Das zugrundeliegende Programm:

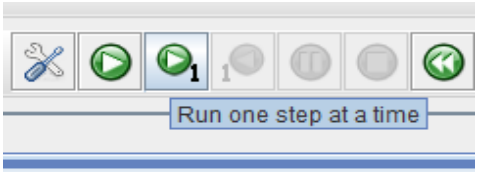
Division zweier Integer mit Eingabe der Zahlen und Ausgabe des Produkts als double

Wie im HowTo beschrieben, wird *Simulation12.asm* im MARS geöffnet. Es sollen 2 Ganzzahlen eingegeben, konvertiert und dann die erste durch die zweite dividiert werden. Der Quotient soll als double ausgegeben werden.

```
.data
        prompt1: .asciiz "\nBitte die erste Zahl (den Dividenden) eingeben, x = "
        prompt2: .asciiz "\nBitte die zweite Zahl (den Divisor) eingeben, y = "
        message: .asciiz "\nDas Ergebnis der Division (der Quotient) ist x : y = "
```

Im *.data* Teil des Codes werden Strings hinterlegt, die einerseits zur Eingabe der Ganzzahlen auffordern und andererseits die Ausgabe begleiten sollen.

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



Der *.text* Teil beginnt mit der Aufforderung, die erste Zahl, also x, einzugeben:

```
.text
# Ausgabe der ersten Nachricht: prompt1
        li $v0, 4           # der Wert 4 für den syscall bedeutet: print string
        la $a0, prompt1     # lädt die Adresse des ersten Strings in $a0
        syscall

# erste Zahl einlesen und im temporären Register $t0 ablegen
        li $v0, 5           # der Wert 5 für den syscall bedeutet: read integer
        syscall
        move $t0, $v0
```

Der Wert *4* für den *syscall* bedeutet *print strin*g, und die Adresse dieses Strings muss dafür in Register *$a0* geladen werden. Nach Ausgabe der Nachricht *prompt1* kann dann *x* eingelesen werden, dazu dient der Wert *5: read integer* für den *syscall.* Wird dann eine Zahl eingegeben und mit *Enter* bestätigt, liegt sie in *$v0* vor und wird hier, zur weiteren Verwendung, in das temporäre Register *$t0* kopiert:

$a3	7	0x00000000
$t0	8	0x0000000c
$t1	9	0x00000000

Wie man sieht, liegt hier als Beispiel *x = 12* in hexadezimaler Darstellung in *$t0* vor.

```
# erste Zahl zum Coproc 1 transferieren und konvertieren
        mtc1.d $t0, $f4
        cvt.d.w $f4, $f4
```

Mithilfe des Befehls *mtc1.d $t0, $f4* wird *x* nun zum Coprozessor 1 übertragen, wo es dann zu double konvertiert werden kann mit dem Befehl *cvt.d.w $f4, $f4*, das hier den Wert aus dem Floating-Point-Register *$f4* nimmt, zu double konvertiert und wieder in *$f4* ablegt.

Ebenso wird dann als nächstes der Divisor *y = 5* eingelesen, zum Coprozessor transferiert und konvertiert, sodass vor Ausführung des Divisionsbefehls *div* in Zeile 41 *x* und *y* als double bereitliegen:

$f4	0x00000000	0x4028000000000000
$f5	0x40280000	
$f6	0x00000000	0x4014000000000000
$f7	0x40140000	

Hier erst in hexadezimaler Darstellung und dann in dezimaler Darstellung zu erkennen.

$f4	0.0	12.0
$f5	2.625	
$f6	0.0	5.0
$f7	2.3125	

Nun liegen also Dividend und Divisor bereit, und die eigentliche Division kann initiiert werden:

```
# Division durchführen, Ergebnis in $f12 speichern
        div.d $f12, $f4, $f6
```

```
# Division durchführen, Ergebnis in $f12 speichern
div.d $f12, $f4, $f6

# Aus  div.d $f2,$f4,$f6   Floating point division double precision : Set $f2 to double-precision floating point value of $f4 divided by $f6
```

Der *div.d* Befehl nimmt also das zweitgenannte Register, teilt den Inhalt durch den Inhalt des drittgenannten Registers und legt den Quotienten im erstgenannten Register ab, sodass nach Ausführung in $f12 das Ergebnis bereitsteht:

$f11	0.0	
$f12	4.172325E-8	2.4
$f13	2.05	

Erst dezimal, dann hexadezimal gezeigt.

$f11	0x00000000	
$f12	0x33333333	0x4003333333333333
$f13	0x40033333	

Nun muss noch das Ergebnis ausgegeben werden, die Ausgabe wird vom String *message* begleitet:

```
# Ausgabe der dritten Nachricht: message
        li $v0, 4
        la $a0, message
        syscall
```

Wie schon bei der Ausgabe der Aufforderungen *prompt1* und *prompt2* steht hier der Wert *4* für den *syscall* für *print string* und die Adresse des auszugebenden Strings muss in *$a0* geladen werden.

```
# Ausgabe des Quotienten
        li $v0, 3   # der Wert 3 für den syscall bedeutet: $f12 = double to print
        syscall
```

Der Wert 3 für den *syscall* bedeutet, dass in *$f12* ein double steht, der ausgegeben werden soll.

Nun bleibt nur noch das Beenden des Programms, was wie in den anderen Simulationen dieser Reihe *Simulationen mit dem MARS Simulator* auch, über den Wert *10: terminate execution* für den *syscall* passiert:

```
# exit
        li $v0, 10          # der Wert 10 für den syscall bedeutet: exit (terminate execution)
        syscall
```

Die Ausgabe ist im Fenster *Run I/O* unterhalb des *Data Segments* im *execute* Fenster zu finden: