Das zugrundeliegende Programm:

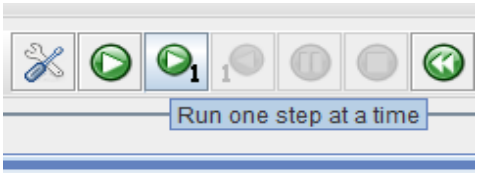
Programm mit Aufruf von Unterprogrammen zur Demonstration der Grundrechenarten

Wie im HowTo beschrieben, wird *Simulation13.asm* im MARS geöffnet. Es sollen 2 Ganzzahlen eingegeben werden, die dann als Parameter für mehrere Unterprogramme dienen, die die Grundrechenarten aus den Simulationen 8 – 12 dieser Reihe realisieren.

```
.data
        prompt1: .asciiz "\nBitte die erste Zahl eingeben, x = "
        prompt2: .asciiz "\nBitte die zweite Zahl eingeben, 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
main:
# 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 Parameterübergabe-Register $a1 ablegen
        li $v0, 5           # der Wert 5 für den syscall bedeutet: read integer
        syscall
        move $a1, $v0
```

Zu erkennen ist ein Unterschied zu den bisherigen Simulationen, wir haben hier ein Hauptprogramm *main*. In den bisherigen kleinen Beispielprogrammen war das nicht so, weil diese so klein und kurz und wenig komplex waren. Da in diesem Programm aber Unterprogramme vorhanden sind, sehen wir hier nun den Anfang des Hauptprogramms, das zugleich der Aufrufende, also der *Caller* für die Unterprogramme sein wird.

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 Parameterübergabe-Register *$a1* kopiert:

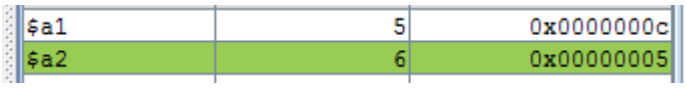
$a0	4	0x10010000
$a1	5	0x0000000c
$a2	6	0x00000000

Wie man sieht, liegt hier als Beispiel *x = 12* in hexadezimaler Darstellung in *$a1* vor.

```
# Ausgabe der zweiten Nachricht prompt2
        li $v0, 4
        la $a0, prompt2 # Adresse des zweiten Strings in $a0 laden
        syscall

# zweite Zahl einlesen und im Parameterübergabe-Register $a2 ablegen
        li $v0, 5
        syscall
        move $a2, $v0
```

Ebenso wird für die zweite Ganzzahl verfahren, sodass nach Ausführen obigen Codefragments *y = 5* als Beispiel im Register *$a2* vorliegt:

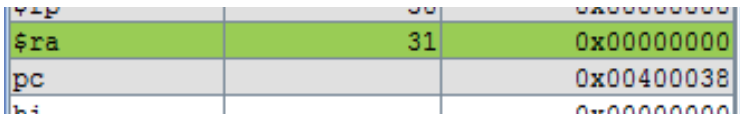


$a1	5	0x0000000c
$a2	6	0x00000005

Da jetzt die Übergabe-Parameter bereitstehen, kann das *main* Programm nun der Reihe nach die Unterprogramme *add, sub, mul* und *div* aufrufen, die ihre Berechnungen ausführen und die Ergebnisse ausgeben:

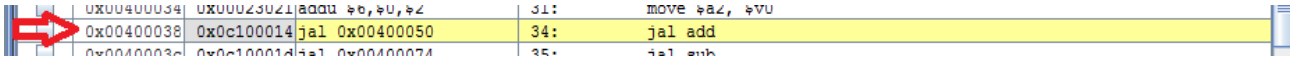
```
# Aufrufe der Unterprogramme
        jal add
        jal sub
        jal mul
        jal div
```

Der Aufruf eines Unterprogramms geschieht jeweils durch den *jal: jump and link* Befehl, der *PC+4* im Register *$ra* speichert, sodass nach Rückkehr aus dem Unterprogramm mit dem nächsten Befehl weitergemacht werden kann. In folgender Abbildung ist zu sehen, dass der *pc* vor Ausführung des *jal* den Wert 0x00400038 enthält:

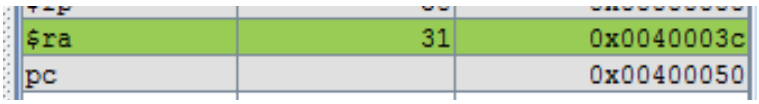


$ra	31	0x00000000
pc		0x00400038
hi		0x00000000

Dies ist gerade die Adresse genau dieses ersten Unterprogrammaufrufs *jal add*, was im MARS sehr gut im Text Segment ersichtlich wird, wie folgender Bildausschnitt zeigt:



Würde nun *jal* den *pc* Inhalt sichern, also in Register *$ra* schreiben, dann würde der Rücksprung wieder zum selben *jal* führen, und das Programm steckte in einer Endlosschleife fest. Entsprechend schreibt *jal* die Adresse des nächsten Befehls, also hier *pc+4 = 0x0040003C* in *$ra,* sodass nach Rückkehr direkt mit dem nächsten Befehl fortgefahren werden kann:



In *pc* steht nun also die Anfangsadresse des Unterprogramms, in diesem Fall ist das *add*, sodass als nächstes die folgenden Codezeilen ausgeführt werden:

```
add:
        .data
                message_add: .asciiz "\nDas Ergebnis der Addition ist x + y = "
        .text
                # Addition
                add $t1, $a1, $a2

                # Ausgabe der Nachricht
                li $v0, 4
                la $a0, message_add
                syscall

                # Ausgabe der Summe
                li $v0, 1
                move $a0, $t1
                syscall

                # Rücksprung
                jr $ra
```

Ebenso wie Hauptprogramme können Unterprogramme einen *.data* und einen *.text* Teil haben. Im *.data* Teil wird hier der String hinterlegt, der die Ausgabe der Summe begleiten soll (Zeile 50). Im *.text* Teil wird die Addition vorgenommen (Zeile 53), der String (Zeilen 56-58) und schließlich die Summe selbst ausgegeben (Zeilen 61-63). Wesentlich ist auch Zeile 66, da sie den Rücksprung zur in *$ra* liegenden Adresse enthält. Durch eben diesen Rücksprung wird mit dem Aufruf des Unterprogramms *sub* fortgefahren, das ebenso wie *add* seine Berechnung durchführt, das Ergebnis ausgibt und zurück in das *main* Programm springt, wo daraufhin der Aufruf zum nächsten Unterprogramm, also *mul* erfolgt. Die Unterprogramme *mul* und *div* werden wie in den Simulationen 12 und 13 dieser Reihe ausgeführt und nach Rückkehr aus *div* bleibt in *main* nur noch, das Programm zu beenden, 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: