Platinencomputer - Handbuch

Alexander Wersching und Simon Walter

2021

Inhaltsverzeichnis

1 Anforderungen und Methode

2 Spezifikation des Computers

Eine 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, Banking, Input/Output, Flaggen, Hardware Stack, Interrupts, Supervisor-Mode und das Instructuion Set.

Aber um zuerst ein paar Standarts festzulegen: der Computer hat eine Daten/Adressen Länge 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

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 eine anderen Ort) oder *geschrieben* werden (den alten Wert mit einem neuen Wert *überschreiben*)

Der Computers besitzt bei uns 8 verschiedene, jeweils 16 bit lange Register. Jedes Register hat dabei eine eigene Registernumber (= 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

Die Register A-D sind dabei die 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 evt. Programmabläufe stark stören könnte. Sie sind ausdrücklich nur für ihre zugewiesenen Aufgaben da.

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	"
5	C	"
6	D	"
7	FLAG	Enthält alle wichtigen Flaggen, siehe Unterunterabschnitt 2.1.1

2.1.1 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	Ungleicheit (>) zweier Werte	TEST arg[0] > arg[1]
3	L	Ungleicheit (<) zweier Werte	TEST arg[0] < arg[0]
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15	OINT	Overwride Interruption, siehe Unterabschnitt 2.4	
15	SUP	Supervisor-/Usermode	siehe Unterabschnitt 2.4

Alle Flaggen, die mit TEST als Schaltung gekennzeichnet werden, 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 macht SUP die nur manuel geschaltet werden kann wenn SUP gesetzt ist, siehe Unterabschnitt 2.5

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 Addresse, welche von 0 bis 65535 reicht, zugeordnet. Der Computer kann damit auf 65536words oder 131072bytes ($\approx$ 131.0kb) gleichzeitig 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 Register, während STR an der angebenen Addresse den Wert mit dem eines Register überschreibt.

Dabei ist die Memory Karte in Abbildung 1 zu stehen. Die zwei Hardware Stacks sind dort gekenzeichnet mit ↑, mehr dazu in Unterunterabschnitt 2.2.2

Die mit Supervisor gekennzeichneten Bereich können nur dann gelesen oder geschrieben werden, wenn die SUP Flagge gesetzt ist. Der Bereich, der mit ROM versehen wurde, ist dabei nicht beschreibbar, sondern kann nur gelesen werden. Eine STR Instruktion hat keine Wirkung.

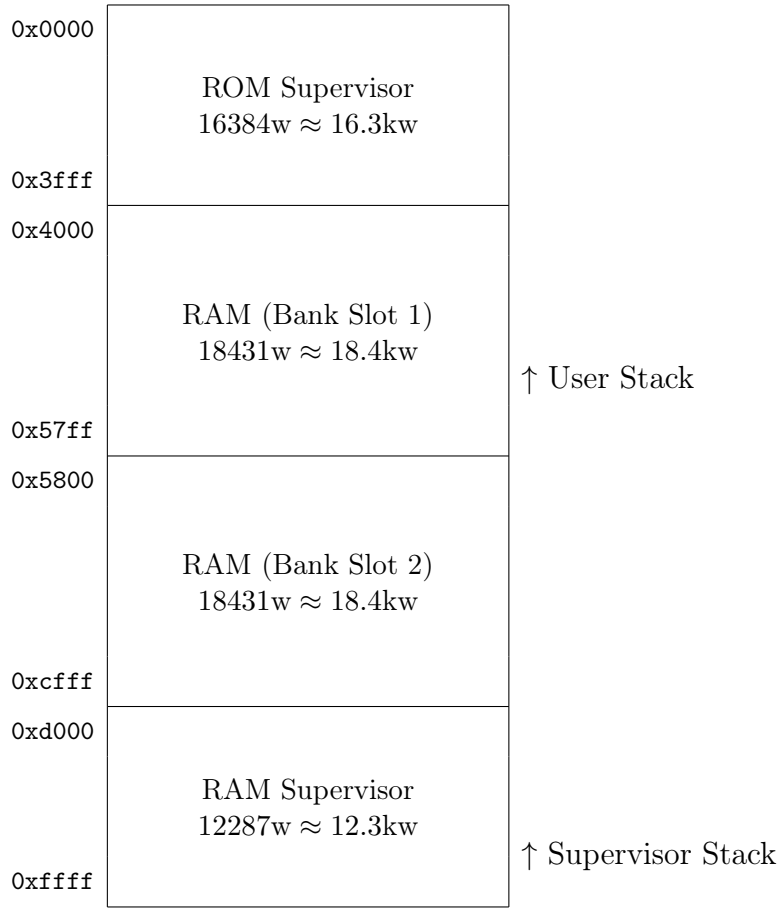
2.2.1 Banking

Banken sind externe Memory Module, die eingesteckt werden können. Alle Banken sind gleich groß (18.4kw = 36.8kb) 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 `0x4000 - 0x3fff` und Slot 2 `0x4000 - 0x57ff`). Er kann nun auf jenden dieser Slots eine der theoretisch 65536 Bank legen

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

Abbildung 1: Memory Karte



[2]. Das Banken legen lässt sich über die SBK1- (Select Bank 1) und SBK2- (Select Bank 2) Instruktion erreichen. Wenn man somit an diese entsprechende Memory Addressen schreibt, schreibt man in die Bank hinein.

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

Das Bankensystem dient zudem dazu Prozesse von einander zu isolieren, da bei nicht gesetzter SUP-Flagge Computer die Bank nicht änderen kann, siehe Unterabschnitt 2.5.

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

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 Münztappel. Es kann ein Wert auf den Stack gelegt werden (PUSH-Instruktion) oder ein Wert von oben abgehoben werden (POP-Instruktion).

Der Stack liegt dabei physisch in Memory vor. Jedes Element des Stacks okkupiert dabei eine Memoryzelle. Das Stack-Point Register-(SP) hält die Addresse, welche auf den oberste Wert des Stacks zeigt. Der Stack wächst dabei von hohen Addressen zu niedrigen Addressen. PUSH entspricht dabei der Dekrementierung des Stack Pointers um 1 und POP der Inkrementierung um 1.

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 und PUSH schalten automatisch zwischen Supervisor-Stack und User-Stack bei Änderung der SUP-Flagge.

2.3 Input/Output

Der Computer hat zudem die Möglichkeit mit verschieden IO-Geräte zu interagieren. Die Komunikation 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 standart 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		
Data$0-7$	InstructionID	Variable		Rückgabewert	Rückgabewert	
Data$8-15$	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 Interupt(Unterabschnitt 2.4) bearbeitet. Der fall, dass gleichzeitig der Computer und das I/O-Gerät die Datenpins versuchen zu beschreiben wird dadurch gelöst, dass meinstens 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.

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 Addressen sich mit dem wechseln zwischen den verschiedenen Banks ändern. gesetzt werden können sie nur durch den Supervisor, wobei auch dieser die RAM-Addressen nicht manuel ändern kann (Unterabschnitt 2.6)

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 Addresse	Iin(Instruktion); Din(Addresse); 1∗Dout(Antwort)
0x03	Drive	rmn (read multiple normal)	Lesen einer reihe an 16-Bit werten von 24-Bit Addressen aus	Iin(Instruktion); Din(startAddresse); Din(Länge); n∗Dout(Antworten)
0x04	Drive	wsn (write singel normal)	Schreiben eines einzelnen 16-Bit werts von einer 24-Bit Addresse	Iin(Instruktion); Din(Addresse); Din(Daten); Dout(bestätigung)
0x05	Drive	wmn (write multiple normal)	Schreiben einer reihe an 16-Bit werten an 24-Bit Addressen	Iin(Instruktion); Din(startAddresse); n∗Din(Daten); Dout(bestätigung)
0x06	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)
0x07	Drive	rmb (multiple write big)	Lesen einer reihe an 16-Bit werten von 40-Bit Addressen aus	Iin(Instruktion); Din(startAddresse); Din(startAddresse2); Din(Länge); n∗Dout(Antworten)

Tabelle 6: I/O-Instruktionen

0x08	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)
0x09	Drive	wmb (write multiple big)	Schreiben einer reihe an 16-Bit werten an 40-Bit Addressen	Iin(Instruktion); Din(startAddresse); n∗Din(Daten); Dout(bestätigung)
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				

2.4 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.

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 angebender 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

REWORK

2.6 Instruktions-Satz

Alle zuverfügenung stehenden Instruktion sind in Tabelle 8. Die Ins-ID, steht für Instruktions-ID, welecher 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).

Tabelle 8: Instruktions-Satz

Ins-ID	Name	op1	op2	op3	Beschreibung
0x00	NOP				Kein Effekt
0x01	MOV	reg1	reg2/c		reg1 = reg2/c
0x02	ADD	reg1	reg2	reg3/c	reg1 = reg2 + reg3/c
0x03	SUB	reg1	reg2	reg3/c	reg1 = reg2 − reg3/c
0x04	MUL?	reg1	reg2	reg3/c	reg1 = reg2 ∗ reg3/c
0x05	DIV?	reg1	reg2	reg3/c	reg1 = reg2 / reg3/c
0x06	XOR	reg1	reg2	reg3/c	reg1 = reg2 ⊕ reg3/c
0x07	AND	reg1	reg2	reg3/c	reg1 = reg2 ∧ reg3/c
0x08	OR	reg1	reg2	reg3/c	reg1 = reg2 ∨ reg3/c
0x09	NOT	reg1	reg2/c		reg1 = ¬reg2/c
0x0a	STR	reg1	reg2	reg3/c	memory[reg2 + reg3/c] = reg1
0x0b	LD	reg1	reg2	reg3/c	reg1 = memory[reg2 + reg3/c]
0x0c	BNK1	reg1/c			Setzt BID für Bank-Slot1
0x0d	BNK2	reg1/c			Setzt BID für Bank-Slot2
0x0e	PUSH	reg1/c			SP–; memory[SP] = reg1
0x0f	POP	reg1			reg1 = memory[SP]; SP++
0x10	PEEK	reg1			reg1 = memory[SP]
0x11	CALL	reg1/c			memory[SP] = IP; SP++;
					IP = reg1/c
0x1b	RET				IP = memory[SP]; SP++
0x12	TEST	reg1	reg2/c		Vergleicht reg1 und reg2/c
0x13	JE	reg1/c			IP = reg1/c if E
0x14	JO	reg1/c			IP = reg1/c if O
0x15	JG	reg1/c			IP = reg1/c if G
0x16	JL	reg1/c			IP = reg1/c if L
0x17	IN	reg1	reg2		reg1 = IO[reg2]
0x18	OUT	reg1/c	reg2		IO[reg2] = reg1/c
0x19	AIO	reg1			Aktiviert IO[reg1]
0x1a	DIO	reg1			Deaktivieren IO[reg1]
0x1b	TYPE	reg1	reg2		reg1 = Type von IO[reg2]
0x1c	INT				Löst einen Interrupt aus
0x1d	STI	reg1/c			Setzt Timed Interrupt für reg1/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 Zahle $2^n - 1$ 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.

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 trozdem ein limitiertes Buget haben, können wir auch nicht eine belibig 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: Maximalerepersäntierbare Zahlen (startend mit 0)

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

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

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 ($2^2 = 8$) nehmen. Der Grund warum wir nicht mehr genomen haben hat mit der Encodierung von Instruktionen zutun siehe Unterabschnitt 3.3. Somit haben sich 8 Register als das Optimum ergeben

3.3 Alternative Instruktion Encodierung

4

5 Quellenverzeichnis

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