

Assembler

Dr.-Ing. Volkmar Sieh

Department Informatik 4
Verteilte Systeme und Betriebssysteme
Friedrich-Alexander-Universität Erlangen-Nürnberg

WS 2017/2018



- Assemblerprogrammierung schon Teil von GRa
- viel zu wenig für das genaue Verständnis von VMs
- hier: detaillierte Beschreibung des Intel/AMD/Cyrix/... 80x86
 - Assemblerprogrammierung
 - Interrupts/Exceptions/System-Calls
 - Segmentierung
 - MMU-Programmierung
 - ohne Erweiterungen (MMX, SSE, ...)
 - ohne Floating Point Unit
 - nur 32-Bit-Mode
 - nur Protected-Mode



Hier nur Beispiele für Intel/AMD/Cyrix/... 80x86-Architektur.

Gründe:

- 80x86-Architektur bei virtuellen Maschinen weit verbreitet.
- Alle benötigten Tools unter Linux (CIP-Pool) für eigene Versuche verfügbar.

Motto: „Wer 80x86-Architektur virtualisieren kann, kann auch alle anderen Architekturen virtualisieren...”



- Intel/AMD/Cyrix/...-Handbücher in „Intel“-Schreibweise
- Linux-Tools in „AT&T“-Schreibweise

Vergleich:

Intel-Syntax:

```
add al, [x]
sub [eax+8], ax
mov word [eax+ebx*4], 42
```

AT&T-Syntax:

```
addb x, %al
subw %ax, 8(%eax)
movw $42, (%eax, %ebx, 4)
```



Hier: Schreibweise gemäß AT&T-Assembler:

- Quell-Operand steht links
- Ziel-Operand steht rechts
- Register mit „%“
- Immediate Operands mit „\$“
- Operationsbreite „b“ (8-Bit), „w“ (16-Bit), „l“ (32-Bit)

Beispiele:

```
addl $1, %eax
```

```
incw %cx
```

```
movb %al, %bl
```

```
xorl %edx, %edx
```



Die 80x86-CPU's kennen (u.a.) folgende Befehle:

- `mov, movzbw, movzbl, movzwl, movsbw, movsbl, movswl, xchg, lea`
- `in, out`
- `add, adc, sub, sbb, mul, div, inc, dec, neg`
- `and, or, xor, not`
- `shl, shr, asr, rol, ror, rcl, rcr`
- `push, pop, pushf, popf, call, ret`
- `je, jne, jcc, jcs, jl, jle, jg, jge, jb, jbe, ja, jae, jmp`
- `int, iret, hlt, cli, sti`
- `test, cmp`

Heute zusätzlich viele, viele MMX-, SSE-, ... Befehle.



Die 80x86-CPU's kennen folgende 32-Bit-Register:

`eflags`: Zero-, Overflow-, Carry-, ... -Flags

`eip`: Instruction Pointer

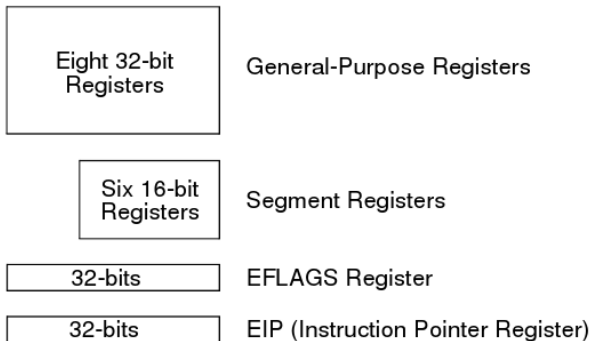
`eax, ebx, ecx, edx, edi, esi, ebp`: General Purpose Register

`esp`: Stack Pointer

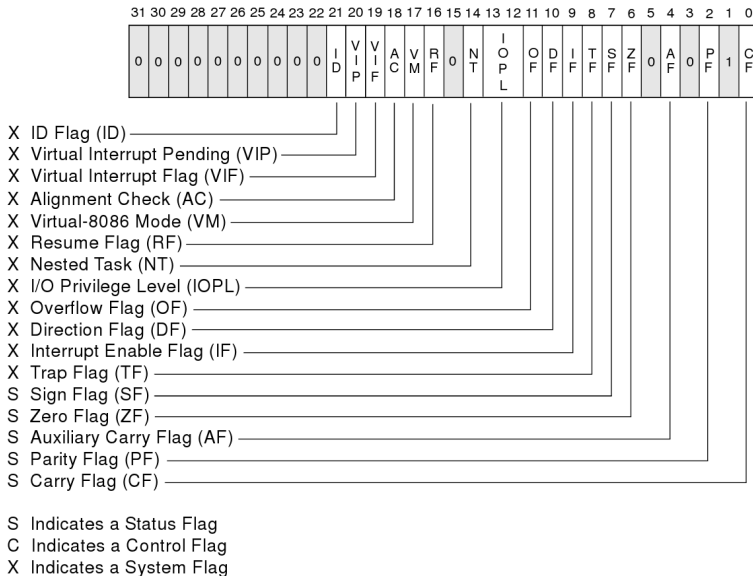
Zusätzlich: Segment-, FPU-, Control-, Debug- und Machine-Specific-Register



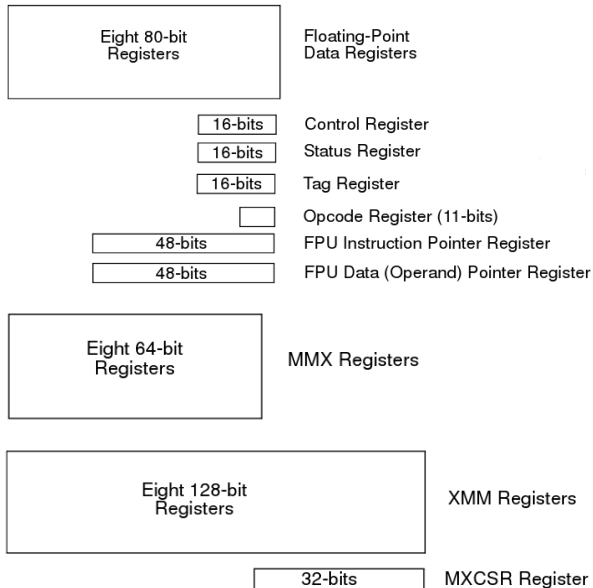
General Purpose / Segment Register

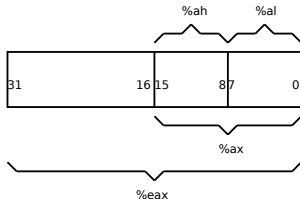


Flags Register

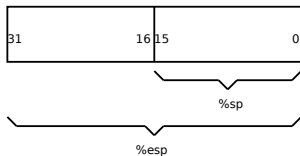


Floating Point Coprocessor / MMX/SSE/SSE2





Entsprechendes gilt für %ebx, %ecx und %edx.



Entsprechendes gilt für %edi, %esi und %ebp.

=> Es existieren je 8 Register für 8-Bit-, 16-Bit- und 32-Bit-Werte.



Beispiel (konvertiert 32-Bit-Little-Endian-Zahl in %eax in entsprechende Big-Endian-Zahl):

```
xchgb %al, %ah  
roll $16, %eax  
xchgb %al, %ah
```

Ab 80486:

```
bswap %eax
```



Die 80x86-CPU's kennen folgende Adressierungsarten:

- Immediate Operand (z.B. \$15)
- Register (z.B. %eax)
- Register indirekt (z.B. (%eax))
- Register indirekt mit Displacement (z.B. 8(%eax))
- Speicher (z.B. tmp)

Im Weiteren nicht verwendet:

- Register indirekt mit Index und Scale-Faktor (z.B. (%eax, %ebx, 8))
- Register indirekt mit Index und Scale-Faktor und Displacement (z.B. 10(%eax, %ebx, 8))



Programmierung – Beispiele

```
int x;    /* mem */
int sum;  /* %eax */
int i;    /* %ebx */

sum = 0;
for (i = 1; i < x; i++) {
    sum += i;
}
```

```
xorl %eax, %eax
movl $1, %ebx
jmp test
loop:
    addl %ebx, %eax
    incl %ebx
test:
    cmpl %ebx, x
    jge loop
```



```
main: .globl main
      movl $42, %eax
      addl %eax, %eax
      cmpl $84, %eax
      pushfl
      popl %eax
      hlt
```

Läßt man das Programm unter gdb laufen, ergibt sich:

```
(gdb) info register
eax                0x246    582
...
eflags             0x246    [ PF ZF IF RF ]
...
```

