Das zugrundeliegende Programm:

Demonstration der arithmetischen Befehle mul und div

Wie im HowTo beschrieben, wird *Simulation02.asm* im MARS geöffnet.

Der Code zeigt die Verwendung der arithmetischen Befehle ***mul*** und ***div*** und wie dabei die Register ***HI*** und ***LO*** benötigt werden. Im Folgenden werden einzelne Codeabschnitte näher erläutert.

```
.data
        number1:        .word 0x0fffffff
        number2:        .word 0x12345678
        number3:        .word 0x01234567
```

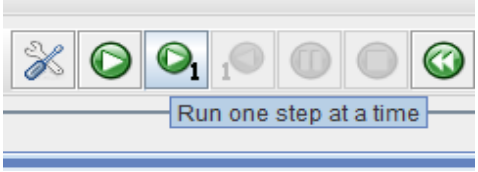
Die Konstanten werden hinterlegt, die im Programmablauf benötigt werden, als Beispielzahlen wurden 0x0fffffff, 0x12345678 und 0x01234567 willkürlich ausgewählt.

```
.text
        # Laden der Beispielzahlen in t0, t1, t2
        lw $t0, number1
        lw $t1, number2
        lw $t2, number3
```

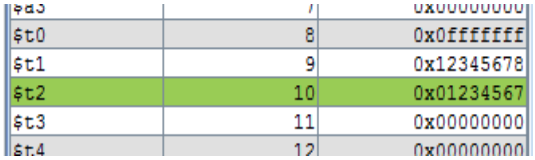
Die hinterlegten Zahlen number1, number2 und number3 werden in die temporären Register t0, t1 und t2 geladen, sehr schön kann man das im „*Execute*“ Fenster im *Text Segment* verfolgen:

	Basic	Source
.001	lui $1,0x00001001	13: lw $t0, number1
0000	lw $8,0x00000000($1)	
.001	lui $1,0x00001001	14: lw $t1, number2
0004	lw $9,0x00000004($1)	
.001	lui $1,0x00001001	15: lw $t2, number3
0008	lw $10,0x00000008($1)	

Empfehlenswert ist es, Schritt für Schritt durch das Programm zu gehen, das geschieht durch Klicks auf den „*Run one step at a time*“ Button:



Im rechten Bereich unter „Register“ findet man nun nach Ausführung der drei Codezeilen die mit den Beispielzahlen gefüllten Register t0-t2,



$t0	8	0x0fffffff
$t1	9	0x12345678
$t2	10	0x01234567
$t3	11	0x00000000
$t4	12	0x00000000

sodass nun die eigentlichen Operationen beginnen können. Zuerst die Multiplikation,

```
# Multiplikation
        mul $t3, $t0, $t1    # Die unteren 32 Bits des Produkts werden in $t3 abgelegt
        mfhi $s0             # Um auf die oberen 32 Bits zugreifen zu können, wird mfhi benötigt
        mflo $s1             # Für den Zugriff auf die unteren 32 Bits des Produkts wird mflo verwendet
```

wobei wichtig ist, dass das Produkt zweier 32-Bit-Zahlen bis zu 64 Bit groß wird. Es werden also 2 Register benötigt, um es berechnen zu können. Die oberen 32 Bit werden automatisch im Register *hi* abgelegt, darauf zugreifen kann man dann mit dem Befehl *mfhi,* die unteren 32 Bit werden hier in Register t3 geschrieben, weil es das *destination register* im Befehl ist, zusätzlich liegen sie im Register *lo,* wo mit dem Befehl *mflo* auf sie zugegriffen werden kann.

Dann folgt die Division,

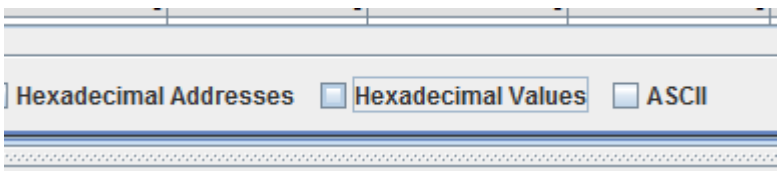
```
# Division $t0 durch $t1

        div $t0, $t2         # speichert den Quotienten in lo und den rest in hi
        mfhi $s2             # holt den Rest der Division aus hi und legt ihn in $s2 ab
        mflo $s3             # holt den Quotienten aus lo und legt ihn in $s3 ab
```

wobei prinzipiell der Quotient in *lo* und der Rest in *hi* abgelegt wird.

```
# exit
        li $v0, 10
        syscall
```

Der Wert 10 für den syscall bedeutet: „*exit (terminate execution)*“ und der syscall mit diesem Wert beendet die Ausführung des Programms.

Das Programm hat keine Ausgabe, im rechten Bereich lassen sich schön die Werte der Register nach Ausführung ablesen, wobei man wahlweise die Darstellung als Hexadezimal-Zahl oder als Dezimalzahl wählen kann, umgeschaltet wird das durch Setzen des Häkchens bei „*Hexadecimal Values*“ am unteren Rand des *Data Segment*s:



Abschließende Registerbelegung:

$t0	8	0x0fffffff
$t1	9	0x12345678
$t2	10	0x01234567
$t3	11	0x6dcba988
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x01234567
$s1	17	0x6dcba988
$s2	18	0x0012345d
$s3	19	0x0000000e
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400038
hi		0x0012345d
lo		0x0000000e