Das zugrundeliegende Programm:

Demonstration der logischen Befehle and, or, xor, nor

Wie im HowTo beschrieben, wird *Simulation01.asm* im MARS geöffnet.

Der Code zeigt die Verwendung der logischen Befehle für die ***UND***- Verknüpfung, die ***ODER***-Verknüpfung, die ***EXKLUSIV-ODER***-Verknüpfung und die ***NICHT-ODER***-Verknüpfung. Im Folgenden werden einzelne Codeabschnitte näher erläutert.

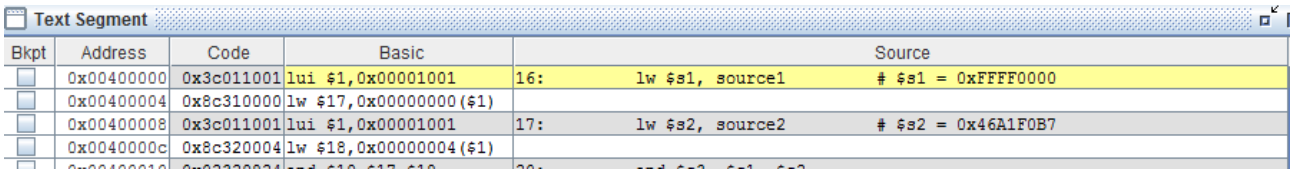
```
.data
        source1:  .word 0xFFFF0000
        source2:  .word 0x46A1F0B7

        and:      .asciiz "\n11111111111111110000000000000000 AND\n01000110101000011111000010110111 =\n"
        or:       .asciiz "\n11111111111111110000000000000000 OR\n01000110101000011111000010110111 =\n"
        xor:      .asciiz "\n11111111111111110000000000000000 XOR\n01000110101000011111000010110111 =\n"
        nor:      .asciiz "\n11111111111111110000000000000000 NOR\n01000110101000011111000010110111 =\n"
        nl:       .asciiz "\n"
```

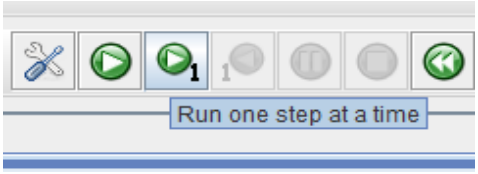
Die Konstanten werden hinterlegt, die im Programmablauf benötigt werden, als Beispielzahlen wurden 0xFFFF0000 und 0x46A1F0B7 willkürlich ausgewählt. Die Strings and, or, xor, nor, nl sind für den Programmablauf nicht notwendig, sondern dienen der Übersichtlichkeit und Verständlichkeit der Ausgabe.

```
.text
        lw $s1, source1          # $s1 = 0xFFFF0000
        lw $s2, source2          # $s2 = 0x46A1F0B7
```

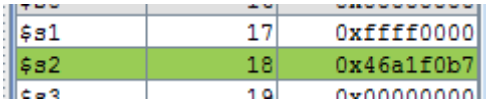
Die hinterlegten Zahlen source1 und source2 werden in die Register s1 und s2 geladen, sehr schön kann man das im „*Execute*“ Fenster im *Text Segment* verfolgen:



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 beiden Codezeilen die mit den Source-Werten gefüllten Register s1 und s2,



$s1	17	0xffff0000
$s2	18	0x46a1f0b7

sodass nun die eigentlichen Operationen beginnen können,

```
# Operationen
        and $s3, $s1, $s2
        or $s4, $s1, $s2
        xor $s5, $s1, $s2
        nor $s6, $s1, $s2
```

wobei die Ergebnisse in den Registern s3 bis s6 gespeichert werden. Die Operationen verknüpfen bitweise, deshalb erfolgt die Ausgabe der Ergebnisse nicht hexadezimal oder dezimal, sondern binär, sodass gut erkennbar wird, was die einzelnen logischen Operationen tun. Der folgende Code dient der Ausgabe des Strings „*and*“ und des Ergebnisses der UND-Verknüpfung, das in Register s3 gespeichert ist:

```
# Ausgabe and

        la $a0, and
        li $v0, 4
        syscall
        move $a0, $s3
        li $v0, 35
        syscall
        la $a0, nl
        li $v0, 4
        syscall
```

Für jeden syscall wird ein Wert benötigt, der bestimmt, welche Art syscall ausgeführt werden soll. Die 4 bedeutet: „*print string*“ mit dem Argument:„*$a0 = address of null-terminated string to print*“, das heißt also, dass in Register a0 die Adresse eines Strings steht, der ausgegeben werden soll. Die 35 bedeutet: „*print integer in binary“* mit dem Argument: „*$a0 = integer to print*“, das heißt also, dass in Register a0 eine Integer-Zahl liegt, die als Binärzahl ausgegeben werden soll. Diese Argumente müssen vor dem syscall in a0 geschrieben werden.

Die Codefragmente für die Ausgabe der Ergebnisse der anderen Operationen funktionieren ebenso, sodass als letztes noch das Ende des Programms steht:

```
# 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 ausgeführten Programms sieht folgendermaßen aus und zeigt die bitweise Verknüpfung der Source-Werte entsprechend der logischen Befehle:

```
11111111111111110000000000000000 AND
01000110101000011111000010110111 =
01000110101000010000000000000000

11111111111111110000000000000000 OR
01000110101000011111000010110111 =
11111111111111111111000010110111

11111111111111110000000000000000 XOR
01000110101000011111000010110111 =
10111001010111101111000010110111

11111111111111110000000000000000 NOR
01000110101000011111000010110111 =
00000000000000000000111101001000
-- program is finished running --
```

Gleichzeitig kann man im rechten Bereich die abschließende Registerbelegung sehen, wobei man wahlweise die Darstellung als Hexadezimal-Zahl oder als Dezimalzahl wählen kann:

$s0	16	0x00000000
$s1	17	0xffff0000
$s2	18	0x46a1f0b7
$s3	19	0x46a10000
$s4	20	0xfffff0b7
$s5	21	0xb95ef0b7
$s6	22	0x00000f48
$s7	23	0x00000000

$s0	16	0
$s1	17	-65536
$s2	18	1185018039
$s3	19	1184956416
$s4	20	-3913
$s5	21	-1184960329
$s6	22	3912
$s7	23	0

umgeschaltet wird das durch Setzen des Häkchens bei „*Hexadecimal Values*“ am unteren Rand des *Data Segment*s: