Das zugrundeliegende Programm:

Schleife mit Verwendung von set on less than: slt

Wie im HowTo beschrieben, wird *Simulation04.asm* im MARS geöffnet.

Der Code zeigt die Verwendung von ***slt*** in einer Schleife. Der folgende High-Level-Code wird in Assembler umgesetzt:

```
int sum = 0;
for (i = 1; i <101; i = i*2)
        sum = sum + i;
```

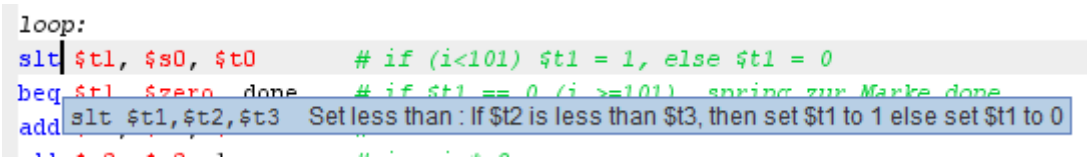
Es werden also die Vielfachen von 2 unter 100 addiert.

```
addi $s1, $zero, 0   # Initialisierung der Summe sum in $s1 mit 0
addi $s0, $zero, 1   # Initialisierung des Zählers i in $s0 mit 1
addi $t0, $0, 101    # Obergrenze 101
```

Zuerst werden die Variable *sum* und der Zähler *i* initialisiert. In Mips Assembler erfolgt das häufig durch den *addi* Befehl, der hier die immediate Werte *0*, bzw *1* zur Konstanten *0* im Register $zero addiert und die Summe in Register s1 bzw. s0 ablegt. Die Obergrenze für den Zähler *i* in der Schleife wird mit dem Wert *101* in das temporäre Register t0 geschrieben.

```
loop:
slt $t1, $s0, $t0          # if (i<101) $t1 = 1, else $t1 = 0
beq $t1, $zero, done       # if $t1 == 0 (i >=101), spring zur Marke done
add $s1, $s1, $s0           sum = sum + i
sll $s0, $s0, 1            # i = i * 2
j loop                     # Sprung zu loop
```

In der ersten Zeile dieses Codefragments ist die ***slt*** Anweisung zu sehen: ***slt $t1, $s0, $t0***



Die ***slt*** Anweisung macht also Folgendes:
Wird die Abfrage *s0 < t0* (hier also *i < 101*) zu JA ausgewertet, dann schreibt ***slt*** eine *1* (true) in Register t1, anderenfalls, wenn die Abfrage zu NEIN ausgewertet wird, dann schreibt ***slt*** eine *0* (false) in Register t1.

In der nächsten Zeile obigen Codefragments folgt die ***beq*** Anweisung: ***branch if equal***.

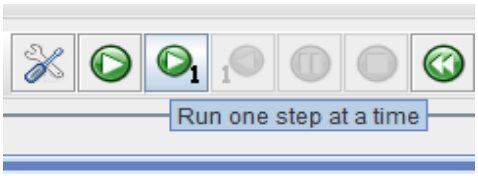
```
beq $t1, $zero, done    # if $t1 == 0 (i >=101), spring zur Marke done
add $s1  $s1  $s0       # sum = sum + i
sll
j l
```

`beq $t1,$t2,label`	Branch if equal : Branch to statement at label's address if $t1 and $t2 are equal
`beq $t1,-100,label`	Branch if EQual : Branch to statement at label if $t1 is equal to 16-bit immediate
`beq $t1,100000,label`	Branch if EQual : Branch to statement at label if $t1 is equal to 32-bit immediate

Die *beq* Anweisung lässt das Programm also zum Label *done* (siehe Seite 3) springen, gdw. der durch die ***slt*** Anweisung in das Register *t1* geschrieben Wert *0* ist, wenn also $t1 = 0$ und demnach $i \geq 101$. Solange $t1=1$, also $i<101$, wird nicht gesprungen und das Programm setzt die Ausführung in der nächsten Zeile mit dem *add* Befehl fort.

Der *add* Befehl addiert den Zählerwert *i* zur Variable *sum* in Register *s1,* und der folgende Schiebebefehl *sll* multipliziert den Zählerwert *i* in Register *s0* mit *2,* indem um ein Bit nach links geschoben und rechts eine *0* eingefügt wird. (Zur Ausführung des Schiebebefehls mehr in Simulation 03 dieser Reihe *Simulationen mit dem MARS Simulator.)*

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



Dabei kann man schön die Schleife beobachten, und wie dabei die Variable *sum* in Register *s0* und der Zähler *i* in Register *s1* wachsen, es folgt eine Tabelle mit den Registerbelegungen in Dezimaldarstellung:

sum in s1	***i in s0***	
-	-	Programmstart
0	1 (<101)	Erster Eintritt in die Schleife
1	2 (<101)	Zweiter Eintritt in die Schleife
3	4 (<101)	Dritter Eintritt in die Schleife
7	8 (<101)	Vierter Eintritt in die Schleife
15	16 (<101)	Fünfter Eintritt in die Schleife
31	32 (<101)	Sechster Eintritt in die Schleife
63	64 (<101)	Siebter Eintritt in die Schleife
127	128 (≥101)	Abbruch!

Da *i* nun größer als *101* ist, wird durch die ***slt*** Anweisung eine *0* in Register *t1* geschrieben und die *beq* Anweisung lässt das Programm zur Marke *done* springen.

Was bleibt ist das Beenden der Programmausführung:

```
done:
li $v0, 10          # Der Wert 10 für den syscall bedeutet:
syscall             # terminate execution
```

Der Wert 10 für den syscall bedeutet „*terminate execution*“ und beendet das Programm.

Dieser Beispielcode entstammt (leicht modifiziert) einem Beispiel im sechsten Kapitel des Buches *Harris & Harris: Digital Design and Computer Architecture, Elsevier, 2012*