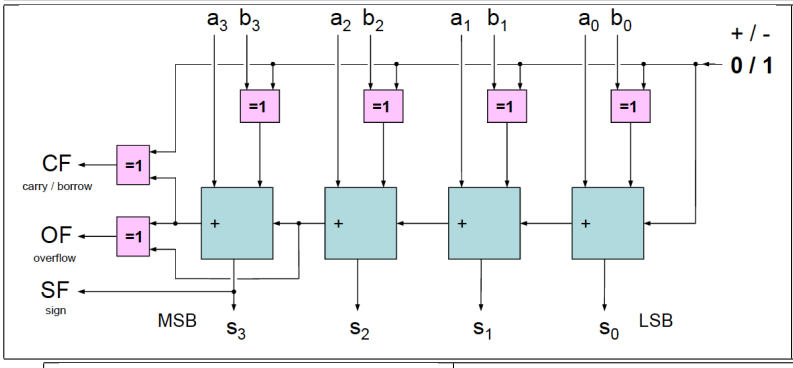
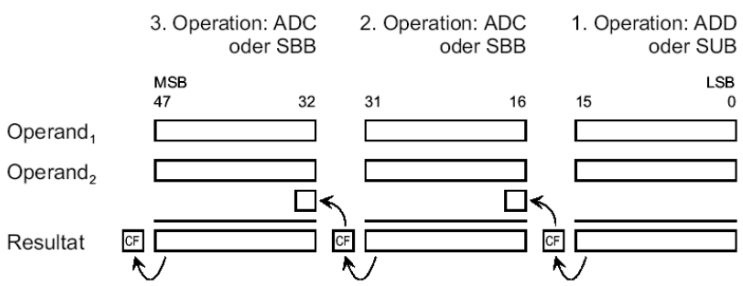
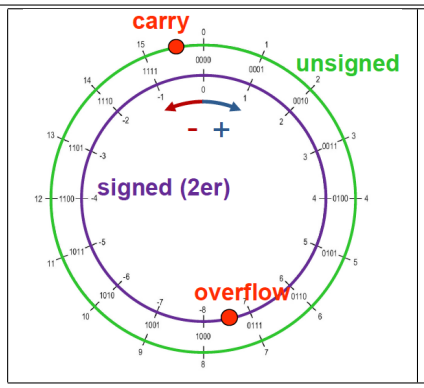
Addition und Subtraktion

2-Complement und Add/Sub in Hardware	Multibyte Addition/Subtraktion



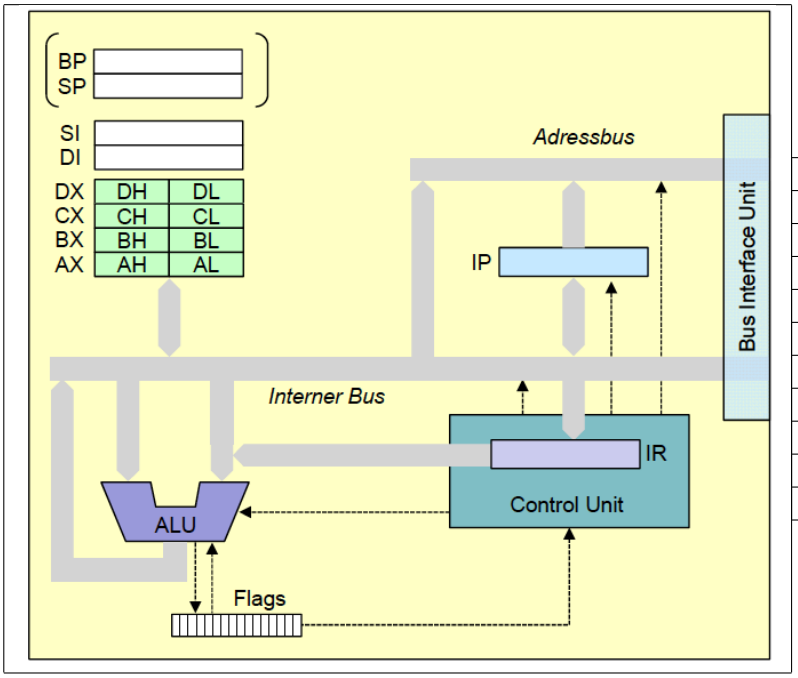




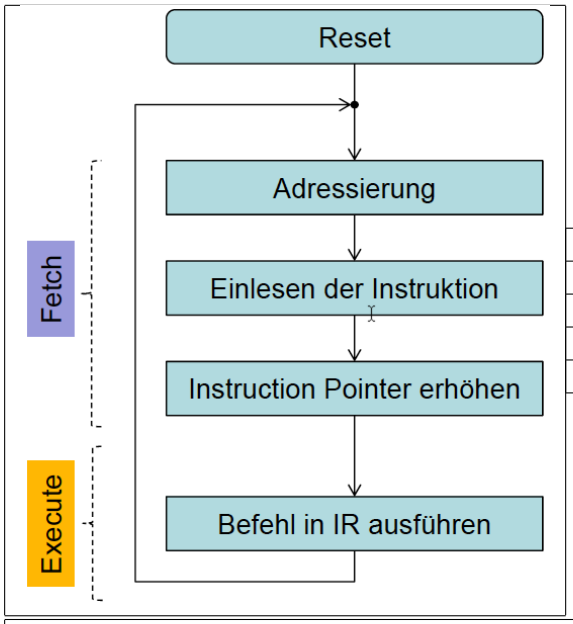

```
ADD   mem, reg
ADD   reg, mem
NOT   mem
NOT   reg
```

~~ADD mem, mem~~

Architektur



Instruction Pointer (IP)	
	Auf anderen Prozessoren oft Program Counter genannt
	16-Bit Register: 0x0000 bis 0xFFFF, 64 KByte adressierbar
	Nach RESET, IP = 0x0000
	Zeigt auf Adresse der nächste Instruction Fetch stattfindet
Instruction Register (IR)	
	Register, das den Opcode enthält, der im Moment ausgef. wird
	Teil der Control Unit, die den Instruktionsablauf steuert
Bus Interface	
	Schnittstelle zum externen System-Bus
	Umsetzung des internen Busses auf den externen Bus



Reset:	Grundzustand herstellen: Instruction Pointer auf 0000h setzen
Addr:	Adresse in Instruction Pointer (IP) über den Adressbus auswählen
Einl:	Inhalt der ausgewählten Speicheradresse in Instr.reg. (IR) kopieren
IP:	IP erhöhen (IP = IP + 2)
Exe:	Der im Instruktionsregister IR stehende Befehl wird ausgeführt

Daten

```
Tabelle  DW       4 DUP (?)                    ; Array Definition
 - - -
MOV               BX, Index                    ; get Index
SHL               BX, 1                        ; Groesse: 2 Byte
                                               ; shift left = * 2
MOV               AX, [BX + OFFSET Tabelle]    ; Daten lesen 1)
MOV               Res, AX                      ; Resultat speichern
```

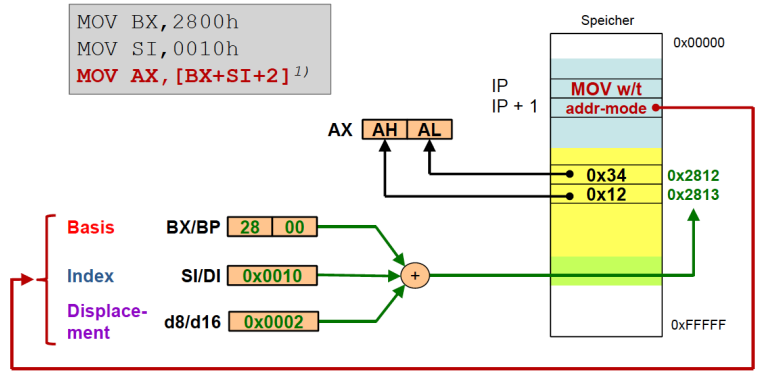
```
Beispiele:
MOV    AX,Zahl        ;direkt
MOV    DX,[BX]        ;indirekt mit basis
MOV    AL,[BX+4]      ;indirekt mit basis und displ.
MOV    AX,[BP-6]      ;indirekt mit basis und displ.
MOV    CX,[BX+SI]     ;indirekt mit basis und index
MOV    CH,[BX+DI-2]   ;indirekt mit basis, index und displ.
```

Adresse 16 Bit

```
.CODE
;
;    index = 17;
;
1 mov  word ptr _index,17
;
;    charArray[index] = 0;
;
2 mov  bx, word ptr _index
3 mov  byte ptr _charArray[bx],0

.DATA
_charArray    db     20 dup (?)
_index        dw     ?
```

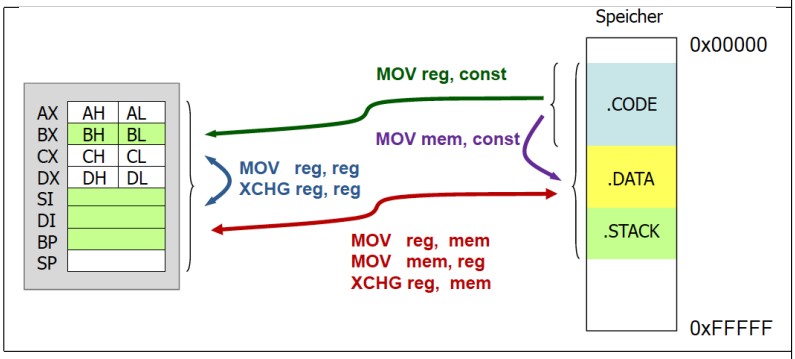
```
MOV BX,2800h
MOV SI,0010h
MOV AX,[BX+SI+2] 1)
```

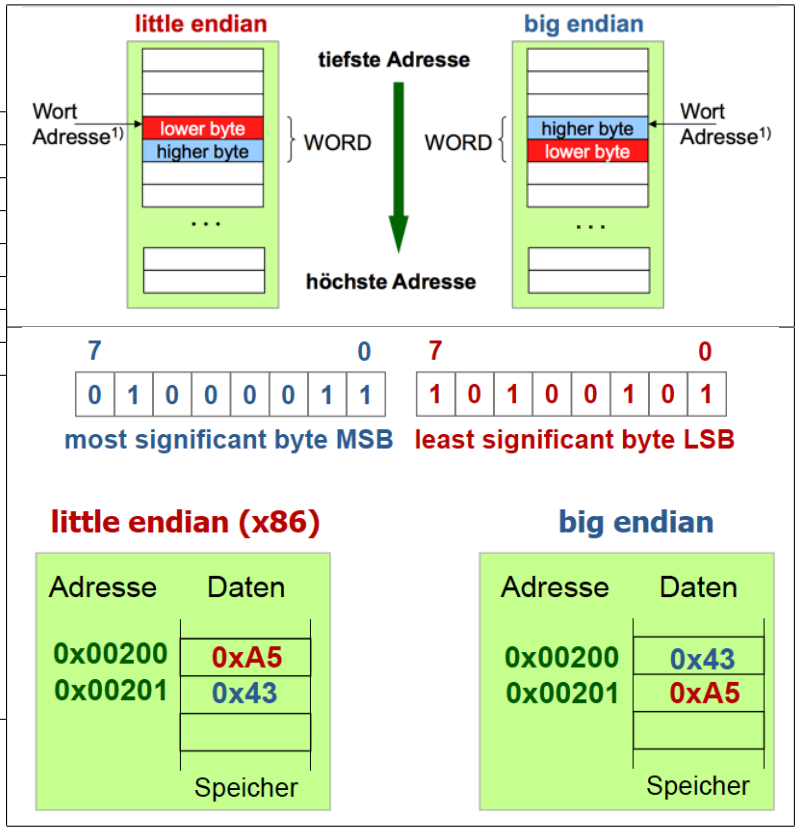


Adressierung

Register	AX
Immediate	1234h
Direkt	DS:[1234h]
Register-Indirekt	[BX] oder [BX+SI+12h] oder 12h[BX+SI]
Implizit	

I/O	nur über AX und DX

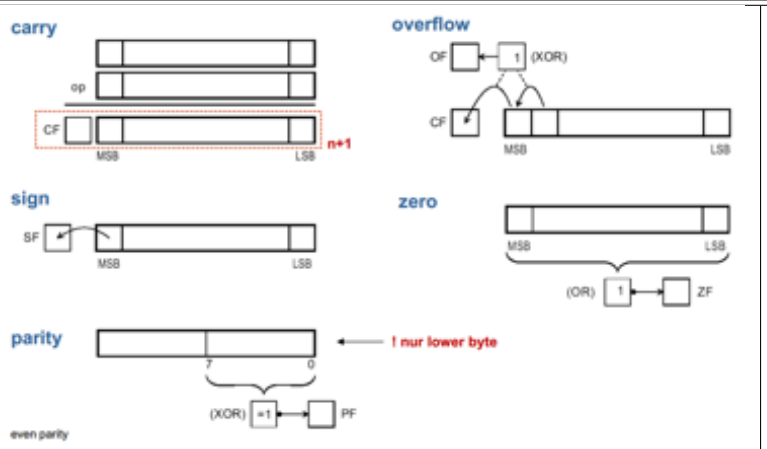
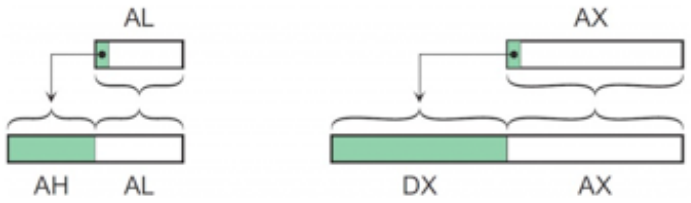




Befehle

Unäre Instruktionen	Binäre Instruktionen
INC, DEC, NEG(2comp), NOT(bitweise), CBW, CWD	ADD, ADC, SUB, SBB, MUL, IMUL, DIV, IDIV

Flags & Signextention

Flags	Sign-Extention
	
	CBW (Byte to Word), CWD (Word to Doubleword)

Jump

JMP reg → absoluter, Register indirekter Sprung

```
Line  Address  Code          Assembler Code
20    0011     BB 0018r      MOV  BX, OFFSET jmpaddr
21    0014     FF E3         JMP  BX
22    0016     90            NOP
23    0017     90            NOP
24    0018     E4 04         jmpaddr: IN   AL,[0004h]
25    001A     90            NOP
```

jmpaddr → BX
BX → IP

JMP Speicheroperand → absoluter Speicher-indirekter Sprung

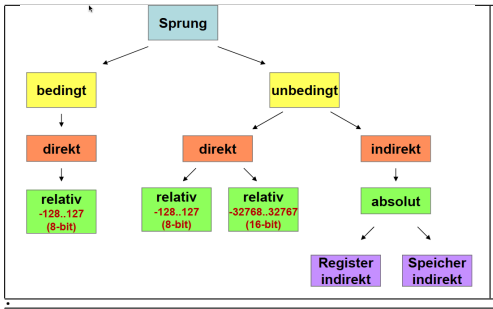
```
Line  Address  Code          Assembler Code
28    001C     0029r         casetable DW OFFSET case0
29    001E     002Dr                   DW OFFSET case1
30    0020     0031r                   DW OFFSET case2
31
32                           ;BX contains index 0,1 or 2
33    0022     D1 E3                   SHL  BX,1 ;index * 2
34    0024     2E: FF A7001Cr          JMP  casetable[BX]
35    0029     B0 00         case0:    MOV  AL,0
36    002B     EB 06                   JMP  endcase
37    002D     B0 64         case1:    MOV  AL,100
38    002F     EB 02                   JMP  endcase
39    0031     B0 C8         case2:    MOV  AL,200
40    0033     EB FE         endcase:
```

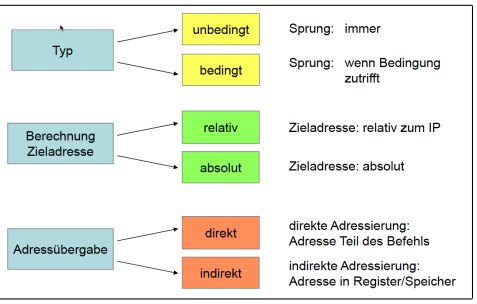
JMP label → direkter, relativer Sprung

```
Line  Address  Code          Assembler Code
13    0005     E4 04         back:     IN   AL,[0004h]
14    0007     E6 06                   OUT  [0006h],AL
15    0009     EB 02                   JMP  forward
16    000B     24 56                   AND  AL,56h
17    000D     E6 08         forward:  OUT  [0008h],AL
18    000F     EB F4                   JMP  back
19    0011     ....
```

- Forward IP = 000Bh + 0002h = 000Dh
- Back IP = 0011h + FFF4h = 0005h

IP displacement

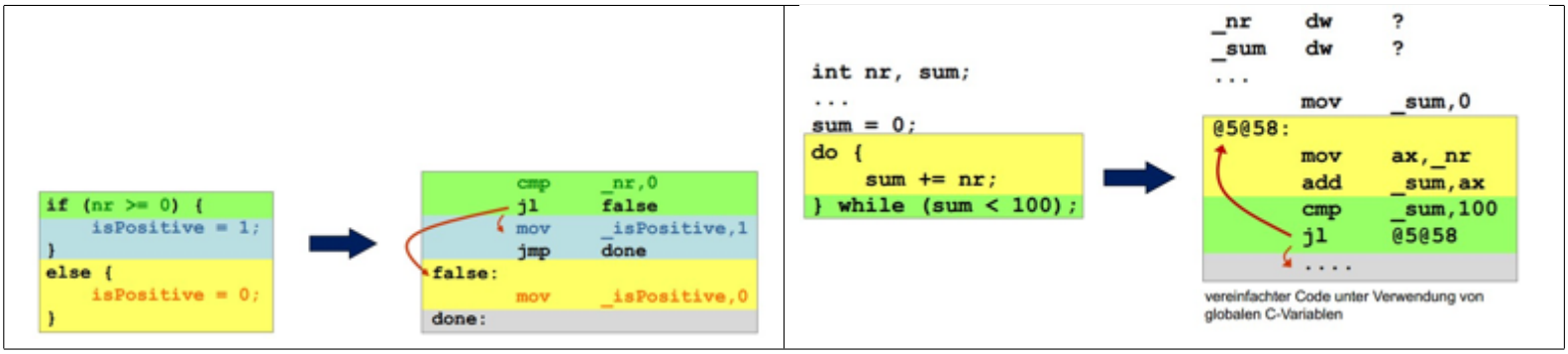




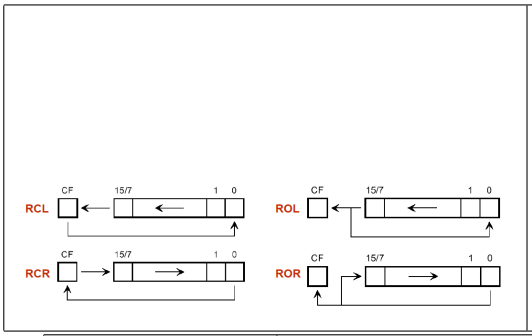
Sprungbefehle:	
unbedingt, relativ, direkt	JMP IP = [IP -128; IP + 128] oder IP = [IP – 32'768; IP + 32'767]
unbedingt, absolut, indirekt	JMP IP = [0; 65'535] → reg oder mem
bedingt, relativ, direkt	Jxx IP = [IP -128; IP + 128]

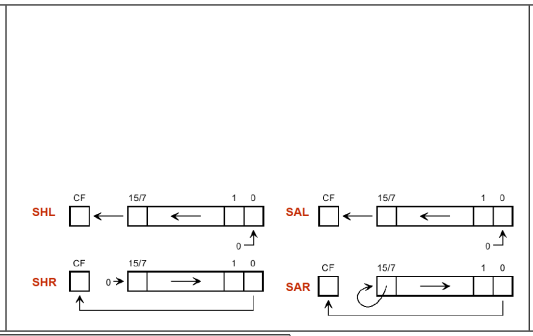
Vergleichsinstruktionen:		Signed	Unsigned	Flagabhängig
CMP	SUB ohne Resultat, aber Flags werden gesetzt	greater	above	J<flag1Letter>
TEST	AND ohne Resultat, aber Flags werden gesetzt	less	below	JN<f1L>

Kontrollstrukturen



Shift Rotate





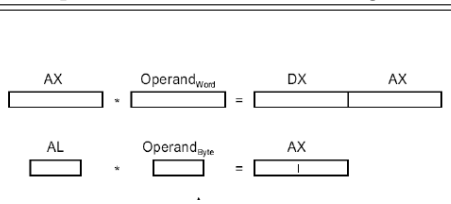
Mnemonic	Instruktion (Befehl)		8 Bit	16 Bit	
SHL[1]	logischer Shift: links	[2] $2^n \cdot A$	✓	✓	[3] 0 → LSB
SHR[1]	logischer Shift: rechts	[2] $2^n \cdot A$	✓	✓	[3] 0 → MSB
SAL[1]	arithmetischer Shift: links	[2] $2^n \cdot \pm A$	✓	✓	[3] 0 → LSB
SAR[1]	arithmetischer Shift: rechts	[2] $2^n \cdot \pm A$	✓	✓	[3] MSB → MSB
ROL[1]	rotate left		✓	✓	[3] MSB → LSB
ROR[1]	rotate right		✓	✓	[3] LSB → MSB
RCL[1]	rotate left through carry		✓	✓	[3] MSB → c → LSB
RCR[1]	rotate right through carry		✓	✓	[3] LSB → c → MSB
CLC	clear carry	0 → c			
STC	set carry	1 → c			
CMC	negate carry	!c → c			

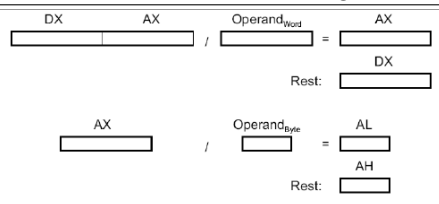
[1] Als Funktion unäre Instruktion: 1 Operand
[2] Parameter n: $1 \le n \le 255$
[3] Funktion für LSB, MSB und Carry

Bits auf 0 setzen:	AND AL, 1111 1110b (letztes Bit 0 setzen)
Bits auf 1 setzen:	OR AL, 0000 0001 (letzes Bit 1 setzen)
Invertieren:	XOR AX, AX

Multiplikation und Division

Multiplikation mit mehreren Registern	Division mit mehreren Registern	MUL bei nicht Signed - IMUL bei Signed






```
   5 *   3          x86: MUL          x86: IMUL
0101 * 0011      5 *   -3          5 *   -3
     0011   0101 * 1101      0101 * 1101
     0000        1101        11111101
   0011          0000        00000000
 0000          1101        11111101
00001111       0000        00000000
               01000001    100111110001
```

Achtung:	Bei x86 nur 32bit/16bit oder 16bit/8bit erlaubt!!!!!

Multiplikation mit Konstanten

Bsp: AX = 13 • DX → AX = (1 + 4 + 8) • DX		Bsp: AL/15 (Durch Multiplikation dargestellt: 256/divisor = multiplikator)	
MOV AX, DX	; AX = DX	XOR AH, AH	; clear AH
SAL DX, 2	; 4 • DX	MOV BX, AX	; save AX
ADD AX, DX	; AX = AX + 4 • DX	SHL AX, 4	; mul by 16
SAL DX, 1	; 2 • DX -> 8 • DX	ADD AX, BX	; add once
ADD AX, DX	; AX = AX + 8 • DX	MOV AL, AH	; result to AL