Verwendung der virtuellen UMach Maschine

28. September 2012

Inhaltsverzeichnis

1 Einführung

1.1 Ein Beispiel

Gleich am Anfang soll ein Beispiel für die Verwendung der UMach virtuellen Maschine und des Assemblers gegeben werden.

Wir haben ein UMach-Programm in eine normale Textdatei geschrieben. Das Programm kann sich über mehrere Dateien erstrecken, aber hier verwenden wir nur eine Datei. Das Programm sieht wie folgt aus:

Listing 1: Ein einfaches Beispiel

```
    set r1 3
loop:
    dec r1
    cmp r1 zero
    be  finish
    jmp loop
finish:
    EOP
```

Dieses Programm wurde in der Datei `prog1.uasm` gespeichert (die Endung der Datei ist eigentlich egal). Wir gehen davon aus, dass der Assembler `uasm`, die Programmdatei `prog1.uasm` und die virtuelle Maschine `umach` sich in dem selben Verzeichniss befinden. Sonst muss man die Befehle entsprechend anpassen.

Das Programm kann wie folgt assembliert werden:

```
./uasm -o prog.ux prog1.uasm
```

Die Option `-o` gibt die Ausgabedatei an. Wird diese Option nicht angegeben, so wird das assemblierte Programm in die Datei `u.out` geschrieben. Ergebnis des Assemblers is also die Datei `prog.ux`. Jetzt kann man diese Datei „ausführen“:

```
./umach prog.ux
```

Das Programm beendet sich ohne Ausgabe. Starten wird also das Programm im Debug-Modus (Option `-d`):

```
./umach -d prog.ux
```

Es wird die erste Anweisung angezeigt, der aktuelle Programm-Counter (Inhalt des Registers `PC`) und ein Prompt, der auf eine Eingabe von uns wartet. So könntes es weiter gehen:

```
[256]    SET    R1     3
umdb > show reg r1
R1 = 0x00000000 = 0
umdb > s
[260]    DEC    R1
umdb > show reg r1
R1 = 0x00000003 = 3
umdb > s
[264]    CMP    R1     ZERO
umdb > s
[268]    BE     2
umdb > s
[272]    JMP    -3
umdb > s
[260]    DEC    R1
umdb > s
[264]    CMP    R1     ZERO
umdb > show reg r1
R1 = 0x00000001 = 1
umdb > s
[268]    BE     2
umdb > s
[272]    JMP    -3
umdb > s
[260]    DEC    R1
umdb > s
[264]    CMP    R1     ZERO
umdb > s
[268]    BE     2
umdb > s
[276]    EOP
umdb > s
umdb > s
The maschine is not running.
umdb > show reg r1 cmpr
R1 = 0x00000000 = 0
CMPR = 0x00000000 = 0
umdb > q
```

2 Ausführen

2.1 Optionen

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.

Das hier ist der zweite Absatz. 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.

Und nun folgt – ob man es glaubt oder nicht – der dritte Absatz. 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.

Nach diesem vierten Absatz beginnen wir eine neue Zählung. 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.

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 Assembler

3.1 Eingabe Dateien

Es können beliebig viele Programmdateien angegeben werden. Sie werden der Reiche nach abgearbeitet. Man beachte, dass die Instruktion `EOP` das Ende des Programms signalisiert. Falls nachher noch weitere Befehle, eventuell in anderen Datein, angegeben werden, werden diese zwar assembliert, aber nicht ausgeführt.

Alles außer Labels is case insensitive. Man kann beliebig Leerzeichen (whitespace) verwenden.

3.2 Labels

Der UMach-Assembler unterstüzt die Verwendung von sogenannten „Labels“, oder Sprungmarken. Ein Label markiert das Sprungziel für die verschiedenen Sprungbefehle.

Um ein Label im Programmcode zu definieren, schreibt man den Labelnamen, gefolgt von einem Doppelpunkt. Zwischen dem Labelnamen und dem Doppelpunkt können, außer das „newline“-Zeichen, beliebig viele Leerzeichen eingefügt werden.

Labelname ist höchstens 128 Buchstaben lang.

Labelnamen müssen selbst keine Leerzeichen (whitespace) beinhalten.

Mehrfache Labels, die auf den selben Befehl zeigen, werden unterstüzt. Diese können beliebing im Code geschrieben werden, d.h. sie können auf verschiedenen Zeilen oder auf einer einzigen Zeile geschrieben werden. Mehrfache Labels zeigen jeweils auf den nächsten Befehl. Das Programm 2 zeigt ein Beispiel für die Verwendung der (mehrfachen) Labels.

Listing 2: Beispiel für Labels

```
set r1 5

loop  :
  do  :

 it:now:cmp r1 zero
    be  finish
    dec r1
```

```
    jmp loop
    jmp do

finish: end: EOP
```

In diesem Beispiel zeigen die Labels `loop`, `do`, `it` und `now` alle auf den selben Befehl: `cmp r1 zero`. Zu diesem Befehl wird gesprungen, indem man einen von diesen Labels verwendet. Hier bewirken beide Sprungbefehle `jmp loop` und `jmp do` einen Sprung zum selben `cmp`-Befehl. Die Labels `finish` und `end` zeigen auf den Befehl `EOP`. Man bemerkt auch die freie Verwendung der Leerzeichen (über die Lesbarkeit dieses Beispiels lässt sich diskutieren).

3.3 Programmdaten

Daten werden nach der Anweisung `.data` angegeben. Diese Anweisung muss alleine auf einer eigenen Zeile stehen.

Die Programmdaten können auch auf verschiedenen Programmdateien verteilt werden, sie werden vom Assembler zusammengefügt und ans Ende der assemblierten Datei eingefügt.

Alle Daten haben jeweils eine Länge, die ein Vielfaches von 4 Byte darstellt. Bedarft ein Datenelement weniger als $4k$ Bytes, so wird es trotzdem auf eine Länge von $4k$ mit Nullbytes gefüllt.

3.3.1 Strings

String Daten werden mit der Anweisung `.string` angegeben. Nach `.string` kommt der Name des Strings und dann der eigentlich String zwischen Anführungszeichen. Siehe das Programm 3 für ein Beispiel.

Strings werden so assembliert, dass sie immer ein Vielfaches von 4 Bytes belegen. Braucht der textuelle Inhalt des Strings weniger als 4 Byte, so wird der String mit Nullbytes gefüllt.

3.3.2 Ganze Zahlen

Ganze Zahlen werden mit der Anweisung `.int` angegeben. Nach `.int` folgt der Name (Label) der Zahl, dann die eigentliche Zahl. Diese kann in Hexa-, Oktal- oder Dezimalsystem angegeben werden, analog wie in der C/Java Sprache.

Listing 3: Verwendung der Daten

```
set  r1 dezimal # r1 = Adresse von 'dezimal'
lw   r1 r1      # r1 = Wert an der Adresse 'dezimal'

set  r2 hexa    # r2 = Adresse von 'hexa'
lw   r2 r2      # r2 = Wert an der Adresse 'hexa'

set  r3 oktal   # r3 = Adresse von 'oktal'
lw   r3 r3      # r3 = Wert an der Adresse 'oktal'

# hier haben r1, r2 und r3 den selben Wert

set  r1 str     # r1 = Adresse der Daten 'str'
set  r2 strsize # r2 = Adresse der Daten 'strsize'
sub  r2 r2 r1   # r2 = laenge der Daten 'str'

out  r1 r2 zero # Ausgabe "Hallo"

set  r1 nl      # r1 = Adresse von nl
addi r1 r1 3    # r1 zeigt auf das 4. Byte von nl
set  r2 1
out  r1 r2 zero


.data
### Datenbereich ###

.int dezimal 171   # dezimalsystem
.int hexa    0xAB  # hexadezimalsystem
.int oktal   0253  # oktalsystem

# String daten

.string str "Hallo Welt"
.int    strsize 0   # dummy Wert
.int    nl  0x0A    # new line
```

Angenommen, der Assembler `uasm`, die virtuelle Maschine `umach` und das Programm `example_data.uasm` befinden sich im aktuellen Verzeichniss, kann das Programm 3 wie folgt assembliert und ausgeführt werden:

```
./uasm example_data.uasm
./umach u.out
```

Es wird lediglig „Hallo Welt“ ausgegeben.

4 Debuggen

4.1 Debug-Befehle

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.

Listings

Index