x86-Assemblerprogrammierung

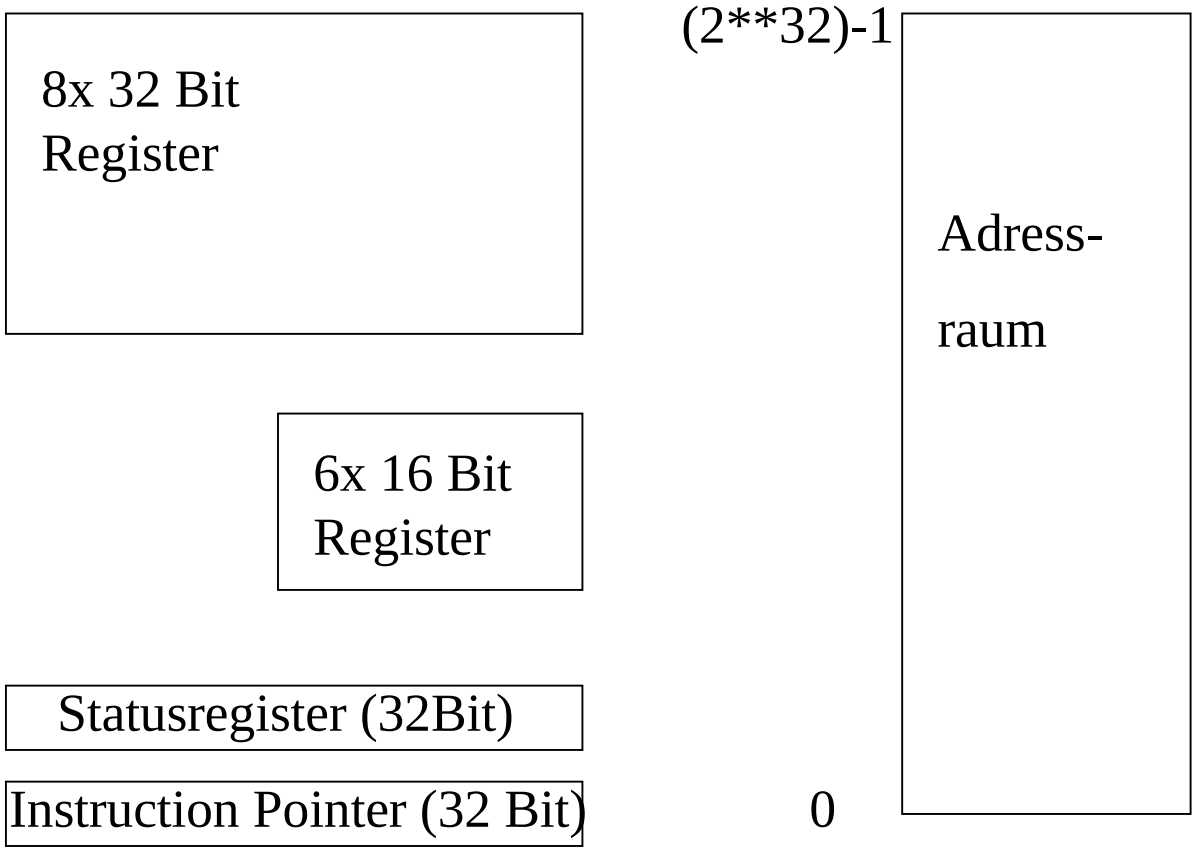
Inhalt

- Register
- Speicherverwaltung
- Adressierung
- Stackverwaltung
- Datentypen und Befehlssatz
- Befehlskodierung

Literatur

- Dandamudi :“Intruduction to Assembly Language Programming“
- „Intel Architecture Developer Manual“ Band 1 bis 3
- Randall Hyde : „The Art Of Assembly Language Programming“

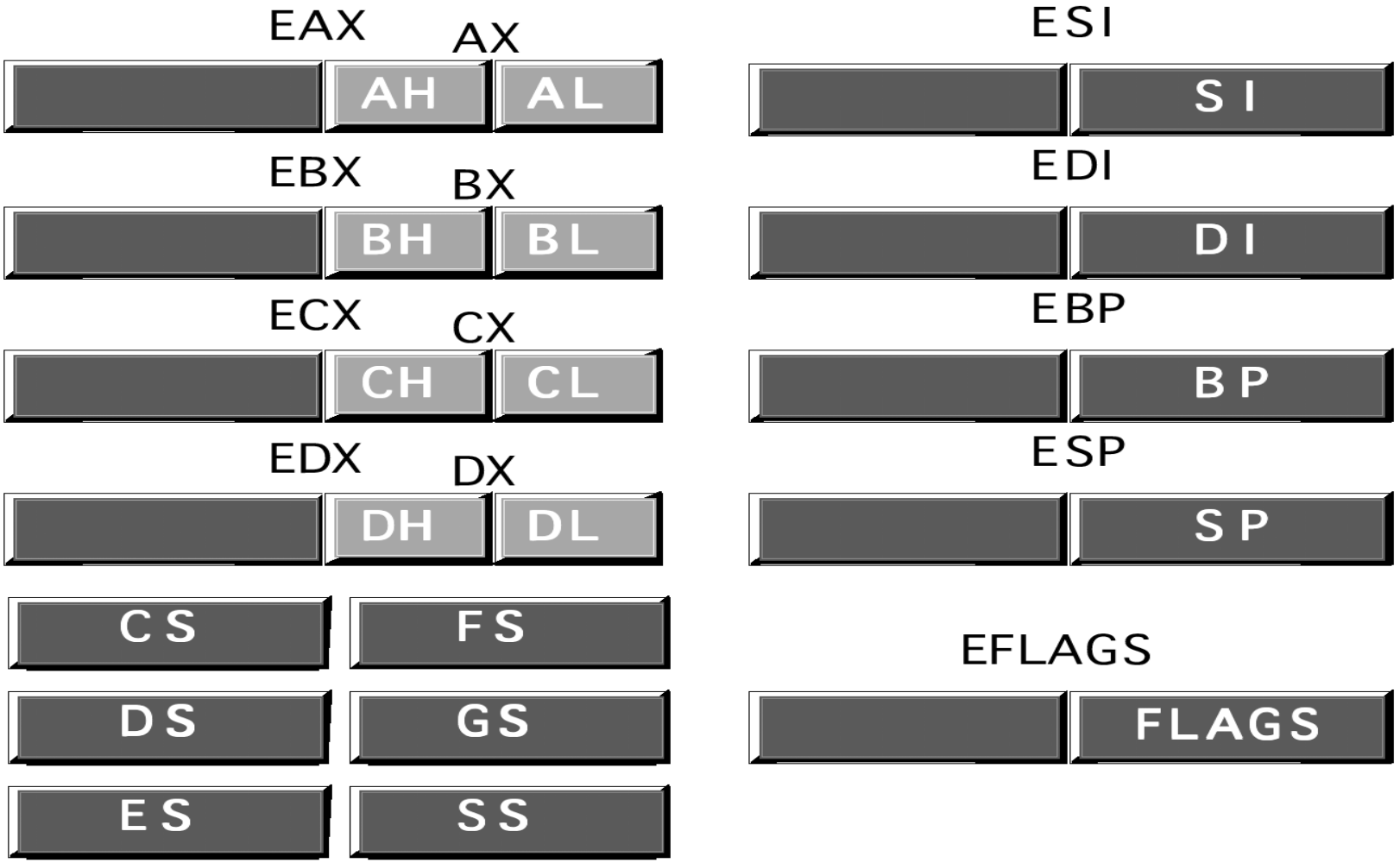
80386+ Programm-Umgebung



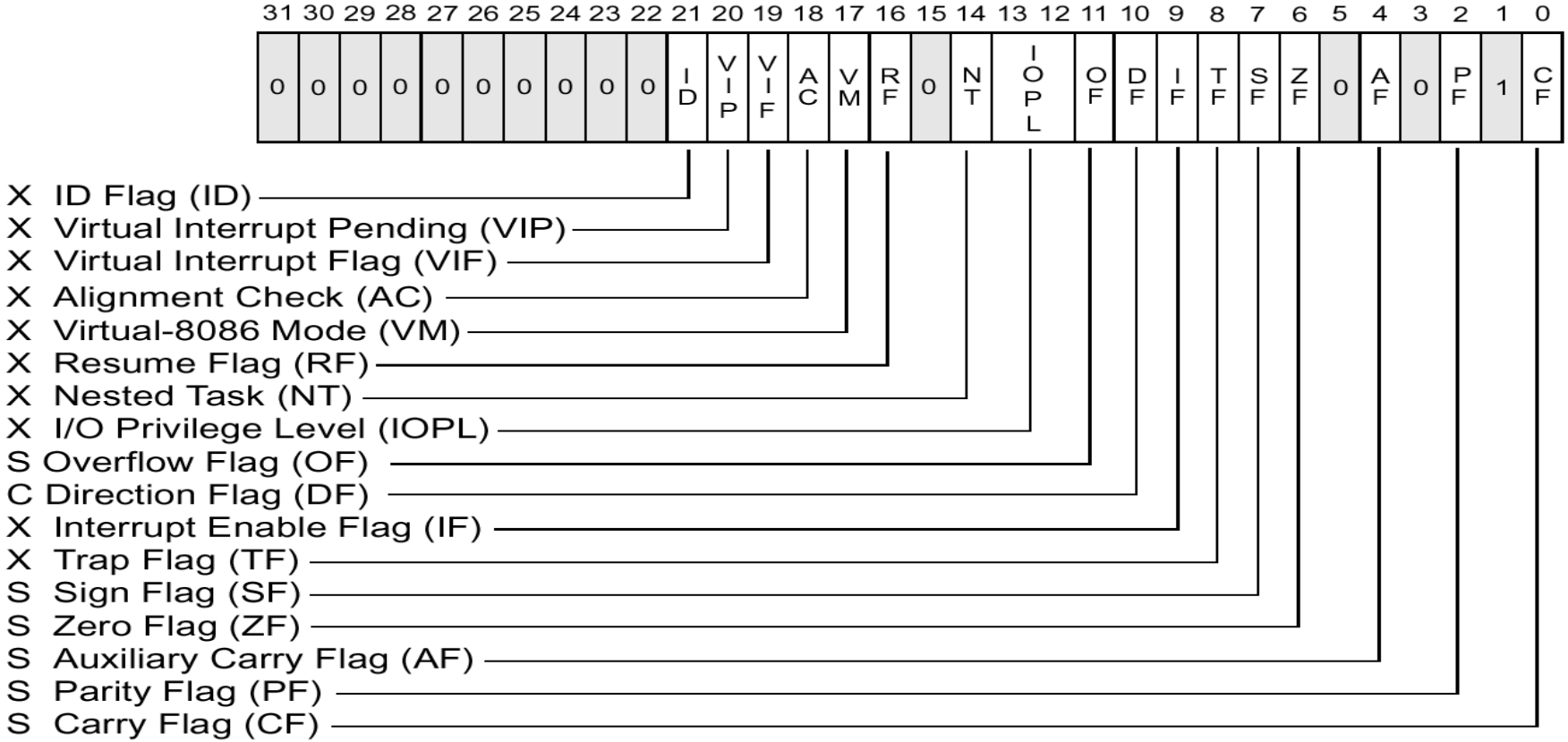
80386+ Register

Register	**Bezeichnung**	**Verwendung**
(E)AX	Accumulator	Allzweckregister
(E)BX	Base Register	Allzweckregister
(E)CX	Count Register	Allzweckregister (Zähler:Schleifen,Strings)
(E)DX	Data Register	Allzweckregister
(E)SI	Source Index	Index-Register (Strings,Arrays)
(E)DI	Destination Index	Index-Register (Strings,Arrays)
(E)SP	Stack Pointer	Zeiger-Register (Stack)
(E)BP	Base Pointer	Zeiger-Register (Stack)
(E)IP	Instruction Pointer	Befehlszeiger
CS	Code Segment	Segment-Register
DS	Data Segment	Segment-Register
ES	Extra Segment	Segment-Register
SS	Stack Segment	Segment-Register
FS,GS	-	Segment-Register (Daten-Segment)

80386+ Register



Das Statusregister



S Indicates a Status Flag
C Indicates a Control Flag
X Indicates a System Flag

Bedingter Befehl: z.B. cmp count, 100

je Label1

Speicherorganisation

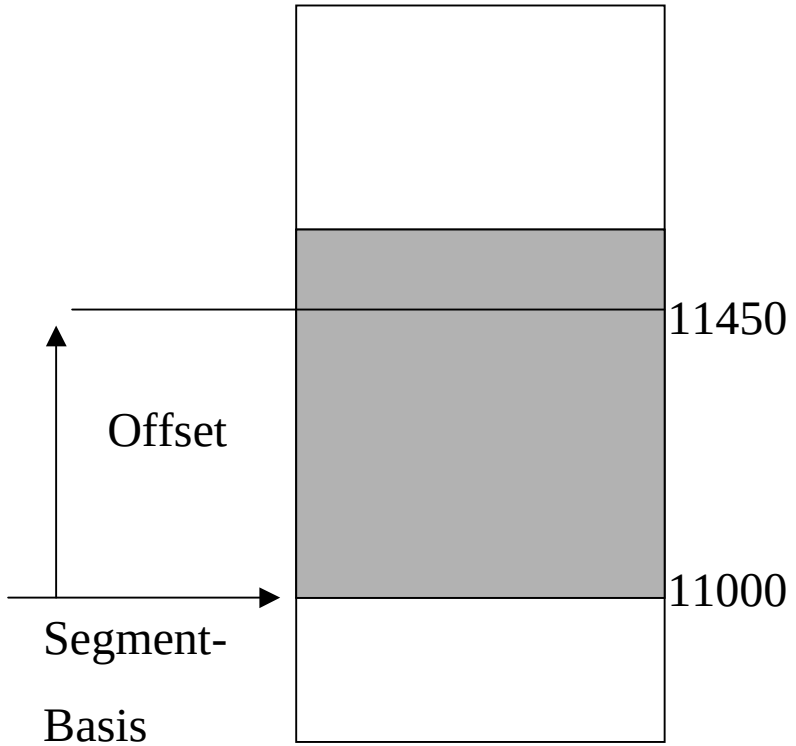
- Physikalischer Speicher byteweise adressierbar
- Höherwertiges Byte an höherer Adresse (Little-Endian)

Operationsmodi

- Real-Address-Mode: Im wesentlichen 8086-Prozessor, Real-Address-Memory-Model
- Protected Mode: Neue Befehle, erweiterte Speichermodelle, Paging, Abwärtskompatibilität durch virtuellen 8086-Modus
- System Management Mode (SMM): für Betriebssytem (Powermanagement, Sicherheitsfunktionen)

Speicherorganisation im Real-Mode

- 20 Bit Areßbus, 1MB Speicher
  16 Bit Register
- Speicher in Segmente unterteilt
- Logische Adresse:
  Segment : Offset
- Segmentregister enthalten die 16 höchstwertigen Bits der Segment-Basis-Adresse, Anhang von 4 0-Bits bei Adreßberechnung
- Offset:16 Bit, 64KB Segmente



Adress-Berechnung,Segmentation

Adress-Berechnung:

Prozessor: logische Adresse - lineare Adresse (20 Bit)

Bsp.: log. Adresse 1100:450H

lineare Adresse: 11000+00450=11450 (20 Bit)

Segmentation:

- bis zu 6 Segmente gleichzeitig aktuell (Code,Daten,Stack)
- Segmentüberlappung möglich, lineare Adresse kann durch viele logische Adresse dargestellt werden
- Vorteile: 1) 20 Bit Bus kann genutzt werden
  2) höhere Zuverlässigkeit

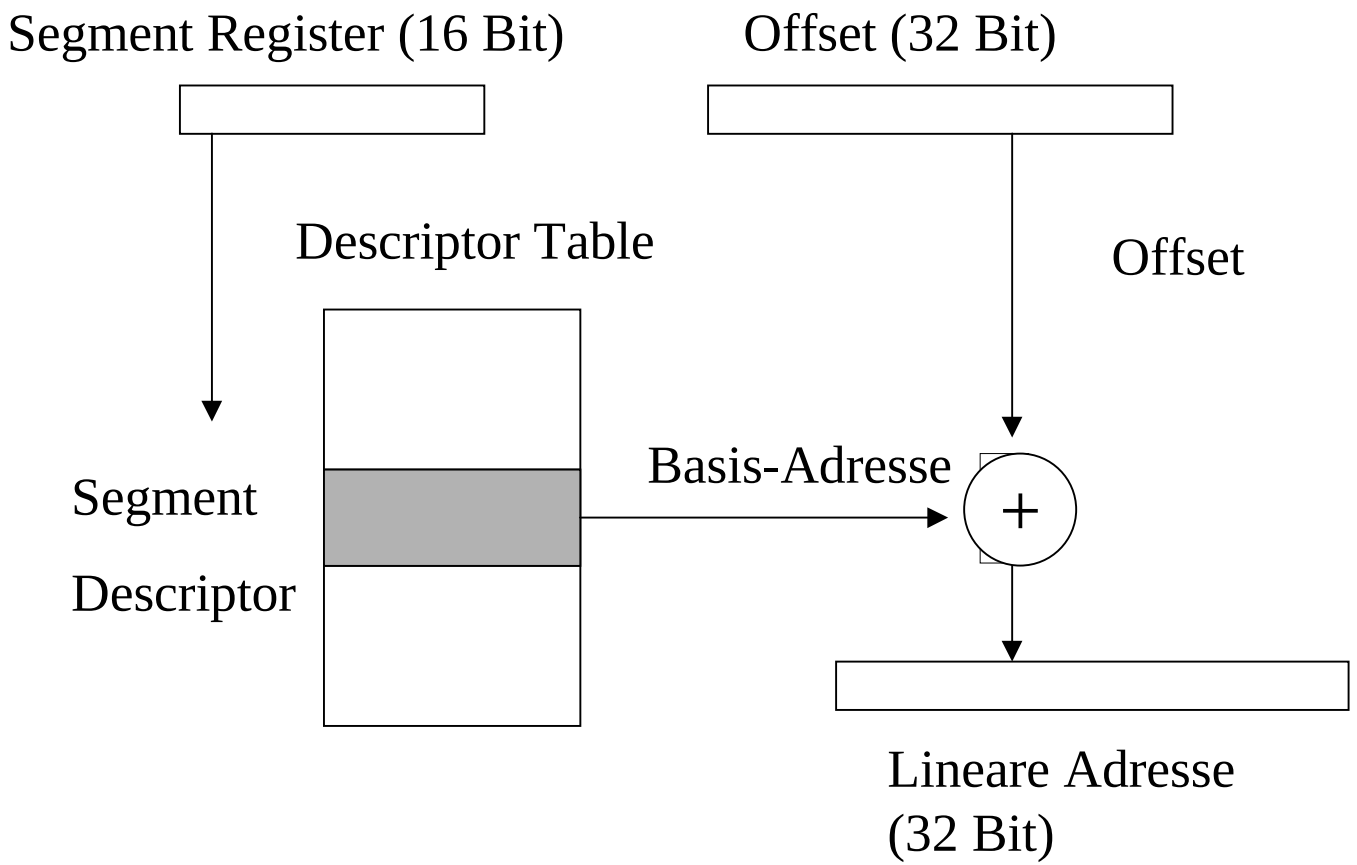
Speicherorganisation im Protected Mode

- 32 Bit Register, 32 Bit Adress-Bus (4GB Speicher)
- ebenfalls Segmentation des Speichers
- Offset :32 Bit , 4GB Segmente
- Segmentregister enthalten keine Zeiger auf die Segment-Basis, sondern verweisen auf „Segment Desciptor Tables“

Segment Desciptor Table:

- Array von (8 Byte)-Segment-Descriptors
- Segment-Descriptor enthält Informationen über das Segment
- 2 Typen: Global Desciptor Table (GDT) für Betriebssystem
  Local Descriptor Table (LDT) für Programme
- Zeiger auf Basis der Arrays im GDTR bzw. LDTR (Register)

Adressberechnung im Protected Mode



Segment Selector, Segment Descriptor

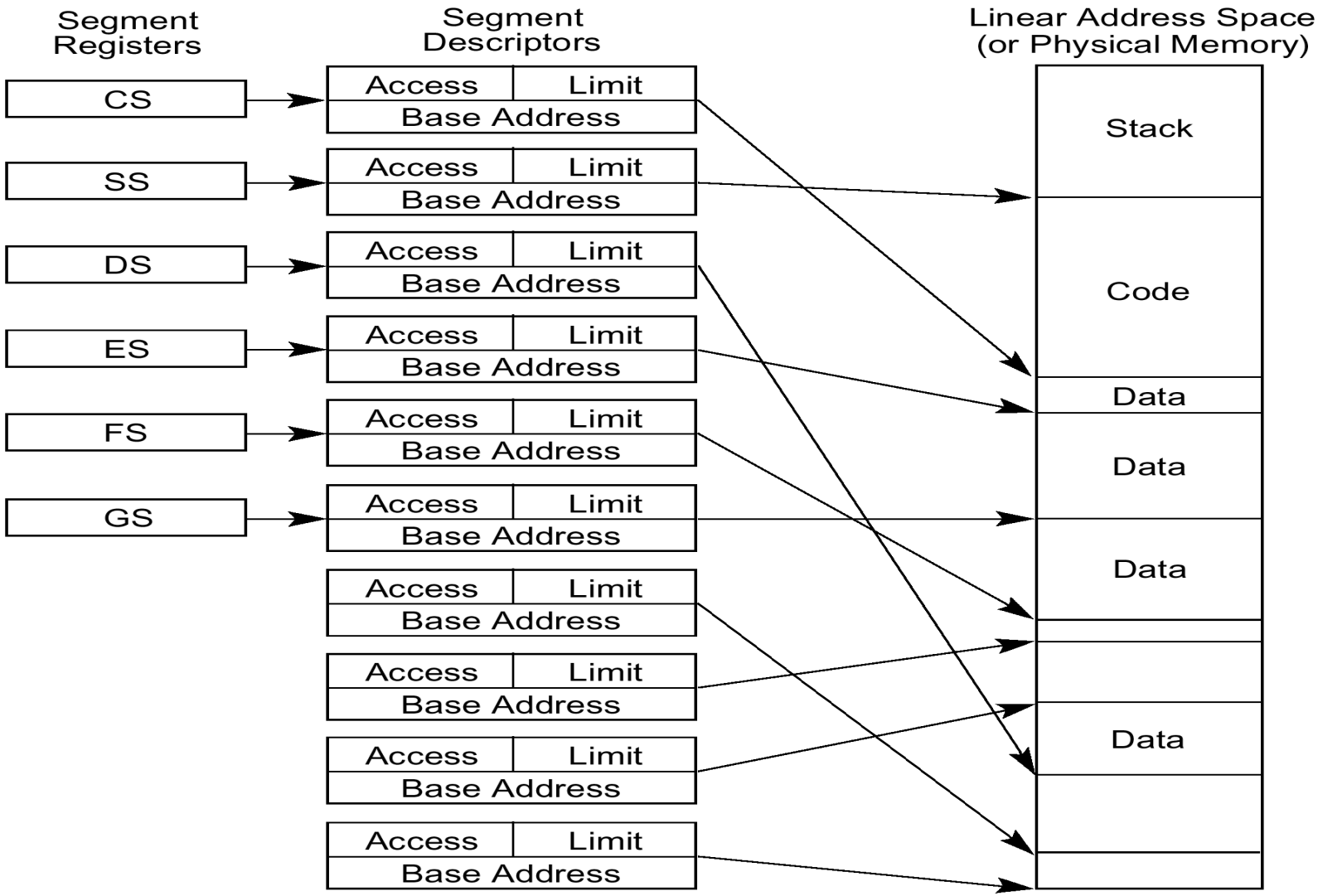
Segment Selector

- TI: Table-Indicator (GDT/LDT)
- INDEX: 13 Bit, 8192 Descriptors
  Adressberechnung: 8*INDEX+ Basis-Adresse (GDTR/LDTR)
- maximal 16384 Segmente

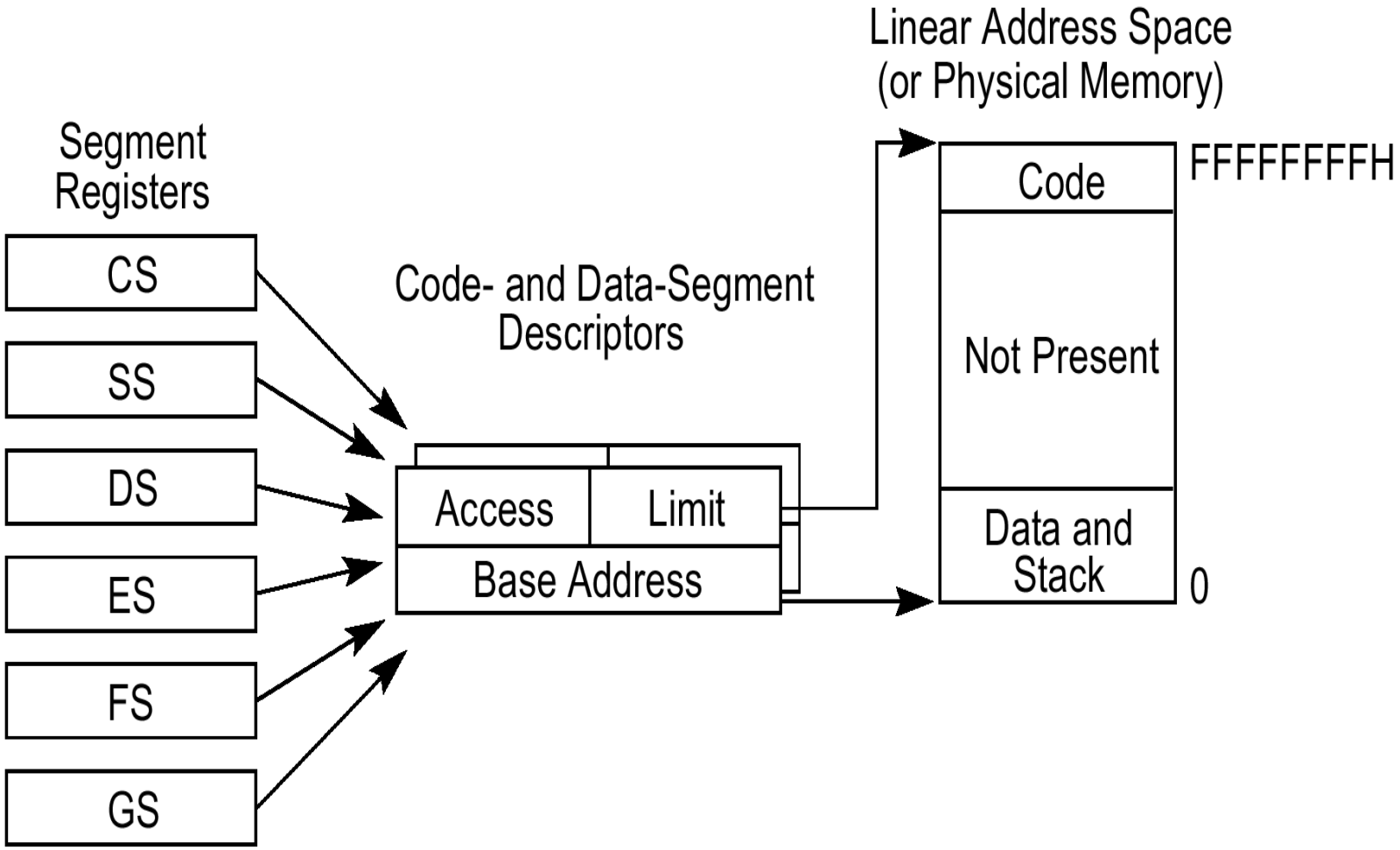
Descriptors

- Basisadresse (32 Bit)
- Segment-Limit (20 Bit)
- Granularität (1B/4KB)
- S-Bit: System-Segment ...

Segment-Register im Multisegment MemoryModel



Segment-Register im Flat Memory Model

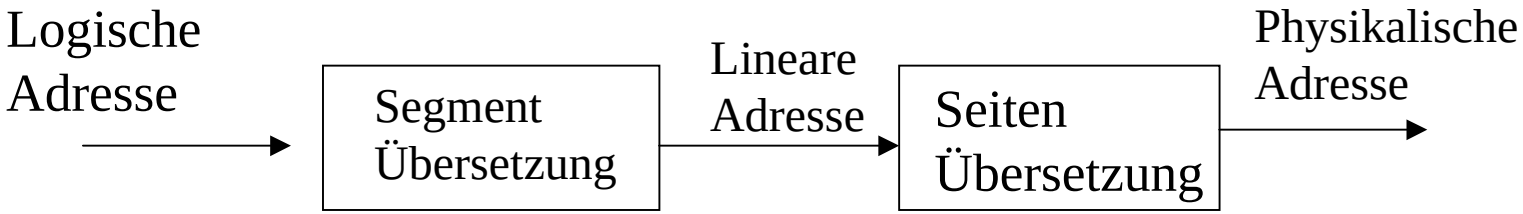


Paging

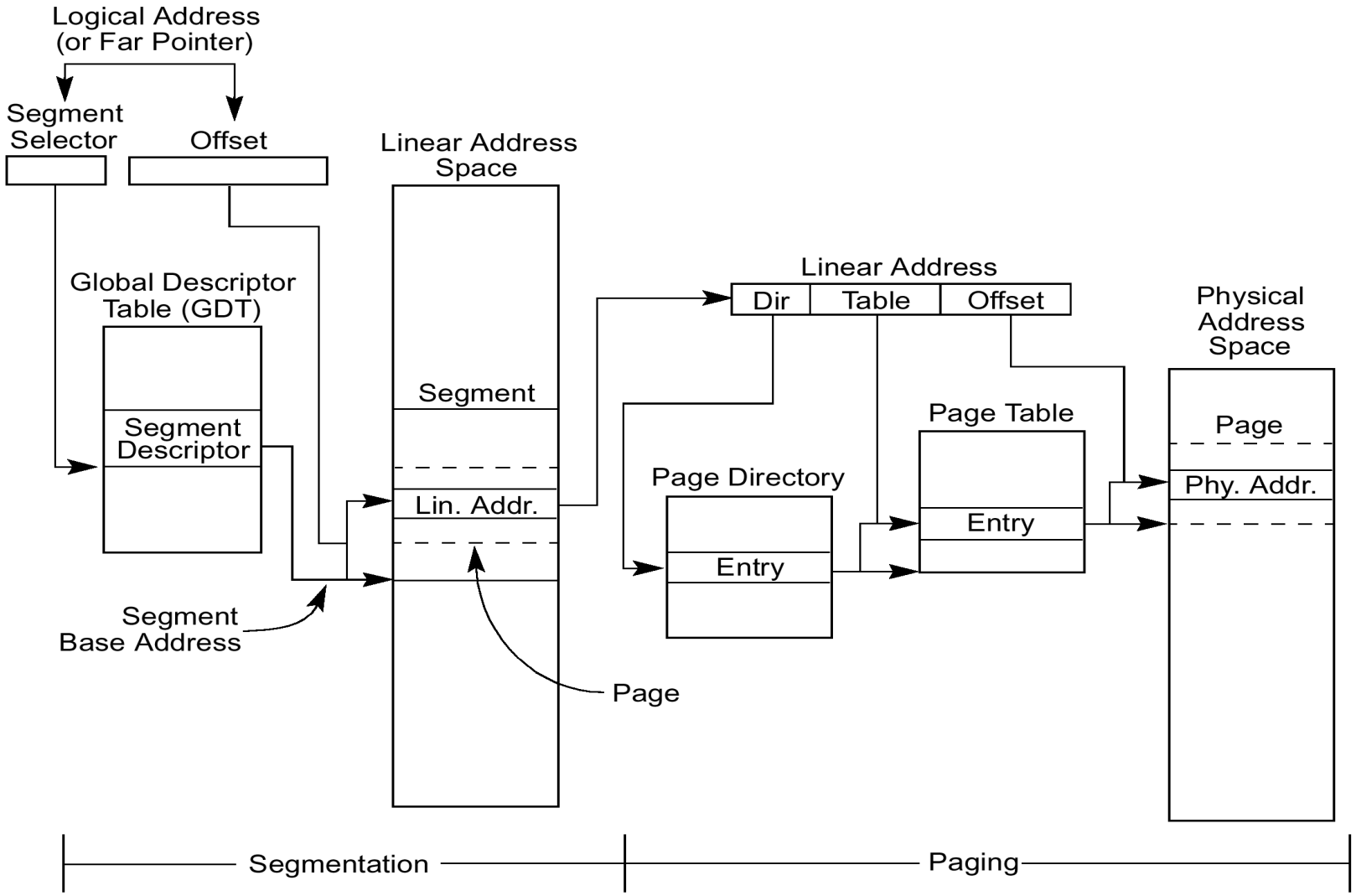
- Mechanismus, mit dem virtueller Speicher implementiert wird
  (Linearer Adressraum > physikalischer Adressraum)
- nur im Protected Mode verfügbar

Funktionsweise:

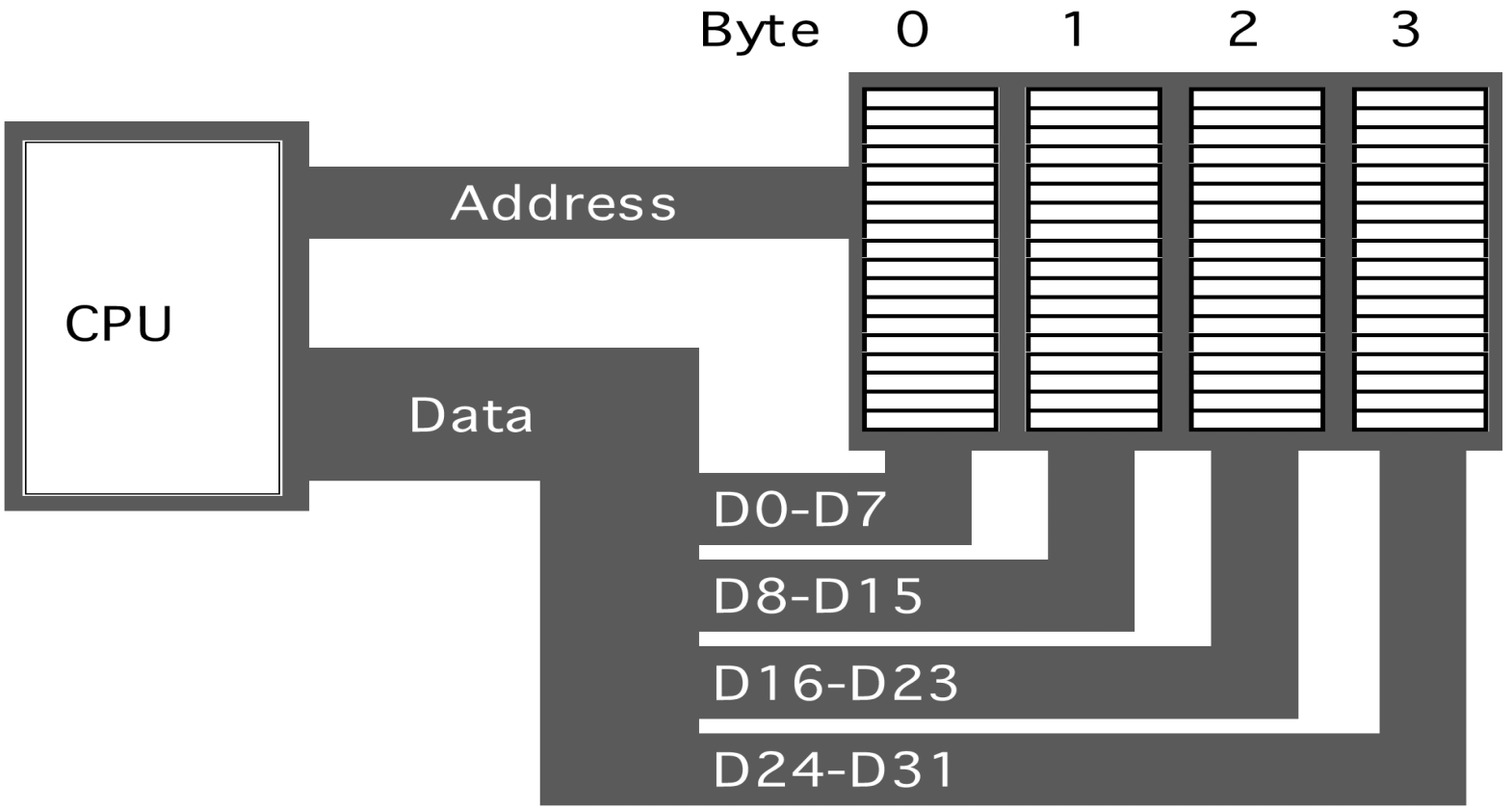
- Unterteilung der Segmente in 4KB-Bereiche (Seiten)
- Speicherung im Hauptspeicher und auf Festplatte
- Lineare Adresse wird über Seitenverzeichnis und Seitentabelle in physikalische Adresse übersetzt
- Seite nicht im Hauptspeicher: Unterbrechung + Laden



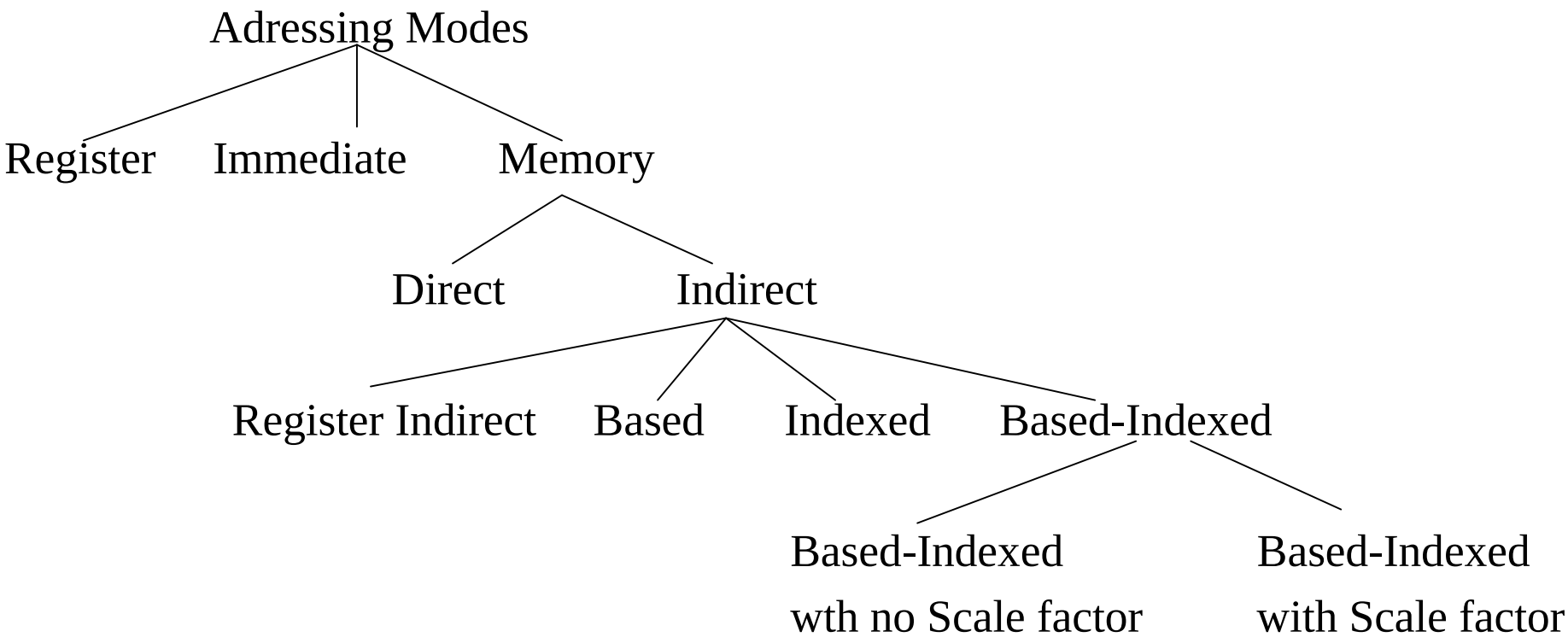
Segmentation und Paging



Datenausrichtung



Adressierung (32 Bit)



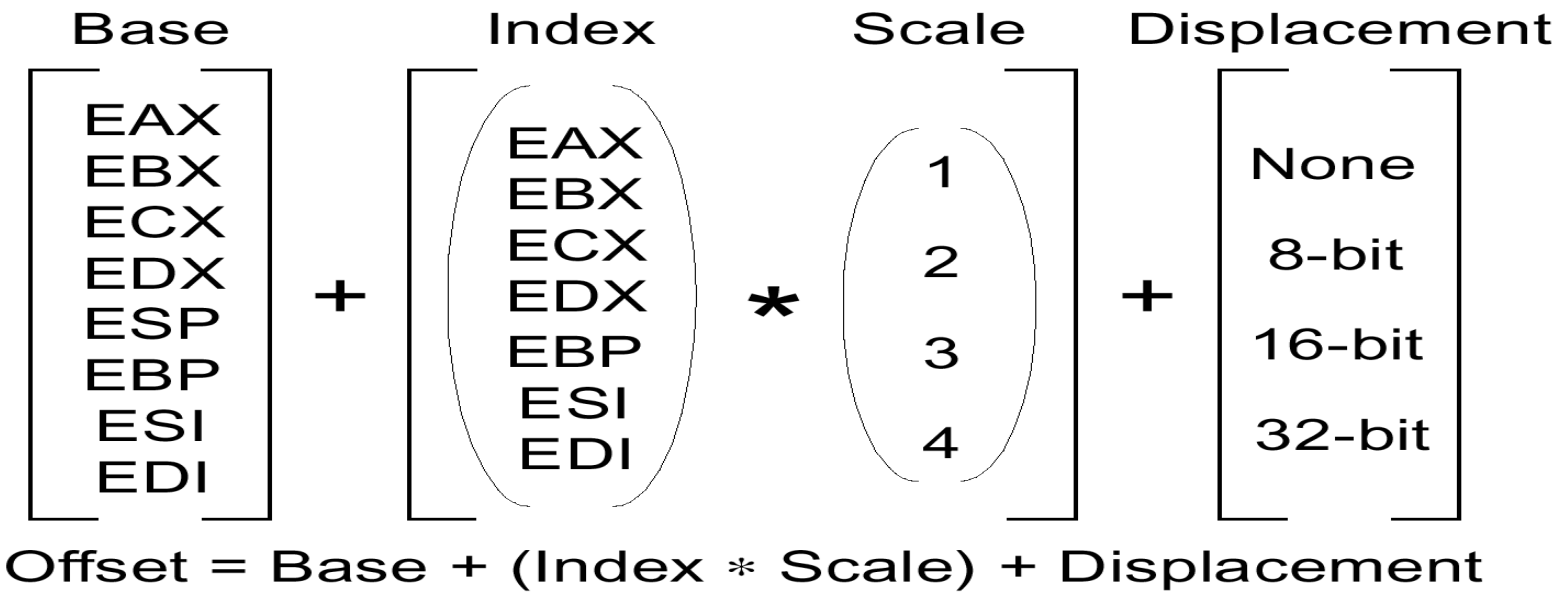
Adressierungsmodi

- Register Addressing Mode : Bsp.: mov EAX,EDX
  kurzer Befehlscode, kein Speicherzugriff für Operanden
- Immediate Addressing Mode: Bsp.: mov AL,56
  Operand im Befehlscode gespeichert (CS), maximal 32 Bit
- Direkte Speicheradressierung: z.B. mov EAX,Variable1

Indirekte Speicheradressierung

- Logische Adresse: Segment : Offset
- Segment implizit durch Befehl festgelegt: nur Offset nötig
  z. B. mov,add : Daten - Segment, push/pop: Stack-Segment...
- Segment Override: Zugriff auf anderes Segment als das Implizite
  z.B. add AX, SS:[BX]
- Grund für Vielzahl der Adressierungsarten: effiziente Implementierung der Datenstrukturen höherer Programmiersprachen

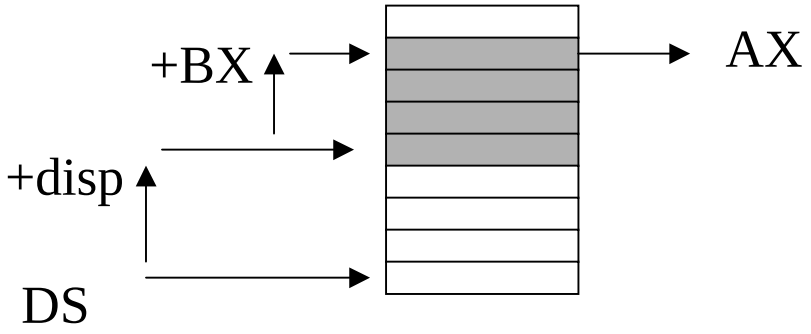
Indirekte Speicheradressierung (32 Bit)



- Registerinhalte und Displacement werden als Zweier-Komplement-Zahlen ausgewertet
- Basis-Register ESP,EBP: Stack-Segment, sonst Datensegment implizit
- Beispiel: mov AX, [BX+SI*2+18]
- 16 Bit-Adressierung: Basis nur BX,BP; Index nur SI,DI; kein Slalenfaktor; maximal 16 Bit Displacement

Verwendung der Adressierungsarten

- Base+Displacement: z. B. Arrays: Disp. Zeigt auf Anfangsadresse des Arrays, Basis-Register enthält Ergebnis der Index-Berechnung
- (Index*Scale)+Disp.: z. B. Arrays mit Elementgrösse von 2,4,8 Bytes: Index-Register enthält Index, Disp. Zeigt auf die Anfangsadresse, der Skalenfaktor gleicht der Elementgrösse
- Base+Index+Disp.: z.B. 2-dimensionale Arrays, array of records (2 unabhängige Parameter)
- Base+(index*Scale)+Disp: z.B. 2-dim. Arrays mit Elementgrösse 2,4,8 Bytes



Bsp.: mov AX,[BX+4]

Programmbeispiel : Arrays

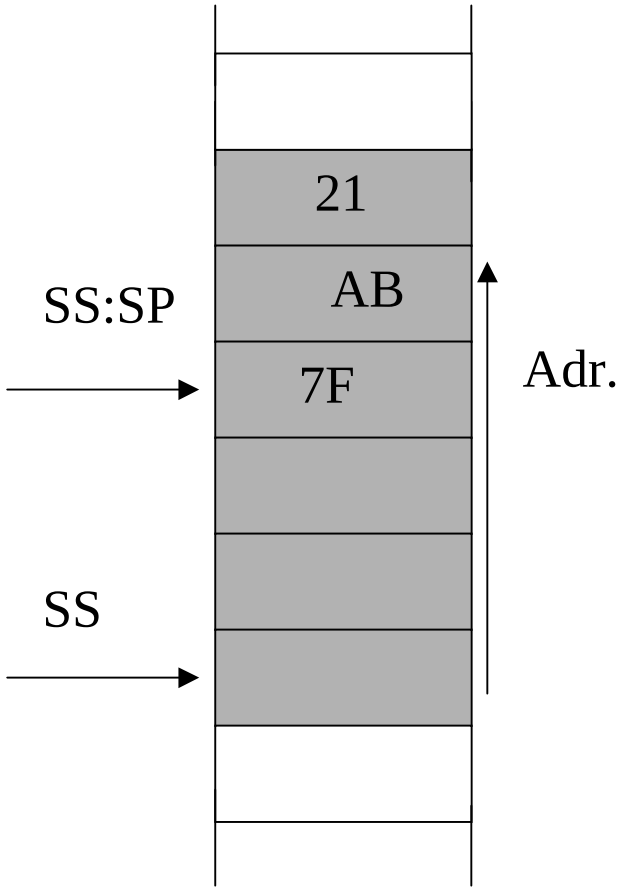
```
TITLE  Array-Summe
.MODEL SMALL
.STACK 100H
.DATA
zahlen  DD 90,50,70,94,81,40,67
anzahl  EQU ($-zahlen)/4

.CODE
.486
INCLUDE io.mac
```

```
main   PROC
       .STARTUP
       mov  CX,anzahl
       sub EAX,EAX
       sub ESI,ESI
schleife:
       add EAX,zahlen[ESI*4]
       inc ESI
       loop schleife
       .EXIT
main ENDP
        END  main
```

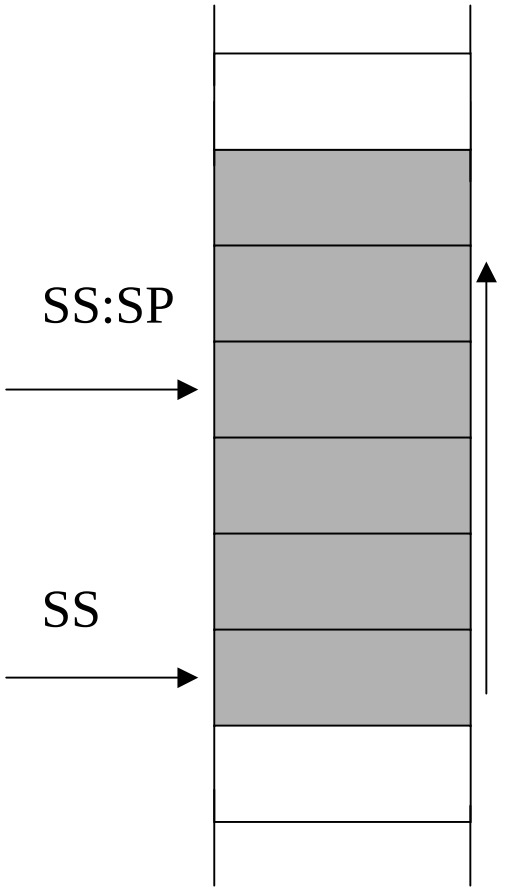
Der Stack

- LIFO-Datenstruktur, implementiert durch:
  - Stack-Segment (SS)
  - (E)SP:Stack-Pointer: „Top of the stack“
  - push source,pop destination
- Verwendung:
  - -temporärer Speicher, z.B Registersicherung, lokale Variable
  - -Speicherung des Befehlszeigers bei Prozeduraufrufen
  - -Parameterübergabe
- Datenbreite: 16 Bit oder 32 Bit (vgl.D-Bit)
- Stack wächst zu niedrigeren Adressen hin



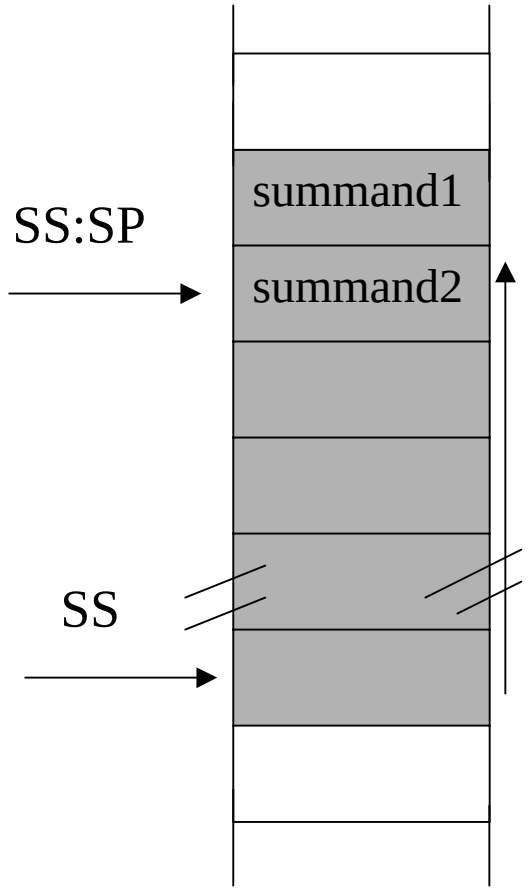
Wichtige Stack-Befehle

- push quelle: 1) SP:=SP-2 (16Bit)
  2) Datentransfer
- pop ziel: 1)Datentransfer
  2) SP:=SP+2
- call proz-name: 1) SP:=SP-2
  2) (SS:SP):=IP
  3) IP:= IP+ Abstand
- ret (wert): 1) IP:=(SS:SP)
  2) SP:=SP+2(+wert)
- pusha/popa : Registersicherung
- pushf/popf : Sicherung Statusregister (16Bit)
  bzw. pushfd/popfd : 32 Bit



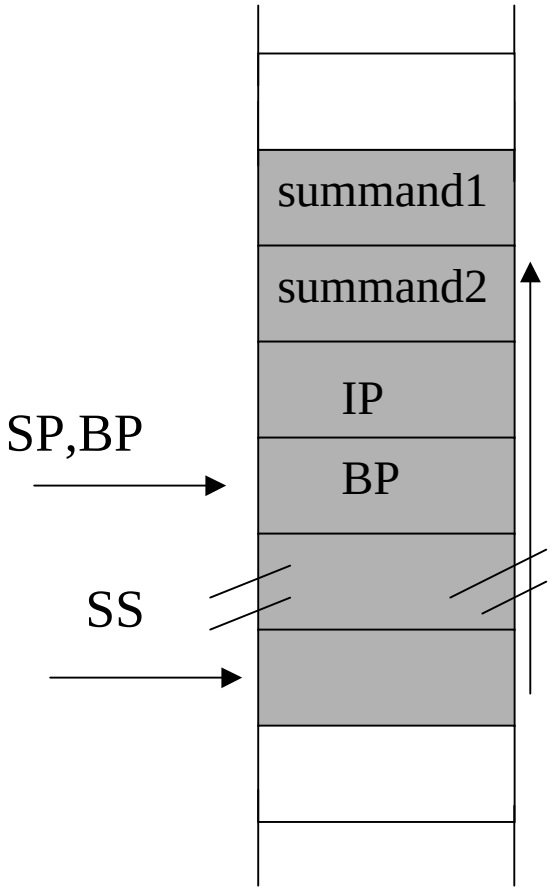
Programmbeispiel : Stack

```
TITLE   Parameterübergabe durch den Stack
.MODEL SMALL
.STACK 100H
.DATA
summand1 DB 126
summand2 DB 15
.CODE
main PROC
      .STARTUP
       push summand1
       push summand2
       call summe
```

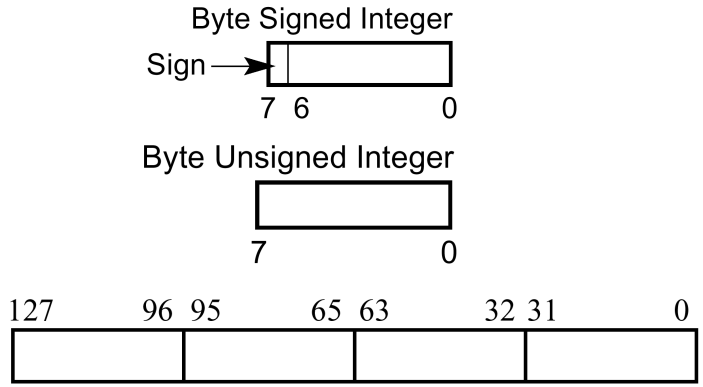


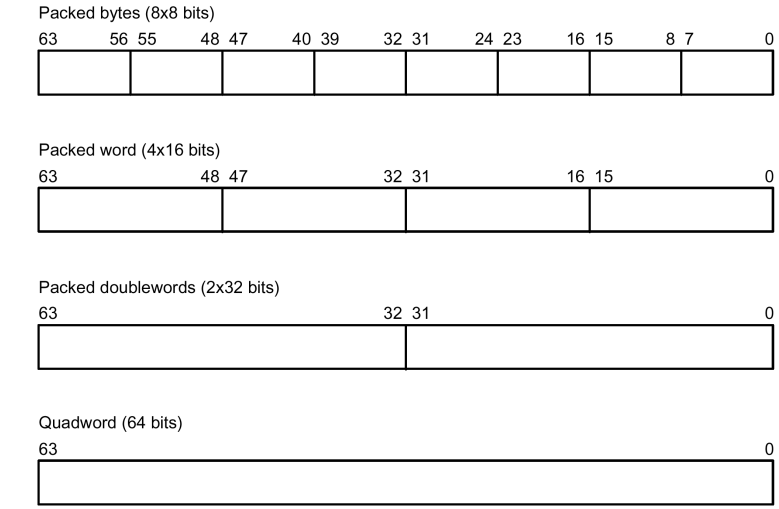
Programmbeispiel: Stack

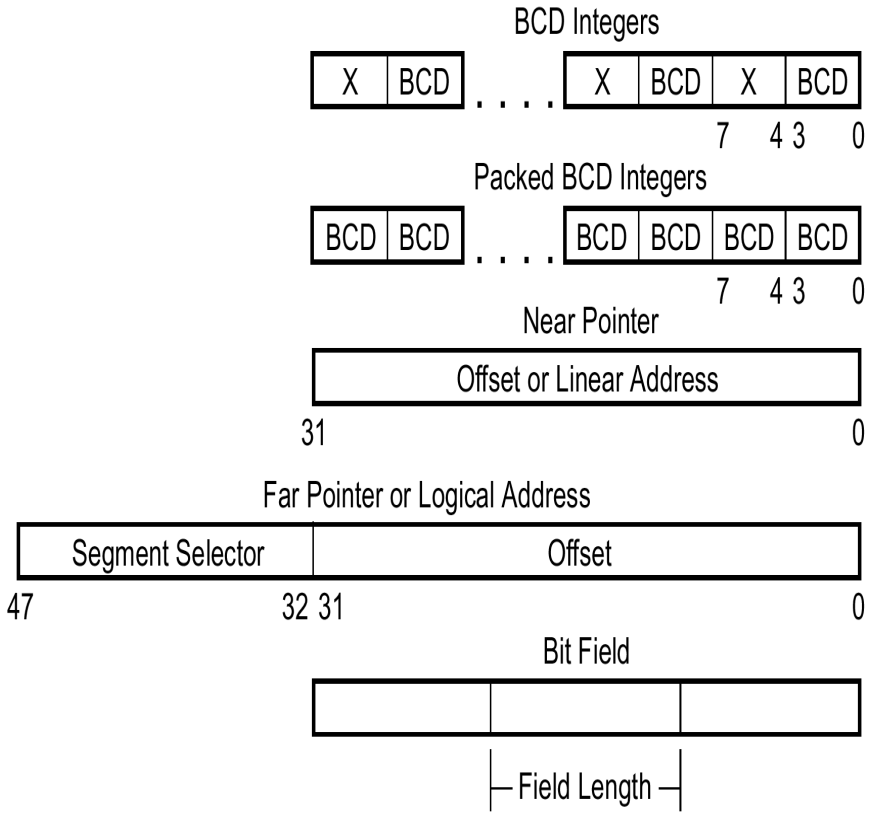
```
done:
        .EXIT
main ENDP
summe PROC
        push BP
        mov BP,SP
        mov AX,[BP+6]
        add AX,[BP+4]
        pop BP
        ret  4
summe ENDP
        END main
```

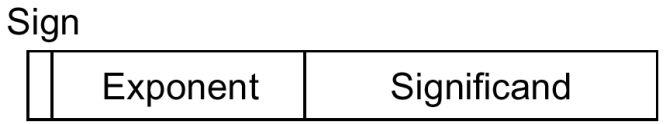


Einige Datentypen

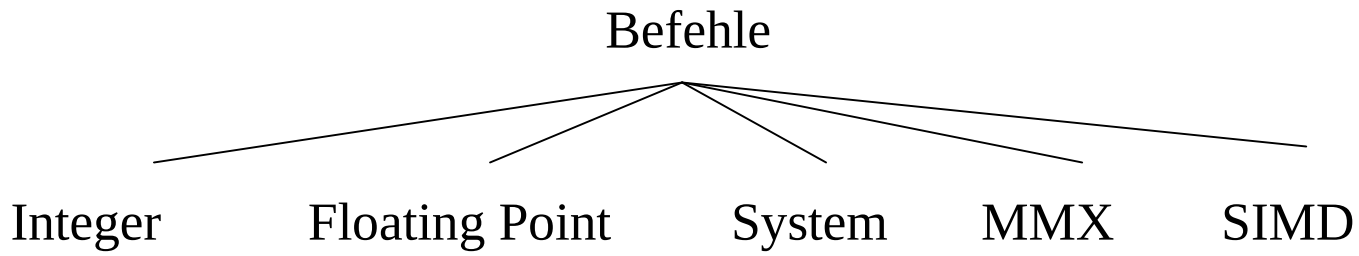








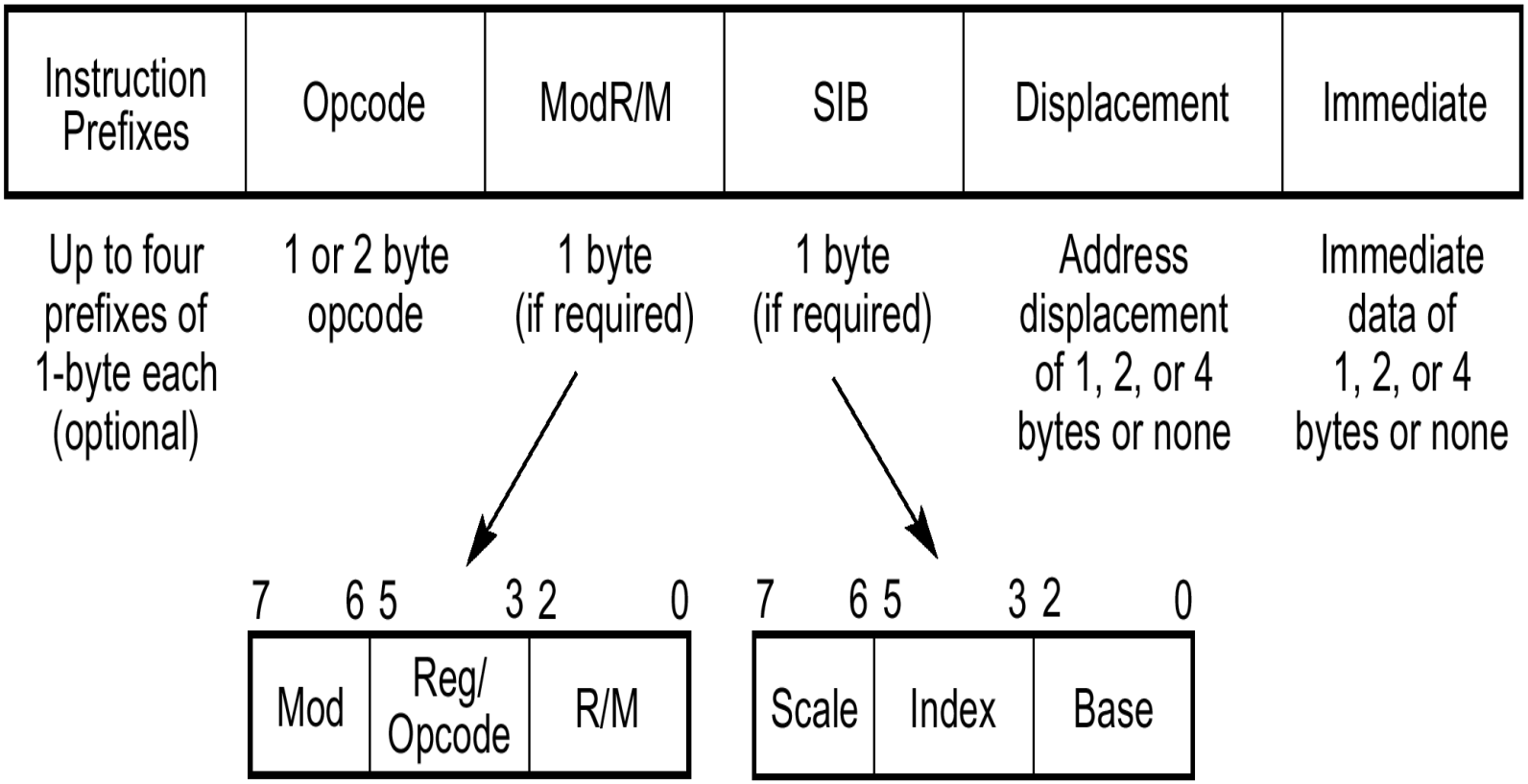
Einige Befehlskategorien



Integer-Befehle:

- Datentransfer (z.B. mov)
- Binärarithmetik
- Dezimalarithmetik
- Logische Befehle (and,or...)
- Verschiebung+Rotation
- Bit und Byte
- Programmkontrolle (loop,jmp,je...)
- String
- Flag-Kontrolle
- Segment-Register (LDS:load far pointer usind DS ...)

Befehlskodierung



Kodierung der Adressierungsarten im MOD R/M-Byte

r8(/r) r16(/r) r32(/r) mm(/r) xmm(/r) /digit (Opcode) REG =			AL AX EAX MM0 XMM0 0 000	CL CX ECX MM1 XMM1 1 001	DL DX EDX MM2 XMM2 2 010	BL BX EBX MM3 XMM3 3 011	AH SP ESP MM4 XMM4 4 100	CH BP EBP MM5 XMM5 5 101	DH SI ESI MM6 XMM6 6 110	BH DI EDI MM7 XMM7 7 111
Effective Address	**Mod**	**R/M**	**Value of ModR/M Byte (in Hexadecimal)**							
[EAX]	00	000	00	08	10	18	20	28	30	38
[ECX]		001	01	09	11	19	21	29	31	39
[EDX]		010	02	0A	12	1A	22	2A	32	3A
[EBX]		011	03	0B	13	1B	23	2B	33	3B
[--][--][1]		100	04	0C	14	1C	24	2C	34	3C
disp32[2]		101	05	0D	15	1D	25	2D	35	3D
[ESI]		110	06	0E	16	1E	26	2E	36	3E
[EDI]		111	07	0F	17	1F	27	2F	37	3F
disp8[EAX][3]	01	000	40	48	50	58	60	68	70	78
disp8[ECX]		001	41	49	51	59	61	69	71	79
disp8[EDX]		010	42	4A	52	5A	62	6A	72	7A
disp8[EBX];		011	43	4B	53	5B	63	6B	73	7B
disp8[--][--]		100	44	4C	54	5C	64	6C	74	7C
disp8[EBP]		101	45	4D	55	5D	65	6D	75	7D
disp8[ESI]		110	46	4E	56	5E	66	6E	76	7E
disp8[EDI]		111	47	4F	57	5F	67	6F	77	7F
disp32[EAX]	10	000	80	88	90	98	A0	A8	B0	B8
disp32[ECX]		001	81	89	91	99	A1	A9	B1	B9
disp32[EDX]		010	82	8A	92	9A	A2	AA	B2	BA
disp32[EBX]		011	83	8B	93	9B	A3	AB	B3	BB
disp32[--][--]		100	84	8C	94	9C	A4	AC	B4	BC
disp32[EBP]		101	85	8D	95	9D	A5	AD	B5	BD
disp32[ESI]		110	86	8E	96	9E	A6	AE	B6	BE
disp32[EDI]		111	87	8F	97	9F	A7	AF	B7	BF
EAX/AX/AL/MM0/XMM0	11	000	C0	C8	D0	D8	E0	E8	F0	F8
ECX/CX/CL/MM1/XMM1		001	C1	C9	D1	D9	E1	E9	F1	F9
EDX/DX/DL/MM2XMM2		010	C2	CA	D2	DA	E2	EA	F2	FA
EBX/BX/BL/MM3/XMM3		011	C3	CB	D3	DB	E3	EB	F3	FB
ESP/SP/AH/MM4/XMM4		100	C4	CC	D4	DC	E4	EC	F4	FC
EBP/BP/CH/MM5/XMM5		101	C5	CD	D5	DD	E5	ED	F5	FD
ESI/SI/DH/MM6/XMM6		110	C6	CE	D6	DE	E6	EE	F6	FE
EDI/DI/BH/MM7/XMM7		111	C7	CF	D7	DF	E7	EF	F7	FF

Kodierung im SIB-Byte

r32 Base = Base =			EAX 0 000	ECX 1 001	EDX 2 010	EBX 3 011	ESP 4 100	[*] 5 101	ESI 6 110	EDI 7 111
Scaled Index	**SS**	**Index**	**Value of SIB Byte (in Hexadecimal)**							
[EAX]	00	000	00	01	02	03	04	05	06	07
[ECX]		001	08	09	0A	0B	0C	0D	0E	0F
[EDX]		010	10	11	12	13	14	15	16	17
[EBX]		011	18	19	1A	1B	1C	1D	1E	1F
none		100	20	21	22	23	24	25	26	27
[EBP]		101	28	29	2A	2B	2C	2D	2E	2F
[ESI]		110	30	31	32	33	34	35	36	37
[EDI]		111	38	39	3A	3B	3C	3D	3E	3F
[EAX*2]	01	000	40	41	42	43	44	45	46	47
[ECX*2]		001	48	49	4A	4B	4C	4D	4E	4F
[EDX*2]		010	50	51	52	53	54	55	56	57
[EBX*2]		011	58	59	5A	5B	5C	5D	5E	5F
none		100	60	61	62	63	64	65	66	67
[EBP*2]		101	68	69	6A	6B	6C	6D	6E	6F
[ESI*2]		110	70	71	72	73	74	75	76	77
[EDI*2]		111	78	79	7A	7B	7C	7D	7E	7F
[EAX*4]	10	000	80	81	82	83	84	85	86	87
[ECX*4]		001	88	89	8A	8B	8C	8D	8E	8F
[EDX*4]		010	90	91	92	93	94	95	96	97
[EBX*4]		011	98	89	9A	9B	9C	9D	9E	9F
none		100	A0	A1	A2	A3	A4	A5	A6	A7
[EBP*4]		101	A8	A9	AA	AB	AC	AD	AE	AF
[ESI*4]		110	B0	B1	B2	B3	B4	B5	B6	B7
[EDI*4]		111	B8	B9	BA	BB	BC	BD	BE	BF
[EAX*8]	11	000	C0	C1	C2	C3	C4	C5	C6	C7
[ECX*8]		001	C8	C9	CA	CB	CC	CD	CE	CF
[EDX*8]		010	D0	D1	D2	D3	D4	D5	D6	D7
[EBX*8]		011	D8	D9	DA	DB	DC	DD	DE	DF
none		100	E0	E1	E2	E3	E4	E5	E6	E7
[EBP*8]		101	E8	E9	EA	EB	EC	ED	EE	EF
[ESI*8]		110	F0	F1	F2	F3	F4	F5	F6	F7
[EDI*8]		111	F8	F9	FA	FB	FC	FD	FE	FF