UMach Spezifikation

19. 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 2.2 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 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 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

2.2.1 Register

Der Register sind die Speichereinheiten im Prozessor, die dem Programmierer sichtbar sind. Die meisten Anweisungen an die UMach Maschine bearbeiten auf einer Art diese Register.

2.2.2 Rechen- und logische Einheit

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.

2.2.3 I/O-Einheit

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.

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?“.

Eine Instruktion kann die UMach Maschine auf zwei Arten dazu veranlassen, den Speicher zu adressieren:

1. Direkte Adressierung

2. Indirekte Adressierung

Direkte Adressierung

Die direkte Adressierung wird durch Angabe einer Speicheradresse spezifiziert. Die Adresse wird in den letzten 2 Bytes einer Instruktion angeben. Diese Instruktion hat das Format RNN (siehe auch 3.1.3). Eine Instruktion, die die direkte Adressierung verwendet, hat also das folgende Format:

erstes Byte	zweites Byte	drittes Byte	viertes Byte	Algebraisch
Ladebefehl	R	Adresse A		$R \leftarrow mem(A)$
Speicherbefehl	R	Adresse A		$R \rightarrow mem(A)$

Die zweite Zeile weist die UMach Maschine an, dass sie den Inhalt der angegebenen Speicheradresse A (die letzten 2 Bytes) in das Register R laden soll. Die dritte Zeile gibt

an, dass die Maschine den Inhalt des Registers R an die angegebene Adresse A schreiben (speichern) soll ($mem(x)$ steht für Inhalt der Speicheradresse x).

Vorteil dieser Adressierungsart ist, dass sie keine zusätzliche Befehle zum Lesen oder Schreiben in den Speicher benötigt. Nachteil ist, dass sie nur Adressen angeben kann, die sich mit 16 Bit darstellen lassen. Adressen, die größer als $2^{16} - 1$ sind können durch die zweite Adressierungsart angegeben werden.

Indirekte Adressierung

Die indirekte Adressierung verwendet nicht, wie die direkte Adressierung, ein Register und eine Speicheradresse, sondern 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 absolute 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 $2^{32} \cdot 2^{32} = 2^{64}$ mögliche Adressen ansprechen kann (2^{32} wegen der 32 Bit Register). 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

Instruktionsbreite Jede UMach-Instruktion hat eine feste Bitlänge von 32 Bit (4 mal 8 Bit). Das gilt unabhängig davon, was die Instruktion tut. 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 Byte-Reihenfolge, die für den Mensch selbstverständlich wäre (von links nach rechts lesen): 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 $n = k \cdot 8$ und $k \in \mathbb{N} \setminus \{0\}$ (Byte für Byte).

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 Instruktion. Die übrigen 24 Bit 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	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	numerische Angabe	
RRN	Befehl	R_1	R_2	numerische Angabe
RRR	Befehl	R_1	R_2	R_3

3.2 Instruktionen

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 Gesamtheit steuern, wie z.B. den Betriebsmodus umschalten oder Ausschalten.

2. Arithmetische Instruktionen, wie z.B. Addieren, Subtrahieren etc. Alle arithmetische Instruktionen operieren auf Register Inhalte.

3. Logische Instruktionen, die eine logische Verknüpfung zwischen den Inhalten von Registern veranlassen.

4. Vergleichsinstruktionen: Vergleichen von Registerinhalten.

5. Sprünge im Programmcode.

6. Load- und Store-Instruktionen die einzigen, die einen Zugriff auf den Speicher ausführen.

7. Input-Output-Instruktionen operieren auf die I/O-Einheit der UMach VM.

8. Andere Befehle: diese Kategorie enthält Instruktionen, die in keiner anderen Kategorie passen und zukünftige Erweiterungen.

Verteilung des Befehlsraums Die oben angegebenen Instruktionskategorien unterteilen den Befehlsraum in 8 Bereiche. Es gibt 256 Befehle, gemäß $2^8 = 256$.

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

Glossar

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

A

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

B

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

Befehlsraum
: Die Anzahl der möglichen Befehlen, 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.

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

R

Register
: Eine sich im Prozessor befindende Speichereinheit. Der Register ist dem Programmierer sichbar und kann mit Werten geladen werden. Siehe Abschnitt 2.2.1, Seite 5.

Index