

Manuals (Beispiel)

INSTRUCTION SET REFERENCE



AND—Logical AND

Opcode	Instruction	Description
24 <i>ib</i>	AND AL, <i>imm8</i>	AL AND <i>imm8</i>
25 <i>iw</i>	AND AX, <i>imm16</i>	AX AND <i>imm16</i>
25 <i>id</i>	AND EAX, <i>imm32</i>	EAX AND <i>imm32</i>
80 /4 <i>ib</i>	AND <i>r/m8,imm8</i>	<i>r/m8</i> AND <i>imm8</i>
81 /4 <i>iw</i>	AND <i>r/m16,imm16</i>	<i>r/m16</i> AND <i>imm16</i>
81 /4 <i>id</i>	AND <i>r/m32,imm32</i>	<i>r/m32</i> AND <i>imm32</i>
83 /4 <i>ib</i>	AND <i>r/m16,imm8</i>	<i>r/m16</i> AND <i>imm8</i> (<i>sign-extended</i>)
83 /4 <i>ib</i>	AND <i>r/m32,imm8</i>	<i>r/m32</i> AND <i>imm8</i> (<i>sign-extended</i>)
20 / <i>r</i>	AND <i>r/m8,r8</i>	<i>r/m8</i> AND <i>r8</i>
21 / <i>r</i>	AND <i>r/m16,r16</i>	<i>r/m16</i> AND <i>r16</i>
21 / <i>r</i>	AND <i>r/m32,r32</i>	<i>r/m32</i> AND <i>r32</i>
22 / <i>r</i>	AND <i>r8,r/m8</i>	<i>r8</i> AND <i>r/m8</i>
23 / <i>r</i>	AND <i>r16,r/m16</i>	<i>r16</i> AND <i>r/m16</i>
23 / <i>r</i>	AND <i>r32,r/m32</i>	<i>r32</i> AND <i>r/m32</i>

Manuals (Beispiel)

Description

This instruction performs a bitwise AND operation on the destination (first) and source (second) operands and stores the result in the destination operand location. The source operand can be an immediate, a register, or a memory location; the destination operand can be a register or a memory location. Two memory operands cannot, however, be used in one instruction. Each bit of the instruction result is a 1 if both corresponding bits of the operands are 1; otherwise, it becomes a 0.

Operation

DEST ← DEST AND SRC;

Flags Affected

The OF and CF flags are cleared; the SF, ZF, and PF flags are set according to the result. The state of the AF flag is undefined.

Manuals (Beispiel)



AND—Logical AND (Continued)

Protected Mode Exceptions

#GP(0)	If the destination operand points to a nonwritable segment. If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit. If the DS, ES, FS, or GS register contains a null segment selector.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

Code-Generierungs-Optionen (GCC)

-fcall-used-REG

Treat the register named REG as an allocable register that is clobbered by function calls. It may be allocated for temporaries or variables that do not live across a call. Functions compiled this way will not save and restore the register REG.

-fcall-saved-REG

Treat the register named REG as an allocable register saved by functions. It may be allocated even for temporaries or variables that live across a call. Functions compiled this way will save and restore the register REG if they use it.

-fpack-struct

Pack all structure members together without holes. Usually you would not want to use this option, since it makes the code suboptimal, and the offsets of structure members won't agree with system libraries.

Code-Generierungs-Optionen (GCC)

-fomit-frame-pointer

Don't keep the frame pointer in a register for functions that don't need one. This avoids the instructions to save, set up and restore frame pointers; it also makes an extra register available in many functions.

-mreg-alloc=REGS

Control the default allocation order of integer registers. The string REGS is a series of letters specifying a register. The supported letters are: 'a' allocate EAX; 'b' allocate EBX; 'c' allocate ECX; 'd' allocate EDX; 'S' allocate ESI; 'D' allocate EDI; 'B' allocate EBP.

-mregparm=NUM

Control how many registers are used to pass integer arguments. By default, no registers are used to pass arguments, and at most 3 registers can be used.

Code-Generierungs-Optionen (GCC)

-malign-loops=NUM

Align loops to a 2 raised to a NUM byte boundary. If '-malign-loops' is not specified, the default is 2.

-malign-jumps=NUM

Align instructions that are only jumped to to a 2 raised to a NUM byte boundary. If '-malign-jumps' is not specified, the default is 2.

-malign-functions=NUM

Align the start of functions to a 2 raised to NUM byte boundary. If '-malign-functions' is not specified, the default is 2.

...

Linux-Beispiel (hello.c -> I/O)

```
void main(void)                                .section .rodata
{                                                .LC0:
  write(1, "Hello!\n", 7);                      .string "Hello!\n"
  _exit(0);                                     .text
}                                                .align 4
                                                .globl main
main:
  pushl $7
  pushl $.LC0
  pushl $1
  call write
  addl $12, %esp
  pushl $0
  call _exit
  addl $4, %esp
  ret
```

Linux-Beispiel (hello.c -> I/O)

```
804c140 <write>:
804c140:      53                push %ebx
804c141:      8b 54 24 10      mov 0x10(%esp,1),%edx
804c145:      8b 4c 24 0c      mov 0xc(%esp,1),%ecx
804c149:      8b 5c 24 08      mov 0x8(%esp,1),%ebx
804c14d:      b8 04 00 00 00  mov $0x4,%eax
804c152:      cd 80           int $0x80
804c154:      5b                pop %ebx
804c155:      3d 01 f0 ff ff  cmp $0xfffff001,%eax
804c15a:      0f 83 10 54 00 00  jae ...
804c160:      c3                ret

804bf70 <_exit>:
804bf70:      89 da           mov %ebx,%edx
804bf72:      8b 5c 24 04      mov 0x4(%esp,1),%ebx
804bf76:      b8 01 00 00 00  mov $0x1,%eax
804bf7b:      cd 80           int $0x80
...

```

Linux-Beispiel (hello.c -> I/O)

linux-2.6.7/arch/i386/kernel/entry.S:

```
system_call: .globl system_call
    pushl %eax
    pushl %es; pushl %ds;
    pushl %eax;
    pushl %ebp; pushl %edi; pushl %esi
    pushl %edx; pushl %ecx; pushl %ebx
    movl $KERNEL_DS, %edx
    movl %edx, %ds; movl %edx, %es
    call *sys_call_table(,%eax,4)
    movl %eax, 18(%esp)
    popl %ebx; popl %ecx; popl %edx
    popl %esi; popl %edi; popl %ebp
    popl %eax
    popl %ds; popl %es
    addl $4, %esp
    iret

sys_call_table:
    .long 0
    .long sys_exit
    .long sys_fork
    .long sys_read
    .long sys_write
    .long sys_open
    ...
```

Linux-Beispiel (hello.c -> I/O)

linux-2.6.7/include/linux/{compiler,linkage}.h:

```
#define asmlinkage __attribute__((regparm(0)))
#define __user __attribute__((noderef, address_space(1)))
```

linux-2.6.7/fs/read_write.c:

```
asmlinkage int
sys_write(int fd, char __user * buf, unsigned int count)
{
    ... = vfs_write(...);
}

int
vfs_write(...)
{
    ... = tty_write(...);
}

...
```

Linux-Beispiel (hello.c -> I/O)

Vergleiche linux-2.6.7/drivers/char/:*

```
char ser_obuf[256];
unsigned ser_ocount, ser_ohhead, ser_otail, ser_ocount;

char ser_ibuf[256];
unsigned ser_icount, ser_ihead, ser_itail, ser_icount;

void ser_interrupt(void) {
    ...
}

int ser_write(..., char __user *buffer, int count, ...) {
    ...
}

int ser_read(...) {
    ...
}
```

Linux-Beispiel (hello.c -> I/O)

ähnlich linux-2.6.7/drivers/char/:*

```
int ser_write(..., char __user *buffer, int count, ...) {
    unsigned long eflags; char c;

    /* Save IE-Register, disable interrupts. */
    eflags = save_flags(); cli();
    /* Put characters into buffer. */
    while (sizeof(ser_obuf) < ser_ocount + count) schedule();
    while (0 < count--) {
        c = get_user(buffer++);
        ser_ohhead = (ser_ohhead+1) % sizeof(ser_obuf); ser_ocount++;
    }
    /* Switch on serial interrupts. */
    outb(inb(SER_CONTROL) | SER_OUT_INTERRUPT, SER_CONTROL);
    /* Restore interrupts. */
    restore_flags(eflags);
}
```

Linux-Beispiel (hello.c -> I/O)

Vergleiche linux-2.6.7/arch/i386/kernel/entry.S:

```
irq03:  pushl $3-256
        pushl %es; pushl %ds
        pushl %eax
        pushl %ebp; pushl %edi; pushl %esi
        pushl %edx; pushl %ecx; pushl %ebx
        movl $KERNEL_DS, %edx
        movl %edx, %ds; movl %edx, %es
        call do_IRQ
        call schedule
        popl %ebx; popl %ecx; popl %edx
        popl %esi; popl %edi; popl %ebp
        popl %eax
        popl %ds; popl %es
        addl $4, %esp
        iret
```

Linux-Beispiel (hello.c -> I/O)

linux-2.6.7/include/linux/{compiler,linkage}.h:

```
#define asmlinkage __attribute__((regparm(0)))
```

Vergleiche linux-2.6.7/arch/i386/kernel/irq.c:

```
asmlinkage void
do_IRQ(void) {
    ...
    ser_interrupt();
    ...
}
```

Linux-Beispiel (hello.c -> I/O)

Vergleiche linux-2.6.7/drivers/char/:*

```
void ser_interrupt(void) {
    if ((inb(SER_STATUS) & SER_OUT_POSSIBLE) && 0 < ser_count) {
        /* Get character from buffer. */
        c = ser_obuf[ser_otail];
        ser_ocount--; ser_otail = (ser_otail+1) % sizeof(ser_obuf);
        /* Send character. */
        outb(c, SER_OUT);
    } else {
        /* Disable further interrupts. */
        outb(inb(SER_CONTROL) & ~SER_OUT_INTERRUPT, SER_CONTROL);
    }
    if ((inb(SER_STATUS) & SER_IN_POSSIBLE) {
        ...
    }
}
```

Linux-Beispiel (hello.c -> I/O)

```
inb: .globl inb
      movl 4(%esp), %edx
      inb %edx, %al
      ret

outb: .globl outb
      movl 4(%esp), %eax
      movl 8(%esp), %edx
      outb %al, %dx
      ret

get_user: .globl get_user
          movl $USER_DS, %eax
          movl %eax, %fs
          movl 8(%esp), %eax
          movb %fs:(%eax), %al
          ret
```

Linux-Beispiel (hello.c -> I/O)

```
cli: .globl cli
     cli
     ret

save_flags: .globl save_flags
           pushfl
           popl %eax
           ret

restore_flags: .globl restore_flags
              movl 4(%esp), %eax
              pushl %eax
              popfl
              ret
```

Linux-Beispiel (hello.c -> I/O)

Vergleiche: linux-2.6.7/kernel/sched.c:

```
asmlinkage void schedule(void) { ... switch_to(prev, next); ... }
```

Vergleiche: linux-2.6.7/include/asm-i386/system.h:

```
switch_to: .globl switch_to
           movl 8(%esp), %edx      /* next */
           movl 4(%esp), %eax     /* prev */
           pushfl
           pushl %ebp
           movl %esp, ...(%eax)
           movl ...(%edx), %esp
           popl %ebp
           popfl
           ret
```