UMach Spezifikation

21. 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 4 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 Betriebmodus 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. Die Speicherkapazität beträgt 32 Bit.

2. 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.

3. 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	ADD	R1	R2	R3
Maschinencode	0x40	0x01	0x02	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.

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 Abschnitt 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		

Die Instruktion im Format NNN besteht aus einem Befehl im ersten Byte und aus einer numerischen Angabe (einer Zahl), die die letzten 3 Bytes belegt. Die Interpretation der numerischen Angabe wird dem jeweiligen Befehl überlassen, jedoch wird diese meistens eine Adresse oder ein Versatz bedeuten.

3.1.3 RNN

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

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

erstes Byte	zweites Byte	drittes Byte	viertes Byte
`0x12`	`0x01`	`0x02`	`0x03`

wird folgenderweise von der UMach Maschine interpretiert: die Operation mit Nummer `0x12` 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

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

erstes Byte	zweites Byte	drittes Byte	viertes Byte
`0x12`	`0x01`	`0x02`	`0x03`

soll wie folgt interpretiert werden: die Operation mit Nummer `0x12` 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 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		
RNN	Befehl	R_1	numerische Angabe	
RRN	Befehl	R_1	R_2	numerische Angabe
RRR	Befehl	R_1	R_2	R_3

3.2 Instruktionen

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 13 angegeben.

Maschinencodes	Kategorie
00 - 1F	Kontrollbefehle
20 - 3F	Lade-/Speicherbefehle
40 - 5F	Arithmetische Befehle
60 - 7F	Logische Befehle
80 - 9F	Vergleichsbefehle
A0 - BF	Sprungbefehle
C0 - DF	Unterprogrambefehle
E0 - FF	Systembefehle

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

Die Tabelle 3.2 auf der Seite 14 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 Mnemonics der einzelnen Befehle sind an der entsprechenden Stelle angegeben.

Im den folgenden Abschnitten werden die einzelnen Instruktionen beschrieben. Zu jeder Instruktion wird der „Mnemonic Code“, der Maschinen Code, das Instruktionsformat und Anwendungsbeispiele angegeben.

	0	1	2	3	4	5	6	7
0	NOP							
1								
2								
3								
4	ADD	ADDI						
5								
6								
7								
8								
9								
A								
B								
C								
D								
E								
F								
	8	9	A	B	C	D	E	F

Tabelle 3.2: Instruktionentabelle

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, so wie er in einem textuellen Programm erscheint. $R1$, $R2$ sind Assemblernamen von Registern.

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 $2^8 = 256$.

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, so wie er im Maschinencode erscheint. `0x01`, `0x02` sind Maschinennamen von Registern.

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 5.

Index