Das zugrundeliegende Programm:

Demonstration der Schiebeoperationen sll, srl, sra

Wie im HowTo beschrieben, wird *Simulation03.asm* im MARS geöffnet.

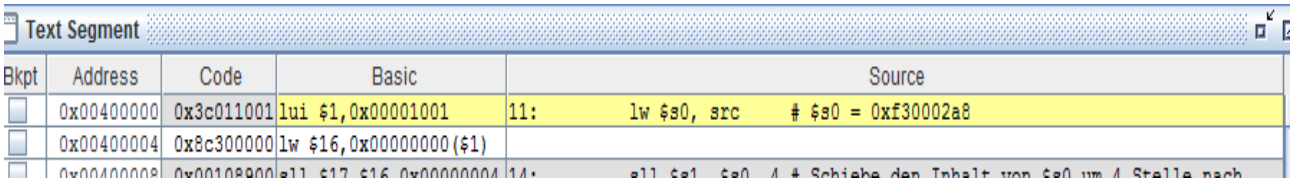
Der Code zeigt die Verwendung der Schiebebefehle ***sll, srl*** und ***sra***. Im Folgenden werden einzelne Codeabschnitte näher erläutert.

```
.data
        src:    .word 0xf30002a8
        sll:    .asciiz "\nshift left logical\n11110011000000000000001010101000 um 4 Stellen = \n"
        srl:    .asciiz "\nshift right logical\n11110011000000000000001010101000 um 4 Stellen = \n"
        sra:    .asciiz "\nshift right arithmetic\n11110011000000000000001010101000 um 4 Stellen = \n"
```

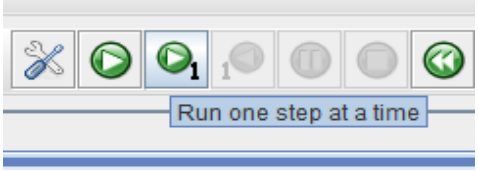
Die Konstanten werden hinterlegt, die im Programmablauf benötigt werden, als Beispielzahl wurde 0xf30002a8 willkürlich ausgewählt. Die Strings sll, srl und sra sind nicht für die Ausführung des Programms nötig, sondern dienen der Übersichtlichkeit und Verständlichkeit der Ausgabe.

```
.text
        lw $s0, src             # $s0 = 0xf30002a8
```

Die Beispielzahl *src* wird in Register s0 geladen, sehr schön kann man das im „*Execute*“ Fenster im *Text Segment* verfolgen:



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



Im rechten Bereich unter „Register“ findet man nun nach Ausführung der Codezeile das mit der Beispielzahl beschriebene Register s0,

$t7	15	0x00000000
$s0	16	0xf30002a8
$s1	17	0x00000000

sodass nun die eigentlichen Operationen beginnen können:

```
# Schiebeoperationen
        sll $s1, $s0, 4      # Schiebe den Inhalt von $s0 um 4 Stelle nach links, fülle rechts mit 0en auf
        srl $s2, $s0, 4      # Schiebe den Inhalt von $s0 um 4 Stellen nach rechts, fülle links mit 0en auf
        sra $s3, $s0, 4      # Schiebe den Inhalt von $s0 um 4 Stellen nach rechts, fülle links mit dem Vorzeichenbit auf
```

wobei wesentlich ist, worin die Unterschiede in den Befehlen liegen:

- **shift left logical, logisches Linksschieben:**

```
# Schiebeoperationen
sll $s1, $s0, 4 # Schiebe den Inhalt von $s0 um 4 Stelle nach links, fülle rechts mit 0en auf
```

```
sll $t1,$t2,10   Shift left logical : Set $t1 to result of shifting $t2 left by number of bits specified by immediate
```

  es wird um die Anzahl Bits nach links geschoben, die durch den im Befehl angegeben immediate Wert bestimmt wird, in diesem Fall sind das 4. Dieselbe Anzahl 0en wird rechts nachgeschoben.

- **shift right logical, logisches Rechtsschieben:**

```
srl $s2, $s0, 4 # Schiebe den Inhalt von $s0 um 4 Stellen nach rechts, fülle links mit 0en auf
```

```
srl $t1,$t2,10   Shift right logical : Set $t1 to result of shifting $t2 right by number of bits specified by immediate
```

  es wird um die Anzahl Bits nach rechts geschoben, die durch den im Befehl angegebenen immediate Wert bestimmt wird, in diesem Fall wieder 4. Dieselbe Anzahl 0en wird von links nachgeschoben.

- **shift right arithmetic, arithmetisches Rechtsschieben:**

```
sra $s3, $s0, 4 # Schiebe den Inhalt von $s0 um 4 Stellen nach rechts, fülle links mit dem Vorzeichenbit auf
```

```
sra $t1,$t2,10   Shift right arithmetic : Set $t1 to result of sign-extended shifting $t2 right by number of bits specified by immediate
```

  es wird um die durch den immediate Wert (hier 4) bestimmte Anzahl Bits nach rechts geschoben, im Gegensatz zum *srl* aber wird nicht pauschal mit 0en aufgefüllt, sondern es wird das Vorzeichen erweitert, im Falle unserer Beispielzahl ist das die 1.

Nach diesen Operationen können die Ergebnisse, die in den Registern s1-s3 vorliegen, nun ausgegeben werden. Damit man das bitweise Schieben besser erkennen kann, erfolgt die Ausgabe nicht hexadezimal, sondern binär. Mittels folgenden Codefragments wird die Ausgabe des Ergebnisses der sll-Operation initiiert:

```
# Ausgabe sll
        la $a0, sll
        li $v0, 4
        syscall
        move $a0, $s1
        li $v0, 35
        syscall
```

Zuerst wird die Adresse des Strings *sll* in das Register a0 geladen, dann der Wert 4 in das Register v0. Der syscall mit dem Wert 4 steht für „*print string*“ und gibt aus, was unter der in a0 liegenden Adresse gespeichert ist. Schließlich wird das Ergebnis der sll-Operation von s1 nach a0 kopiert, um es dann mit dem syscall 35 („*print integer in binary*“) als Dualzahl auszugeben. Die Ausgaben der beiden anderen Operationen folgen demselben Prinzip.

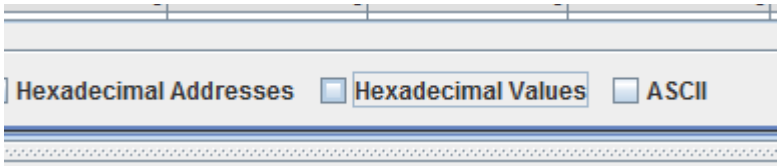
```
# exit
        li $v0, 10
        syscall
```

Der Wert 10 für den syscall bedeutet: „*exit (terminate execution)*“ und der syscall mit diesem Wert beendet die Ausführung des Programms.

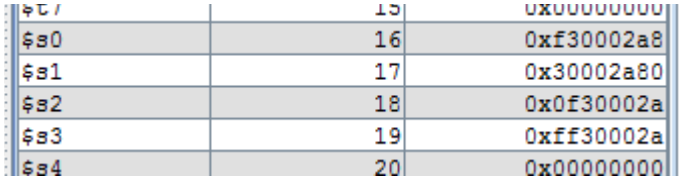
Die Ausgabe des Programms ist die folgende:

```
shift left logical
11110011000000000000001010101000 um 4 Stellen =
00110000000000000010101010000000
shift right logical
11110011000000000000001010101000 um 4 Stellen =
00001111001100000000000000101010
shift right arithmetic
11110011000000000000001010101000 um 4 Stellen =
11111111001100000000000000101010
-- program is finished running --
```

Ebenfalls lassen sich nach Ausführung im rechten Bereich die Werte der Register ablesen, wobei man wahlweise die Darstellung als Hexadezimal-Zahl oder als Dezimalzahl wählen kann, umgeschaltet wird das durch Setzen des Häkchens bei „*Hexadecimal Values*“ am unteren Rand des *Data Segment*s:



Abschließende Registerbelegung:



$s0	16	0xf30002a8
$s1	17	0x30002a80
$s2	18	0x0f30002a
$s3	19	0xff30002a
$s4	20	0x00000000