Das zugrundeliegende Programm:

Demonstration des Aufrufs von Unterprogrammen

Wie im HowTo beschrieben, wird *Simulation06.asm* im MARS geöffnet.

Der Code zeigt den Aufruf eines Unterprogrammes, der folgende High-Level Code wird in Assembler umgesetzt:

```
int main ()
 {
            int y;
            ...
            y = diffofsum (2,3,4,5);
            ...
 }
 int diffofsums (int f, int g, int h, int i)
 {
            int result;
            result = (f+g)-(h+i);
            return result;
 }
```

Die *main*-Methode ruft also die Funktion *diffofsums* mit 4 integer-Parametern f,g,h und i auf, die dann den Integer-Wert *result = (f+g)-(h+i)* zurückgibt.

```
main:
            # Laden der f,g,h,i in die Übergaberegister a0-a3
            addi $a0, $zero, 2
            addi $a1, $zero, 3
            addi $a2, $zero, 4
            addi $a3, $zero, 5
```

Zuerst werden die Beispielwerte *2,3,4* und *5* in die Parameter-Übergabe-Register a0-a3 geschrieben, hierzu wird der Befehl *addi* in Kombination mit dem zero Register genutzt, das die Konstante *0* enthält.

```
# Aufruf der Unterprozedur diffofsum, Speichern der Rücksprungadresse in $ra
            jal diffofsums
```

Es folgt der Aufruf der Unterprozedur *diffofsum* durch den Befehl ***jal: jump and link***

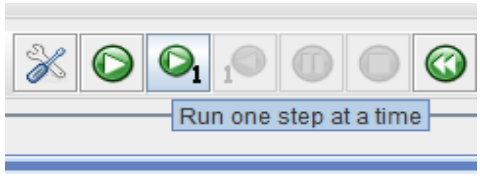
```
# Aufruf der Unterprozedur diffofsum, Speichern der Rücksprungadresse in $ra
jal diffofsums

# B
```

jal target Jump and link : Set $ra to Program Counter (return address) then jump to statement at target address

Wesentlich hierbei ist, dass die Rücksprungadresse im Register ra gespeichert wird.

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



So kann man schön sehen, dass zunächst die Integerwerte in die Register geschrieben werden:

$a0	4	0x00000002
$a1	5	0x00000003
$a2	6	0x00000004
$a3	7	0x00000005

und dann, nach dem ***jal***-Befehl, wie die Rücksprungadresse in ra geschrieben wurde:

$ra	31	0x00000000
pc		0x00400010

vorher

$ra	31	0x00400014
pc		0x00400020

nachher

Woher kommt diese Rücksprungadresse? Es ist der Inhalt des PC (program counter), zudem 4 addiert wird. Würde als Rücksprungadresse der Inhalt von PC eingetragen, ohne, dass 4 addiert wird, würde jeder Rücksprung wieder zum ***jal***-Befehl führen und wir steckten in einer Endlosschleife fest. Also ist die Rücksprungadresse die des nächsten Befehls nach ***jal***.

Was passiert gleichzeitig in PC? Vor Ausführung des ***jal***-Befehls steht in PC *0x004000010,* also die Adresse des ***jal***-Befehls. Nach Ausführung steht dort *0x004000020*, also 16_{10} mehr, warum?
Weil wir durch den Sprung, den ***jal*** ausgelöst hat, nicht mit dem Befehl nach ***jal*** (also PC+4) sondern mit der Unterprozedur *diffofsums* weitermachen, und der erste Befehl dieser Prozedur liegt unter der Adresse *0x004000020*. Sehen kann man das sehr gut im *Text Segment*:

Text Segment

Bkpt	Address	Code	Basic		Source
	0x00400000	0x20040002	addi $4,$0,2	24:	addi $a0, $zero, 2
	0x00400004	0x20050003	addi $5,$0,3	25:	addi $a1, $zero, 3
	0x00400008	0x20060004	addi $6,$0,4	26:	addi $a2, $zero, 4
	0x0040000c	0x20070005	addi $7,$0,5	27:	addi $a3, $zero, 5
	0x00400010	0x0c100008	jal 0x00400020	30:	jal diffofsums
	0x00400014	0x00408020	add $16,$2,$0	33:	add $s0, $v0, $zero
	0x00400018	0x2402000a	addiu $2,$0,10	36:	li $v0, 10
	0x0040001c	0x0000000c	syscall	37:	syscall
	0x00400020	0x00854020	add $8,$4,$5	42:	add $t0, $a0, $a1 # t0 = f+g
	0x00400024	0x00c74820	add $9,$6,$7	43:	add $t1, $a2, $a3 # t1 = h+i
	0x00400028	0x01095022	sub $10,$8,$9	44:	sub $t2, $t0, $t1 # s0 = (f+g)-(h+i)
	0x0040002c	0x01401020	add $2,$10,$0	45:	add $v0, $t2, $zero # v0 = result
	0x00400030	0x03e00008	jr $31	46:	jr $ra # Rücksprung

```
######################## Unterprogramm diffofsums ################################

diffofsums:
        add $t0, $a0, $a1       # t0 = f+g
        add $t1, $a2, $a3       # t1 = h+i
        sub $t2, $t0, $t1       # s0 = (f+g)-(h+i)
        add $v0, $s0, $zero     # v0 = result
        jr $ra                  # Rücksprung
```

Im Unterprogramm *diffofsums* schließlich findet die Berechnung *result = (f+g)-(h+i)* statt, hier für werden die temporären Register t0, t1 und t2 benutzt. Dann wird die Variable *result* im Register v0 abgelegt und es folgt der Rücksprung mit dem Befehl ***jr: jump register unconditionally:***

```
jr $ra                  # Rücksprung
###jr $t1   Jump register unconditionally : Jump to statement whose address is in $t1 ##
```

es wird also zu dem Befehl gesprungen, dessen Adresse im Register ra liegt, wie wir oben gesehen haben, ist das die Adresse des Befehls, der direkt auf ***jal*** folgt.

```
# Benutzen des Rückgabewertes, der in $v0 liegt, $s0 = y
        add $s0, $v0, $zero

        # exit
        li $v0, 10
        syscall
```

Dieser Befehl schreibt den Rückgabewert der Unterprozedur in das Register s0, danach bleibt nur noch, das Programm zu beenden. Der Wert 10 für den syscall bedeutet „*terminate execution*“.

Abschließende Registerbelegung (hexadezimal):

$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x00000002
$a1	5	0x00000003
$a2	6	0x00000004
$a3	7	0x00000005
$t0	8	0x00000005
$t1	9	0x00000009
$t2	10	0xfffffffc
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0xfffffffc
$s1	17	0x00000000

Abschließende Registerbelegung (dezimal):

$v0	2	10
$v1	3	0
$a0	4	2
$a1	5	3
$a2	6	4
$a3	7	5
$t0	8	5
$t1	9	9
$t2	10	-4
$t3	11	0
$t4	12	0
$t5	13	0
$t6	14	0
$t7	15	0
$s0	16	-4
$s1	17	0

Der verwendete Code entstammt (leicht modifiziert) dem Kapitel 6 des Buches
Harris & Harris: Digital Design and Computer Architecture, Elsevier, 2012