Logik, Sprünge und Sprungmarken

Der Befehlssatz des Prozessors enthält die Befehle AND, OR, XOR, TEST und NOT der booleschen Logik. D

Das Format für diese Befehle ist folgendes:

Anweisung	Format im Programmcode
AND	AND operand1, operand2
OR	OR operand1, operand2
XOR	XOR operand1, operand2
TEST	TEST operand1, operand2
NOT	NOT operand1

Der erste Operand kann entweder in einem Register oder im Speicher sein. Der zweite Operand kann entweder in einem Register/Speicher oder ein unmittelbarer (konstanter) Wert sein. Speicher-zu-Speicher-Operationen sind nicht möglich.

Die Operatoren werden bitweise ausgeführt je nach Ergebis werden die Flags CF, OF, PF, SF oder ZF gesetzt.

Die AND-Instruktion

Die AND-Anweisung vergleicht zwei Operanden indem sie eine bitweise AND-Operation durchführt. Die bitweise UND-Verknüpfung ergibt 1, wenn die übereinstimmenden Bits beider Operanden 1 sind, andernfalls ergibt sie 0.

Das Ergebnis der Operation wird im ersten Operand gespeichert.

Beispiel:

```
      Operand1:    1010 0101
      Operand2:    1000 0011
---------------------------------
AND -> Operand1: 1000 0001
```

Die AND-Verknüpfung kann verwendet werden, um ein oder mehrere Bits zu löschen. Beispiel: Das BL-Register enthält 0011 1010. Wenn du die höherwertigen Bits auf Null setzen willst, verknüpfe BL mit 0FH:

```
  mov BL, 00111010B; Schreibe 00111010 ins Register BL
  and BL, 0FH      ; Jetzt steht in BL das Bitmuster 00001010
```

(A1)

Analysiere den folgenden Code. Welchen Inhalt hat das Register BL nach der AND Operation? Welche Ausgabe erzeugt das Programm?

```
 section .data
tabelle TIMES 10 DW 97

section .text
   global _start     ;must be declared for linker (ld)
_start:              ;tell linker entry point


   MOV BL, 01111010B ;
   AND BL, 0F0H      ;

   MOV [tabelle], BL  ; Was passiert hier?

   ;tabelle ausgeben
   mov  edx,20     ;message length
   mov  ecx,tabelle   ;message to write
   mov  ebx,1       ;file descriptor (stdout)
   mov  eax,4       ;system call number (sys_write)
   int  0x80        ;call kernel

   mov  eax,1       ;system call number (sys_exit)
   int  0x80        ;call kernel
```

(A2)

An folgendem Beispiel kann man sich einige neue Möglichkeiten erschließen - das folgende Programm testet eine Zahl, ob sie gerade oder ungerade ist.

```
section .data
even_msg  db  'Gerade Zahl!'
len1  equ  $ - even_msg

odd_msg db  'Ungerade Zahl!'
len2  equ  $ - odd_msg

section .text
   global _start
```

```
_start:                    ;tell linker entry point
   mov   ax, 8h           ;getting 8 in the ax
   and   ax, 1              ;and ax with 1
   jz    evnn


   mov   eax, 4             ;system call number (sys_write)
   mov   ebx, 1             ;file descriptor (stdout)
   mov   ecx, odd_msg       ;message to write
   mov   edx, len2          ;length of message
   int   0x80               ;call kernel
   jmp   exitprog

evnn:

   mov   ah,  09h
   mov   eax, 4             ;system call number (sys_write)
   mov   ebx, 1             ;file descriptor (stdout)
   mov   ecx, even_msg      ;message to write
   mov   edx, len1          ;length of message
   int   0x80               ;call kernel

exitprog:

   mov   eax,1              ;system call number (sys_exit)
   int   0x80               ;call kernel
```

- Teste das Programm und überprüfe, ob es korrekte Ausgaben erzeugt.
- Erweitere das Programm um eine Benutzereingabe, die ein Zeichen als Eingabe akzeptiert. Wie kannst du Zahlen testen die größer sind als 10?

(A3)

Bearbeite die Seite https://www.tutorialspoint.com/assembly_programming/assembly_logical_instructions.htm, um Informationen über die weiteren logischen Operatoren zu erhalten.

Informationen zu Bedingungen und Sprungbefehlen findest du auf dieser Seite:

https://www.tutorialspoint.com/assembly_programming/assembly_conditions.htm