Das zugrundeliegende Programm:

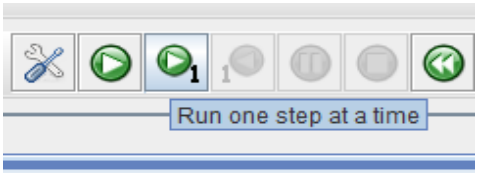
Addition zweier Integer mit Ausgabe der Summe und Eingabe der Summanden

Wie im HowTo beschrieben, wird *Simulation09.asm* im MARS geöffnet. Die Simulationen *07* und *08* haben eine Addition zweier vorgegebener Integerzahlen durchgeführt, nun soll zusätzlich ermöglicht werden, die Summanden selber einzugeben. Die folgenden Codefragmente zeigen schrittweise, wie dafür vorgegangen werden kann.

```
.data
        prompt1: .asciiz "\nBitte die erste Zahl eingeben, x = "
        prompt2: .asciiz "\nBitte die zweite Zahl eingeben, y = "
        message: .asciiz "\nDas Ergebnis der Addition ist x + y = "
```

Im *.data* Teil des Codes werden Strings hinterlegt, die einerseits zur Eingabe der Summanden 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:



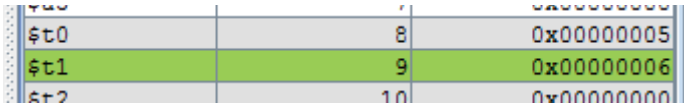
Der *.text* Teil beginnt mit der Aufforderung, die erste Zahl, also x, einzugeben:

```
.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.

Ebenso wird für *y* verfahren, sodass nach Ausführen dieser Codezeilen *x* und *y* in den Registern *$t0* und *$t1* vorliegen und mit dem Additionsbefehl fortgefahren werden kann.
Hier zu sehen mit der beispielhaften Eingabe *x = 5* und *y = 6*:



Der Additionsbefehl schreibt dann die Summe der Inhalte von $t0 und $t1 in das Register $t2:

```
# Addition t2 = t0 + t1
        add $t2, $t0, $t1
```

```
# Addition t2 = t0 + t1
add $t2, $t0, $t1

# e
li
sys
```

```
add $t1,$t2,$t3      Addition with overflow : set $t1 to ($t2 plus $t3)
add $t1,$t2,-100     ADDition : set $t1 to ($t2 plus 16-bit immediate)
add $t1,$t2,100000   ADDition : set $t1 to ($t2 plus 32-bit immediate)
```

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.

```
# Wert ausgeben:
        li $v0, 1           # der Wert 1 für den syscall bedeutet: $a0 = integer to print
        move $a0, $t2       # legt den Wert aus $t2 im Register $a0 für Parameterübergabe ab
        syscall
```

Der Wert *1* für den *syscall* bedeutet, dass im Übergaberegister *$a0* eine Integerzahl steht, die ausgegeben werden soll. Entsprechend wird die Summe aus *$t2* nach *$a0* kopiert, dann der *syscall* initiiert.

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: