterunterabschnitt 2.2.2 erläutert.

Mehr zu der Motivation hinter diesem System in Unterabschnitt 3.4

2.6 Instruktions-Satz

Alle zuverfügung stehenden Instruktion sind in Tabelle 8. Die Ins-ID, steht für Instruktions-ID, welcher jeder Instruktion eine zugeordnet ist. op1, op2 und op3 sind dabei die möglich parameter der Instruktion. Sie können dabei Register (reg1, reg2 oder reg3) sein oder ein Konstante (c).

3 Kommentar zur Specification

3.1 16 bit Daten/Address Länge

Die Address- und Datenlänge besagt 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 Zahl $2^n - 1$ ist wenn n die Anzahl der Binary Ziffern (= Bits) ist und wenn man mit 0 anfängt zu zählen. Sie Tabelle 9

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.

Tabelle 8: Instruktions-Satz

Ins-ID	Name	op1	op2	op3	Beschreibung
0x00	NOP				Kein Effekt
0x01	MOV	reg1		reg3/c	$\text{reg1} = \text{reg3}/c$
0x02	ADD	reg1	reg2	reg3/c	$\text{reg1} = \text{reg2} + \text{reg3}/c$
0x03	SUB	reg1	reg2	reg3/c	$\text{reg1} = \text{reg2} - \text{reg3}/c$
0x04	MUL?	reg1	reg2	reg3/c	$\text{reg1} = \text{reg2} * \text{reg3}/c$
0x05	DIV?	reg1	reg2	reg3/c	$\text{reg1} = \text{reg2} / \text{reg3}/c$
0x06	XOR	reg1	reg2	reg3/c	$\text{reg1} = \text{reg2} \oplus \text{reg3}/c$
0x07	AND	reg1	reg2	reg3/c	$\text{reg1} = \text{reg2} \wedge \text{reg3}/c$
0x08	OR	reg1	reg2	reg3/c	$\text{reg1} = \text{reg2} \vee \text{reg3}/c$
0x09	NOT	reg1		reg3/c	$\text{reg1} = \neg \text{reg3}/c$
0x0a	STR	reg1	reg2	reg3/c	$\text{memory}[\text{reg2} + \text{reg3}/c] = \text{reg1}$
0x0b	LD	reg1	reg2	reg3/c	$\text{reg1} = \text{memory}[\text{reg2} + \text{reg3}/c]$
0x0c	BNK1			reg3/c	Setzt BID für Bank-Slot1
0x0d	BNK2			reg3/c	Setzt BID für Bank-Slot2
0x0e	PUSH			reg3/c	$\text{SP--}; \text{memory}[\text{SP}] = \text{reg1}$
0x0f	POP			reg3	$\text{reg3} = \text{memory}[\text{SP}]; \text{SP}++$
0x10	CALL			reg3/c	$\text{memory}[\text{SP}] = \text{IP}; \text{SP}++;$ $\text{IP} = \text{reg1}/c$
0x11	RET				$\text{IP} = \text{memory}[\text{SP}]; \text{SP}++$
0x12	TEST	reg1		reg3/c	Vergleicht reg1 und reg3/c
0x13	ME	reg1		reg3/c	$\text{reg1} = \text{reg3}/c$ if E
0x14	MOF	reg1		reg3/c	$\text{reg1} = \text{reg3}/c$ if O
0x15	MG	reg1		reg3/c	$\text{reg1} = \text{reg3}/c$ if G
0x16	ML	reg1		reg3/c	$\text{reg1} = \text{reg3}/c$ if L
0x17	MS	reg1		reg3/c	$\text{reg1} = \text{reg3}/c$ if SUP
0x18	MI	reg1		reg3/c	$\text{reg1} = \text{reg3}/c$ if OINT
0x19	IOUT	reg1		reg3/c	$\text{HIO}[\text{reg1}] = \text{reg3}/c$
0x1a	DOUT	reg1		reg3/c	$\text{DIO}[\text{reg1}] = \text{reg3}/c$
0x1b	DIN	reg1		reg3	$\text{reg3} = \text{DIO}[\text{reg1}]$
0x1c	AIO	reg1			Aktiviert IO[reg1]
0x1d	DIO	reg1			Deaktivieren IO[reg1]
0x1e	INT				Löst einen Interrupt aus
0x1f	STI			reg3/c	Setzt Timed Interrupt für reg1/c

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 Codea man laut Tabelle 9 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 eine sehr ausschlag gebende, daher haben wir diese zuerst festgelegt. Sie wird für viele weitere Entscheidung eine wichtige Rolle spielen. Wichtig ist nur das wir maximal 65536 Zustände oder Zahlen von 0 bis 65535 darstellen können mit unseren 16 bit

Tabelle 9: Maximalerepersäsentierbare 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 ausfolgenden Gründen entschieden: Zu erst die minimale Anzahl anregisteren die wirklich gebraucht würde, ist 5! Man braucht mindestens zwei all Zweck Register um binäre Rechenoperation durch zu führen, einen Stackpointer, einen Instruktion Point 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 ($2^3 = 8$) nehmen. Der Grund warum wir nicht mehr genommen haben hat mit der Encodierung von Instruktionen zutun siehe Unterabschnitt 3.3. Somit haben sich 8 Register als das Optimum ergeben

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

⁴Theorthisch ist es möglich das Flaggen Register wegzulassen, wir haben uns aber für eine solches Register entschieden siehe ??

3.3 Alternative Instruktion Encodierung

3.4 Gedanken Hinter dem Supervisormode

4 Quellenverzeichnis