# UMach Spezifikation

18. 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

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.

## 2.4 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, dass sie die 24 Bits nach dem 8-Bitigen Befehl unterschiedlich verwenden.

In den folgenden Abschnitten werden die UMach-Instruktionsformate vorgestellt. Jede Angegebene Tabelle gibt in der ersten Zeile die Reihenfolge der Bytes an. Alle folgenden Zeilen geben die spezielle Belegung der einzelnen Bytes an. Falls es mehrere Alternativen zu einer Belegung möglich ist, werden die Alternativen untereinander aufgelistet (jede in eigener Zeile).

### 3.1.1 Op0

| erstes Byte | zweites Byte | drittes Byte | viertes Byte |
|---|---|---|---|
| Befehl | nicht verwendet | | |

Eine Instruktion, die das Format Op0 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. Der Name kommt von „Operation Zero“, ein Befehl gefolgt von Nullen.

### 3.1.2 OpNo

| erstes Byte | zweites Byte | drittes Byte | viertes Byte |
|---|---|---|---|
| Befehl | feste Zahl | | |

Die Instruktion im Format OpNo besteht aus einem Befehl im ersten Byte und aus einer Zahl, die die letzten 3 Bytes belegt. Die Interpretation dieser Zahl wird dem jeweiligen Befehl überlassen, wird aber meistens eine Adresse oder ein Versatz bedeuten. Der Name kommt von „Operation Number“, ein Befehl gefolgt von einer Zahl.

### 3.1.3 OpRNo

| erstes Byte | zweites Byte | drittes Byte | viertes Byte |
|---|---|---|---|
| Befehl | $R$ | feste Zahl | |

Eine Instruktion im Format OpRNo besteht aus einem Befehl, gefolgt von einem Register, gefolgt von einer festen Zahl, die die letzten 2 Bytes der Instruktion belegt. Die genaue Interpretation der Zahl wird dem jeweiligen Befehl überlassen. Der Name kommt von „Operation Register Number“, ein Befehl, gefolgt von einem Register, gefolgt von einer Zahl.

### 3.1.4 OpRRNo

| erstes Byte | zweites Byte | drittes Byte | viertes Byte |
|---|---|---|---|
| Befehl | $R_1$ | $R_2$ | feste Zahl |

Eine Instruktion im Format OpRR besteht aus einem Befehl, gefolgt von der Angabe zweier Registers, jeweils in einem Byte, gefolgt von einer festen Zahl im letzten Byte. Das erste Byte wird nicht verwendet. Der Name kommt von „Operarion Register Register Number“.

### 3.1.5 Op3R

| erstes Byte | zweites Byte | drittes Byte | viertes Byte |
|---|---|---|---|
| Befehl | $Z$ | $R_1$ | $R_2$ |

Eine Instruktion im Format Op3R 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 haben die folgenden Bedeutungen:

- $Z$ ist das Zielregister, wo das Ergebnis der Operation gespeichert werden sollte. Die Operation wird vom Befehl angegeben.
- $R_1$ ist der erste Registeroperand.
- $R_2$ ist der zweite Registeroperand.

Dieses Format entspricht dem folgenden algebraischen Ausdruck:

$$Z \leftarrow R_1 \text{ (Befehl) } R_2$$

Der Name kommt von „Operation 3 Registers“, ein Befehl gefolgt von 3 Registern.

### 3.1.6 Zusammenfassung

Im folgenden werden die Instruktionsformate tabellarisch zusammengefasst.

| Format | erstes Byte | zweites Byte | drittes Byte | viertes Byte |
|---|---|---|---|---|
| Op0 | Befehl | nicht verwendet | | |
| OpNo | Befehl | feste Zahl | | |
| OpRNo | Befehl | $R$ | feste Zahl | |
| OpRR | Befehl | $R_1$ | $R_2$ | feste Zahl |
| Op3R | Befehl | $Z$ | $R_1$ | $R_2$ |

## 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

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

## 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