UMach Spezifikation

25. April 2012

Inhaltsverzeichnis

1 Einführung

UMach ist eine einfache virtuelle Maschine (VM), die einen definierten Instruktionssatz und eine definierte Architektur hat. UMach orientiert sich dabei an Prinzipien von RISC Architekturen: feste Instruktionslänge, kleine Anzahl von einfachen Befehlen, Speicherzugriff durch Load- und Store-Befehlen, u.s.w. Die UMach Maschine ist Register-basiert. Der genaue Aufbau dieser Rechenmaschine ist im Abschnitt 2.1 ab der Seite 5 beschrieben.

Für den Anwender der virtuellen Maschine wird zuerst eine Assembler-Sprache zur Verfügung gestellt. In dieser Sprache werden Programme geschrieben die anschließend kompiliert werden. Die kompilierte Dateien (Maschinen-Code) wird von der virtuellen Maschine ausgeführt.

1.1 Anwendungsbeispiel

```
LOAD R1 90
LOAD R2 09
REV  R3 R1
```

2 Organisation der UMach VM

2.1 Aufbau

Dieser Abschnitt beschreibt den Aufbau der UMach virtuellen Maschine. Die virtuelle Maschine besteht aus internen und aus externen Komponenten. Dabei sind die externen Komponenten nicht wesentlich für die Funktionsfähigkeit der gesamten Maschine, d.h. die Maschine kann im Prinzip auch ohne die externen Komponenten funktionieren – in diesem Fall fehlt ihr eine Menge von Funktionen.

Interne Komponenten sind diejenigen Komponenten, die für die Funktionsfähigkeit der UMach Maschine wesentlich sind:

1. Recheneinheit
2. Logische Einheit
3. Register

Externe Komponenten

1. Anbindung an einem I/O-Port

2.1.1 Betriebsmodi

Ein Betriebsmodus bezieht sich auf die Art, wie die UMach VM läuft. Die UMach VM kann in einem der folgenden Betriebsmodi laufen:

1. Normalmodus
2. Einzelschrittmodus

Normalmodus Die virtuelle Maschine führt ohne Unterbrechung ein Programm aus. Nach der Ausführung befindet sich die Maschine in einem Wartezustand, falls sie nicht ausdrücklich ausgeschaltet wird.

Einzelschrittmodus Die virtuelle Maschine führt immer eine einzige Instruktion aus und nach der Ausführung wartet sie auf einen externen Signal um mit der nächsten Instruktion fortzufahren. Dieser Modus soll dem Entwickler erlauben, ein Programm schrittweise zu debuggen.

2.2 Register

Die Register sind die Speichereinheiten im Prozessor. Die meisten Anweisungen an die UMach Maschine operieren auf einer Art mit den Registern.

Für alle Register gilt:

1. Jedes Register ist ein Element aus der Menge $\mathcal{R}$, die alle Register beinhaltet. Die Notation $x \in \mathcal{R}$ bedeutet, dass x ein Register ist.

2. Die Speicherkapazität beträgt 32 Bit.

3. Es gibt eine eindeutige Maschinenzahl, die innerhalb der Maschine das Register identifiziert. Diese Zahl heißt Maschinenname und wird von einer Instruktion auf Maschinencode-Ebene verwendet, wenn sie das Register anspricht.

4. Die UMach Maschine erwartet die Angabe eines Registers als numerischer Wert (Maschinenname). Jedoch verwendet der Programmierer der Maschine auf Assembler Ebene einen eindeutigen Namen dieses Registers. Dieser Name heißt Assemblername.

Die UMach Maschine hat zwei Gruppen von Registern: die Allzweckregister und die Spezialregister.

2.2.1 Allzweckregister

Es gibt 32 Allzweckregister, die dem Programmierer für allgemeine Zwecke zur Verfügung stehen. Diese 32 Register werden beim Hochfahren der Maschine auf Null (`0x00`) gesetzt. Außer dieser Initialisierung, verändert die Maschine den Inhalt der Allzweckregister nur auf explizite Anfrage, bzw. infolge einer Instruktion.

Die 32 Register werden auf Maschinencode-Ebene von 1 bis 32 nummeriert (`0x01` bis einschliesslich `0x20` im Hexadezimalsystem). Diese Nummer ist der Maschinenname des Registers. Auf Assembler-Ebene, werden sie mit den Namen $R1, R2 \ldots$ bis $R32$ angesprochen (Assemblername). Die Zahl nach dem Buchstaben R ist im Dezimalsystem angegeben und ist fester Bestandteil des Registernamens.

Assemblername	`R1`	`R2`	`R3`	`...`	`R32`
Maschinenname	`0x01`	`0x01`	`0x03`	`...`	`0x20`

Register mit Nummer Null ($R0$) gibt es nicht und die Verwendung des Null-Registers wird von der Maschine als Fehler gemeldet.

Beispiel für die Verwendung von Registernamen:

Assembler Maschinencode	`ADD` `0x40`	`R1` `0x01`	`R2` `0x02`	`R3` `0x03`
Bytes	erstes Byte	zweites Byte	drittes Byte	viertes Byte
Algebraisch	$R_1 \leftarrow R_2 + R_3$			

2.2.2 Spezialregister

Die Spezialregister werden von der UMach Maschine für spezielle Zwecke verwendet, sind aber dem Programmiere sichtbar. Der Inhalt der Spezialregister kann von der Maschine während der Ausführung eines Programms ohne Einfluss seitens Programmierers verändert werden.

Nicht alle Spezialregister können durch Instruktionen überschrieben werden (sind schreibgeschützt).

Die Maschinennamen der Spezialregister setzen die Nummerierung der Allzweckregister zwar fort, die Assemblernamen aber nicht: es gibt kein Register $R33$. Die Tabelle 2.1 auf Seite 7 enthält die Liste aller Spezialregister. In der ersten Spalte steht der Assemblername, so wie er vom Programmierer verwendet wird. In der zweiten Spalte steht der Maschinenname im Hexadezimalsystem, so wie er im Maschinencode steht. Die dritte Spalte enthält eine kurze Beschreibung und Bemerkungen. Falls die Beschreibung nicht spezifiziert, dass ein Register schreibgeschützt ist, ist das Register nicht schreibgeschützt.

Tabelle 2.1: Liste der Spezialregister

IP	33	„Instruction Pointer“. Enthält zu jeder Zeit die Adresse der nächsten Instruktion. Wird auf Null gesetzt, wenn die Maschine hochfährt. Wird nach dem Abfangen einer Instruktion in das Register CIN automatisch inkrementiert. Schreibgeschützt.
CIN	??	„Current Instruction“. Enthält die nächste Instruktion. Vor der Ausführung einer Instruktion gilt, dass der Inhalt dieses Registers und der Speicher an der Adresse IP gleich sind. Schreibgeschützt.
ERR	??	Fehlerregister. Die einzelnen Bits geben Auskunft über Fehler, die mit der Ausführung des Programms verbunden sind. Schreibgeschützt.
RMOD	??	Betriebsmodus. („Run Mode“).
SMOD	??	Systemmodus. Kernel mode etc.
ZERO	??	Enthält die Zahl Null. Schreibgeschützt.
ARST	??	Arithmetischer Status. Division durch Null, Überlauf etc. Schreibgeschützt.
INT	??	„Interrupt Number“, wird für syscalls benutzt.
DEAD	??	Gesetzt, wenn der Zustand der Maschine keine weiteren Ausführungen mehr erlaubt. Die Maschine ist „tot“, wenn dieses Register einen Wert ungleich Null hat.

2.3 Speichermodell

2.3.1 Adressierungsarten

Als RISC-orientierte Maschine, greift die UMach lediglich in zwei Situationen auf den Speicher zu: zum Schreiben von Registerinhalten in den Speicher (Schreibzugriff) und zum Lesen von Speicherinhalten in einen Register (Lesezugriff). Die Adressierungsart beschreibt dabei, wie der Zugriff auf den Speicher erfolgen sollte, bzw. wie die angesprochene Speicheradresse angegeben wird. Anders ausgedrückt, beantwortet die Adressierungsart die Frage „wie kann eine Instruktion der Maschine eine Adresse angeben?“.

Die UMach Maschine kennt eine einzige Adressierungsart: die indirekte Adressierung, die unten beschrieben wird. Die direkte Adressierung, die aus einer direkten Angabe eines Speicheradresse besteht, wird von der indirekten Adressierung überdeckt.

Indirekte Adressierung

Die indirekte Adressierung verwendet zwei Register B und I, die von der Maschine verwendet werden, um die endgültige Adresse zu berechnen: Eine Instruktion, die diese Adressierung verwendet, hat also das Format RRR (siehe auch 3.1.5).

erstes Byte	zweites Byte	drittes Byte	viertes Byte	Algebraisch
Ladebefehl	R	B	I	$R \leftarrow mem(B + I)$
Speicherbefehl	R	B	I	$R \rightarrow mem(B + I)$

Die fünfte Spalte gibt jeweils den äquivalenten algebraischen Ausdruck wieder. $mem(x)$ steht dabei für den Inhalt der Adresse x.

Die zweite Zeile (Ladebefehl) bedeutet, dass die UMach Maschine die Inhalte der Register B und I aufaddieren soll, diese Summe als Adresse im Speicher zu verwenden und den Inhalt an dieser Adresse in den Register R zu kopieren.

Die dritte Zeile (Speicherbefehl) bedeutet: die Maschine soll den Inhalt des Registers R an die Adresse $B + I$ schreiben.

Üblicherweise enthält B eine Startadresse und I einen Versatz oder Index zur Adresse in B.

Vorteil der indirekten Adressierung ist, dass sie $2^{33} - 1$ mögliche Adressen ansprechen kann. Nachteil ist, dass zwei oder mehrere Instruktionen gebraucht werden, um diese Adressierung zu verwenden, denn die Register B und I erst entsprechend geladen werden müssen.

Die Register R, B und I stehen für beliebige Register.

2.4 Datentypen

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig ob ich schreibe: »Dies ist ein Blindtext« oder »Huardest gefburn«?. Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muß

keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie »Lorem ipsum« dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

3 Instruktionssatz

In diesem Kapitel werden alle Instruktionen der UMach VM vorgestellt.

3.1 Instruktionsformate

Eine Instruktion besteht aus einer Folge von 4 Bytes. Das Instruktionsformat beschreibt die Struktur einer Instruktion auf Byte-Ebene. Das Format gibt an, ob ein Byte als eine Registerangabe oder als reine numerische Angabe zu interpretieren ist.

Instruktionsbreite Jede UMach-Instruktion hat eine feste Bitlänge von 32 Bit (4 mal 8 Bit). Instruktionen, die für ihren Informationsgehalt weniger als 32 Bit brauchen, wie z.B. `NOP`, werden mit Nullbits gefüllt. Alle Daten und Informationen, die mit einer Instruktion übergeben werden, müssen in diesen 32 Bit untergebracht werden.

Byte Order Die Byte Order (Endianness) der gelesenen Bytes ist big-endian. Die zuerst gelesenen 8 Bits sind die 8 höchstwertigen (Wertigkeiten 2^{31} bis 2^{24}) und die zuletzt gelesenen Bits sind die niedrigstwertigen (Wertigkeiten 2^7 bis 2^0). Bits werden in Stücken von n Bits gelesen, wobei $n = k \cdot 8$ mit $k \in \{1, 4\}$ (byteweise oder wortweise).

Allgemeines Format Jede Instruktion besteht aus zwei Teilen: der erste Teil ist 8 Bit lang und entspricht dem tatsächlichen Befehl, bzw. der Operation, die von der UMach virtuellen Maschine ausgeführt werden soll. Dieser 8-Bit-Befehl belegt also die 8 höchstwertigen Bits einer 32-Bit-Instruktion. Die übrigen 24 Bits, wenn sie verwendet werden, werden für Operanden oder Daten benutzt. Beispiel einer Instruktionszerlegung:

Instruktion (32 Bit)	`00000001`	`00000010`	`00000011`	`00000100`
Hexa	`01`	`02`	`03`	`04`
Byte Order	erstes Byte	zweites Byte	drittes Byte	viertes Byte
Interpretation	Befehl (8 Bit)	Operanden, Daten oder Füllbits		

Die Instruktionsformate unterscheiden sich lediglich darin, wie sie die 24 Bits nach dem 8-Bit Befehl verwenden. Das wird auch in der 3-buchstabigen Benennung deren Formate wiedergeben.

In den folgenden Abschnitten werden die UMach-Instruktionsformate vorgestellt. Jede Angegebene Tabelle gibt in der ersten Zeile die Reihenfolge der Bytes an. Die nächste Zeile gibt die spezielle Belegung der einzelnen Bytes an.

3.1.1 000

erstes Byte	zweites Byte	drittes Byte	viertes Byte
Befehl	nicht verwendet		

Eine Instruktion, die das Format 000 hat, besteht lediglich aus einem Befehl ohne Argumenten. Die letzen drei Bytes werden von der Maschine nicht ausgewertet und sind somit Füllbytes. Es wird empfohlen, die letzten 3 Bytes mit Nullen zu füllen.

3.1.2 NNN

erstes Byte	zweites Byte	drittes Byte	viertes Byte
Befehl	numerische Angabe N		

Die Instruktion im Format NNN besteht aus einem Befehl im ersten Byte und aus einer numerischen Angabe N (einer Zahl), die die letzten 3 Bytes belegt. Die Interpretation der numerischen Angabe wird dem jeweiligen Befehl überlassen.

3.1.3 RNN

erstes Byte	zweites Byte	drittes Byte	viertes Byte
Befehl	R_1	numerische Angabe N	

Eine Instruktion im Format RNN besteht aus einem Befehl, gefolgt von einer Register Nummer R_1, gefolgt von einer festen Zahl N, die die letzten 2 Bytes der Instruktion belegt. Die genaue Interpretation der Zahl N wird dem jeweiligen Befehl überlassen. Zum Beispiel, die Instruktion

erstes Byte	zweites Byte	drittes Byte	viertes Byte
`0x20`	`0x01`	`0x02`	`0x03`

wird folgenderweise von der UMach Maschine interpretiert: die Operation mit Nummer `0x20` soll ausgeführt werden, wobei die Argumenten dieser Operation sind das Register mit Nummer `0x01` und die numerische Angabe `0x0203`.

3.1.4 RRN

erstes Byte	zweites Byte	drittes Byte	viertes Byte
Befehl	R_1	R_2	numerische Angabe N

Eine Instruktion im Format RRN besteht aus einem Befehl, gefolgt von der Angabe zweier Registers R_1 und R_2, jeweils in einem Byte, gefolgt von einer numerischen Angabe N (festen Zahl) im letzten Byte. Zum Beispiel, die Instruktion

erstes Byte	zweites Byte	drittes Byte	viertes Byte
`0x41`	`0x01`	`0x02`	`0x03`

soll wie folgt interpretiert werden: die Operation mit Nummer `0x41` soll ausgeführt werden, wobei die Argumenten dieser Operation sind Register mit Nummer `0x01`, Register mit Nummer `0x02` und die Zahl `0x03`.

3.1.5 RRR

erstes Byte	zweites Byte	drittes Byte	viertes Byte
Befehl	R_1	R_2	R_3

Eine Instruktion im Format RRR besteht aus der Angabe eines Befehls im ersten Byte, gefolgt von der Angabe dreier Register R_1, R_2 und R_3 in den jeweiligen folgenden drei Bytes. Die Register werden als Zahlen angegeben und deren Bedeutung hängt vom jeweiligen Befehl ab.

3.1.6 Zusammenfassung

Im folgenden werden die Instruktionsformate tabellarisch zusammengefasst.

Format	erstes Byte	zweites Byte	drittes Byte	viertes Byte
000	Befehl	nicht verwendet		
NNN	Befehl	numerische Angabe N		
RNN	Befehl	R_1	numerische Angabe N	
RRN	Befehl	R_1	R_2	numerische Angabe N
RRR	Befehl	R_1	R_2	R_3

Verteilung des Befehlsraums Zur besseren Übersicht der verschiedenen UMach-Instructionen, unterteilen wir den Instruktionssatz der UMach virtuellen Maschine in den folgenden Kategorien:

1. Kontrollinstruktionen, die die Maschine in ihrer gesamten Funktionalität betreffen, wie z.B. den Betriebsmodus umschalten.

2. Lade- und Speicherbefehle, die Register mit Werten aus dem Speicher, anderen Registern oder direkten numerischen Angaben laden und die Registerinhalte in den Speicher schreiben.

3. Arithmetische Instruktionen, die einfache arithmetische Operationen zwischen Registern veranlassen.

4. Logische Instruktionen, die logische Verknüpfungen zwischen Registerinhalten oder Operationen auf Bit-Ebene in Registern anweisen.

5. Vergleichsinstruktionen, die einen Vergleich zwischen Registerinhalten angeben.

6. Sprunginstruktionen, die bedingt oder unbedingt sein können. Sie weisen die UMach Maschine an, die Programmausführung an einer anderen Stelle fortzufahren.

7. Unterprogramm-Steuerung, bzw. Instruktionen, die die Ausführung von Unterprogrammen (Subroutinen) steuern.

8. Systeminstruktionen, die die Unterstützung eines Betriebssystem ermöglichen.

Die oben angegebenen Instruktionskategorien unterteilen den Befehlsraum in 8 Bereiche. Es gibt 256 mögliche Befehle, gemäß $2^8 = 256$. Die Verteilung der Kategorien auf die verschiedenen Maschinencode-Intervallen wird in der Tabelle 3.1 auf Seite 15 angegeben.

Maschinencodes	Kategorie
`00 - 0F`	Kontrollbefehle
`10 - 4F`	Lade-/Speicherbefehle
`50 - 8F`	Arithmetische Befehle
`90 - AF`	Logische Befehle
`B0 - BF`	Vergleichsbefehle
`C0 - DF`	Sprungbefehle
`E0 - EF`	Unterprogrambefehle
`F0 - FF`	Systembefehle

Tabelle 3.1: Verteilung des Befehlsraums nach Befehlskategorien. Die Zahlen sind im Hexadezimalsystem angegeben.

Die Tabelle 3.2 auf der Seite 16 enthält eine Übersicht aller Befehle und deren Maschinencodes. Diese Tabelle wird folgenderweise gelesen: in der am weitesten linken Spalte wird die erste hexadezimale Ziffer eines Befehls angegeben (ein Befehl ist zweistellig im Hexadezimalsystem). Jede solche Ziffer hat rechts Zwei Zeilen, die von links nach rechts gelesen werden: eine Zeile für die Ziffern von 0 bis 8, die anderen für die übrigen Ziffern 9 bis F (im Hexadezimalsystem). Die Assemblernamen (Mnemonics) der einzelnen Befehle sind an der entsprechenden Stelle angegeben.

Definitionsstruktur Im den folgenden Abschnitten werden die einzelnen Instruktionen beschrieben. Zu jeder Instruktion wird der Assemblername, die Parameter, der Maschinencode (Maschinenname) und das Instruktionsformat, das die Typen der Parameter definiert, formal angegeben. Zudem werden Anwendungsbeispiele angegeben. Die Instruktionsformate können im Abschnitt 3.1 ab der Seite 11 nachgeschlagen werden.

Zur Notation Mit $\mathcal{R}$ wird die Menge aller Register gekennzeichnet[1]. Die Notation $X \in \mathcal{R}$ bedeutet, dass X ein Element aus dieser Menge ist, mit anderen Worten, dass X ein Register ist. Analog bedeutet die Schreibweise $X, Y \in \mathcal{R}$, dass X und Y beide Register sind.

Gilt $X, Y \in \mathcal{R}$ und ist $\sim$ eine durch einen Befehl definierte Relation zwischen X und Y, so bezieht sich die Schreibweise $X \sim Y$ nicht auf die Maschinennamen von X und Y, sondern auf deren Inhalte. Zum Beispiel, haben die Register $R1$ und $R2$ die Maschinencodes `0x01` und `0x02` und sind sie mit den Werten 4 bzw. 5 belegt, so bedeutet $R1 + R2$ das gleiche wie $4 + 5 = 9$ und nicht `0x01 + 0x02 = 0x03`.

Andere verwendeten Schreibweisen:

[1] Nicht verwechseln mit den Symbolen $\mathbb{R}$ und $\mathbb{R}$, die die Menge aller reelen Zahlen bedeuten.

Tabelle 3.2: Befehlentabelle

	0	1	2	3	4	5	6	7
0	NOP	RST	CRM	CSM	DIE	RSR	AUTSM	SOCL
	HATE	TRST	ZMB	ALIV				
1	SET	SETU	COPY	MOVE				
2	LB	LBU	LH	LHU	LW	LWU		
3	SB	SBU	SH	SHU	SW	SWU		
4	PUSHB	PUSHH	PUSH					
	POPB	POPH	POP					
5	ADD	ADDU	ADDI	ADDIU				
	SUB	SUBU	SUBI	SUBIU				
6	MUL	MULU	MULI	MULIU				
	DIV	DIVU	DIVI	DIVIU				
7	MOD							
	NEG	ABS	INC	DEC				
8								
9	AND	ANDI	OR	ORI	XOR	XORI		
	NAND	NANDI	NOR	NORI				
A	SHL	SHLI	SHR	SHRI	SHRA	SHRAI	ROTL	ROTR
	NOT							
B	CMP	CMPU	CMPI	CMPIU				
C	BZ	BNZ	BLZ	BLEZ	BGZ	BGEZ		
	BEI	BNI	BLI	BLEI	BGI	BGEI		
D								
	JMP							
E	CALL	RET						
F	WAKE							
	KILL							
	8	9	A	B	C	D	E	F

$\mathbb{N}$	Menge aller natürlichen Zahlen: $0, 1, 2, \ldots$
$\mathbb{Z}$	Menge aller ganzen Zahlen: $\ldots, -2, -1, 0, 1, 2, \ldots$
$N \in \mathbb{N}$	N ist Element von $\mathbb{N}$, oder liegt im Bereich von $\mathbb{N}$
$X \leftarrow Y$	X wird auf Y gesetzt

3.2 Arithmetische Instruktionen

3.2.1 ADD

Assemblername	Parameter	Maschinencode	Format
ADD	$X, Y, Z \in \mathcal{R}$	0x50	RRR

Vorzeichen behaftete Addition der Registerinhalte Y und Z und speichern des Ergebnisses in das Register X. Entspricht dem algebraischen Ausdruck

$$X \leftarrow Y + Z$$

Beispiel:

```
SET   R1 1       # R1 ← 1
SET   R2 2       # R2 ← 2
ADD   R3 R1 R2   # R3 ← R1 + R2 = 1 + 2 = 3
#     X  Y  Z
SET   R2 -2      # R2 ← −2
ADD   R3 R3 R2   # R3 ← R3 + R2 = 3 + (−2) = 1
ADD   R3 R4  5   # Fehler! 5 kein Register
```

Vorzeichenlose Addition wird durch den Befehl ADDU ausgeführt.

3.2.2 ADDU

Assemblername	Parameter	Maschinencode	Format
ADDU	$X, Y, Z \in \mathcal{R}$	0x51	RRR

Vorzeichenlose Addition der Register Y und Z. Das Ergebnis wird in das Register X gespeichert. Enthält Y oder Z ein Vorzeichen (höchstwertiges Bit auf 1 gesetzt), so wird es nicht als solches interpretiert, sondern als Wertigkeit, die zum Betrag des Wertes hinzuaddiert wird ($+2^{31}$).

```
SET   R1 1      # R1 ← 1
SET   R2 -2     # R2 ← −2
ADDU  R3 R1 R2  # R3 ← (1 + 2 + 2^31) = 2147483651
```

3.2.3 ADDI

Assemblername	Parameter	Maschinencode	Format
ADDI	$X, Y \in \mathcal{R}$, $N \in \mathbb{Z}$	0x52	RRN

Hinzuaddieren eines festen vorzeichenbehafteten ganzzahligen Wert N zum Inhalt des Registers Y und speichern des Ergebnisses in das Register X. Entspricht dem algebraischen Ausdruck

$$X \leftarrow Y + N$$

N wird als vorzeichenbehaftete 8-Bit Zahl in Zweierkomplement-Darstellung interpretiert und kann entsprechend Werte von -128 bis 127 aufnehmen.

Beispiel:

```
SET   R1 1      # R1 ← 1
ADDI  R2 R1 2   # R2 ← R1 + 2 = 1 + 2 = 3
#     X  Y  N
ADDI  R2 R2 -3  # R2 ← R2 + (−3) = 3 + (−2) = 1
ADDI  R2 R3 R4  # Fehler! R4 kein n ∈ Z
```

3.2.4 ADDIU

Assemblername	Parameter	Maschinencode	Format
ADDIU	$X, Y \in \mathcal{R}$, $N \in \mathbb{N}$	0x53	RRN

Vorzeichenlose Addition des ganzzahligen Wertes N zum Inhalt des Registers Y und speichern des Ergebnisses in das Register X. Der Inhalt des Registers Y, die Zahl N und das Ergebnis $Y + N$ werden als vorzeichenlose Werte interpretiert.

3.3 Kontrollinstruktionen

3.3.1 `NOP`

Assemblername	Parameter	Maschinencode	Format
`NOP`	keine	`0x00`	000

Diese Instruktion („No Operation“) bewirkt nichts. Der Sinn dieser Instruktion ist, den Maschinencode mit Nullen füllen zu können, ohne dabei die gesamte Ausführung zu beeinflussen, außer, Zeitlupen zu schaffen.

3.3.2 `RST`

Assemblername	Parameter	Maschinencode	Format
`RST`	keine	`0x01`	000

Setzt alle Register außer dem Befehlszähler auf Null.

3.3.3 `CRM`

Assemblername	Parameter	Maschinencode	Format
`CRM`	$N \in \mathbb{N}$	`0x02`	NNN

„Change Run Mode“. Setzt das Betriebsmodus. Dabei ist das Parameter einer der folgenden konstanten Werten:

0	Normalmodus
1	Einzelschrittmodus

Siehe Abschnitt 2.1.1 auf Seite 5.

3.3.4 CSM

Assemblername	Parameter	Maschinencode	Format
CSM	$N \in \mathbb{N}$	0x03	NNN

Change System Mode.

3.3.5 DIE

Assemblername	Parameter	Maschinencode	Format
DIE	keine	0x04	000

Die Maschine ausschalten.

3.3.6 RSR

Assemblername	Parameter	Maschinencode	Format
RSR	keine	0x05	000

„Resurrect“. Die Maschine neustarten.

3.3.7 AUTSM

Assemblername	Parameter	Maschinencode	Format
AUTSM	keine	0x06	000

„Become autistic“. Bewirkt, dass alle Lese- und Schreibbefehle, die sich nicht ausschließlich auf Register beziehen, wirkungslos sind. Praktisch wird die Kommunikation mit dem Bussystem ausgeschaltet.

3.3.8 SOCL

Assemblername	Parameter	Maschinencode	Format
SOCL	keine	0x07	000

„Become social“. Schaltet die Kommunikation mit dem Bussystem ein.

3.3.9 HATE

Assemblername	Parameter	Maschinencode	Format
HATE	keine	0x08	000

„Hate“. Schaltet alle Schutzmechanismen ein.

3.3.10 TRST

Assemblername	Parameter	Maschinencode	Format
TRST	keine	0x09	000

„Trust“. Schaltet alle Schutzmechanismen aus.

3.3.11 ZMB

Assemblername	Parameter	Maschinencode	Format
ZMB	keine	0x0A	000

„Become a zombie“. Schaltet alle Registeränderungen aus. Nach der Ausführung dieses Befehls, alle Operationen, die einen Register modifizieren sollen, sind wirkungslos. Die Maschine ändert ihren Zustand nicht mehr, außer, dass sie weitere Befehle liest.

3.3.12 ALIV

Assemblername	Parameter	Maschinencode	Format
ALIV	keine	0x0B	000

„Become alive“. Nach der Ausführung dieses Befehls, alle Register können wie normal modifiziert werden.

3.4 Lade- und Speicherbefehle

3.4.1 SET

Assemblername	Parameter	Maschinencode	Format
SET	$X \in \mathcal{R}$, $N \in \mathbb{Z}$	0x10	RNN

Setzt den Inhalt des Registers X auf den ganzzahligen Wert N. Da N mit 16 Bit und im Zweierkomplement dargestellt wird, kann N Werte von -2^{15} bis $2^{15} - 1$ aufnehmen, bzw. von -32768 bis $+32767$. Werte außerhalb dieses Intervalls werden auf Assembler-Ebene entsprechend gekürzt (es wird modulo berechnet, bzw. nur die ersten 16 Bits aufgenommen).

Beispiele:

```
SET R1  8     # R1 ← 8
SET R2 -3     # R2 ← −3
SET R3 65536  # R3 ← 0, da 65536 = 2^16 ≡ 0 mod 2^16
SET R4 70000  # R3 ← 4464 = 70000 mod 2^16
```

3.4.2 SETU

Assemblername	Parameter	Maschinencode	Format
SETU	$X \in \mathcal{R}$, $N \in \mathbb{N}$	0x11	RNN

Setzt den Inhalt des Registers X auf den positiven natürlichen Wert N. N wird vorzeichenlos interpretiert. Entsprechend kann N Werte von 0 bis +65535 aufnehmen. Wird

dem Assembler einen Wert außerhalb dieses Bereichs gegeben, so schneidet der Assembler alle Bits außer den ersten 16 weg und betrachtet das Ergebnis als vorzeichenlose Zahl.

Beispiele:

```
SETU R1    8      # R1 ← 8
SETU R2    70000  # R2 ← 4464
SETU R2   -70000  # R2 ← 61072
```

Tabellenverzeichnis

Glossar

A | **B** | **I** | **M** | **R**

A

Adressierungsart
: Die Art, wie eine Instruktion die Umach Maschine dazu veranlasst, einen Speicherbereich zu adressieren. Siehe auch Abschnitt 2.3.1.

Assemblername
: Der Name eines Registers oder eines Befehls, so wie er in einem textuellen Programm (ASCII) verwendet wird. $R1$, $R2$, ADD sind Assemblernamen von Registern und Befehlen.

B

Befehl
: Die ersten 8 Bits in einer Instruktion. Operation code.

Befehlsraum
: Die Anzahl der möglichen Befehle, abhängig von der Befehlsbreite. Beträgt die Befehlsbreite 8 Bit, so ist der Befehlsraum $2^8 = 256$.

Betriebsmodus
: Die Art, wie die UMach Maschine die einzelnen Instruktionen abarbeitet. Siehe auch 2.1.1.

Byte
: Eine Reihe oder Gruppe von 8 Bit.

I

Instruktion
: Eine Anweisung an die UMach VM etwas zu tun. Eine Instruktion besteht aus einem Befehl (Operation Code) und eventuellen Argumenten.

Instruktionsformat
: Beschreibt die Struktur einer Instruktion auf Byte-Ebene und zwar es gibt an, ob ein Byte als eine Registerangabe oder als reine numerische Angabe zu interpretieren ist. Siehe 3.1.

Instruktionssatz
: Die Menge aller Instruktionen, die von der UMach Maschine ausgeführt werden können.

M

Maschinenname
: Der Name eines Registers oder eines Befehls, so wie er im Maschinencode erscheint. `0x01`, `0x02`, `0x40` sind Maschinennamen von Registern und Befehlen.

R

Register
: Eine sich im Prozessor befindende Speichereinheit. Das Register ist dem Programmierer sichbar und kann mit Werten geladen werden. Siehe Abschnitt 2.2, Seite 6.

Index