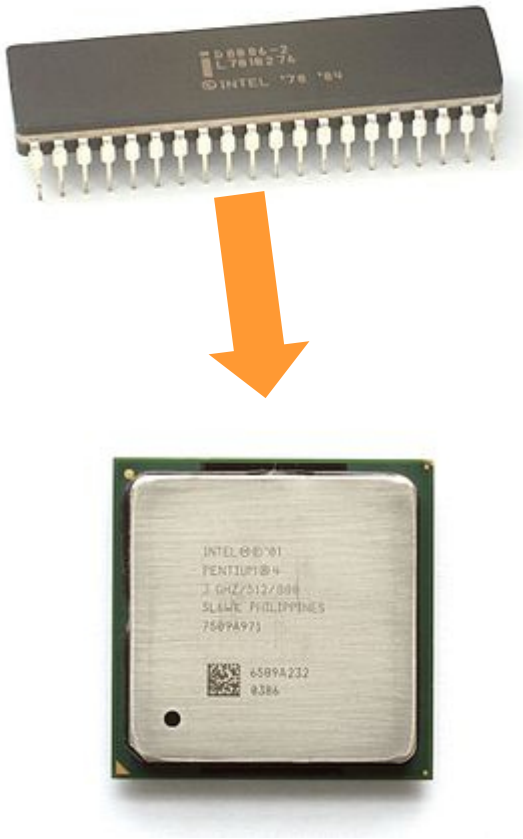
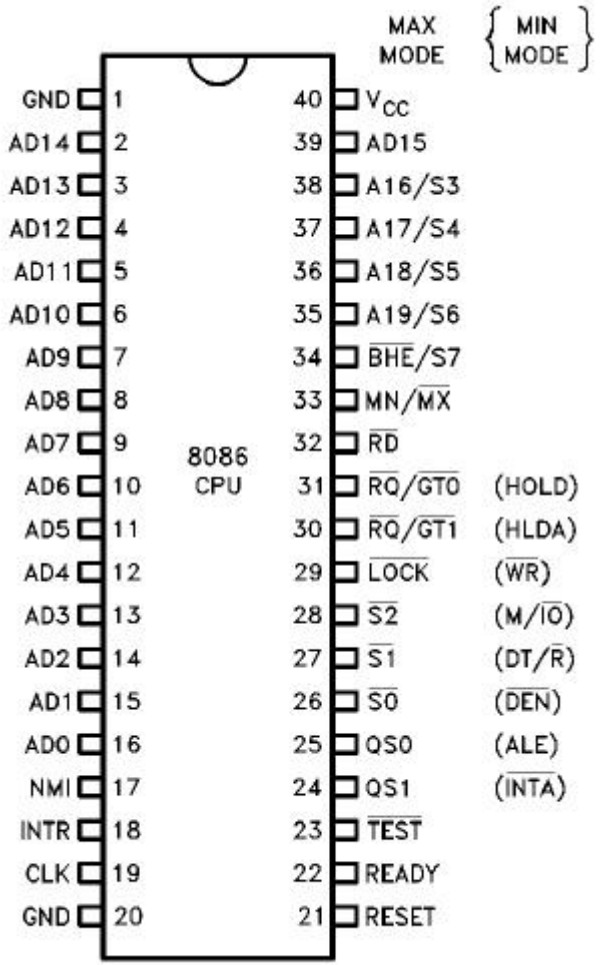
Programmiermodell – 80x86

1. Register
2. Adressierungsarten
3. Instruktionssatz

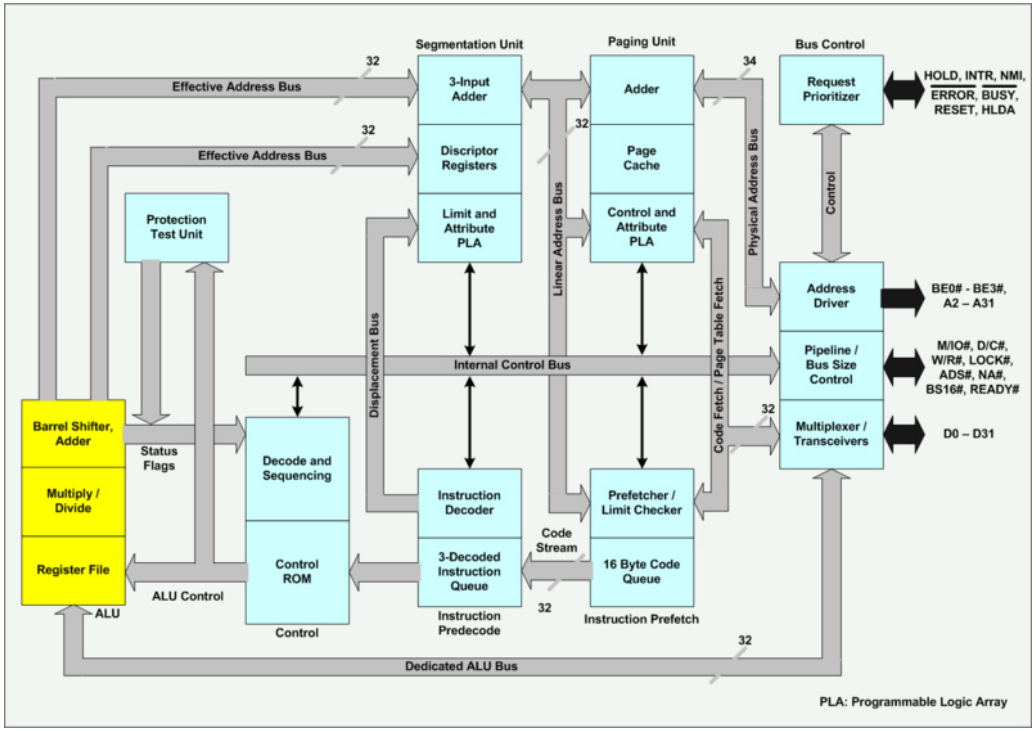


Programmiermodell – 80x86



Programmiermodell – 80x86

1. Register: CPU



Programmiermodell – 80x86

1. Register

Main registers

AH	AL	**AX** (primary accumulator)
BH	BL	**BX** (base, accumulator)
CH	CL	**CX** (counter, accumulator)
DH	DL	**DX** (accumulator, other functions)

Index registers

SI	**S**ource **I**ndex
DI	**D**estination **I**ndex
BP	**B**ase **P**ointer
SP	**S**tack **P**ointer

Status register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	(bit position)
-	-	-	-	O	D	I	T	S	Z	-	A	-	P	-	C	Flags

Segment register

CS	**C**ode **S**egment
DS	**D**ata **S**egment
ES	**E**xtra**S**egment
SS	**S**tack **S**egment

Instruction pointer

IP	**I**nstruction **P**ointer

The 8086 registers

Programmiermodell – 80x86

1. Register: General Purpose – Arithmetik/Pointer

Allgemeine Register

Name	Bemerkung
eax	allgemein verwendbar, spezielle Bedeutung bei Arithmetikbefehlen
ebx	allgemein verwendbar
ecx	allgemein verwendbar, spezielle Bedeutung bei Schleifen
edx	allgemein verwendbar
ebp	Basepointer
esi	Quelle (eng: source) für Stringoperationen
edi	Ziel (eng: destination) für Stringoperationen
esp	Stackpointer

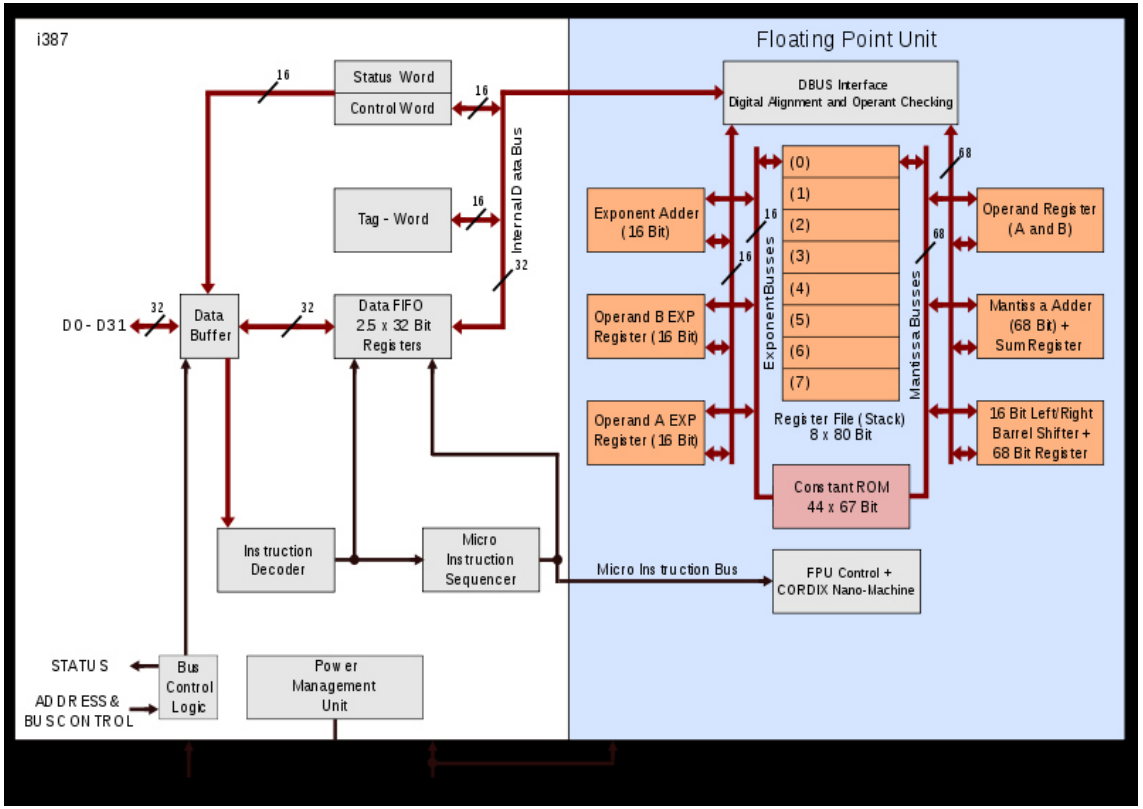
Programmiermodell – 80x86

1. Register: Flags - Status

- Bit 11 – Überlaufflag / *Overflow Flag* (OF)
- Bit 10 – Richtungsflag / Direction Flag *(DF)*
- Bit 9 – *Interruptflag* (IF)
- Bit 8 – Einzelschrittflag / *Trap Flag* (TF)
- Bit 7 – Vorzeichenflag / *Sign Flag* (SF)
- Bit 6 – Nullflag / *Zero Flag* (ZF)
- Bit 4 – *Auxiliary Flag* (AF)
- Bit 2 – Paritätsflag / *Parity Flag* (PF)
- Bit 0 – Übertragsflag / *Carryflag* (CF)

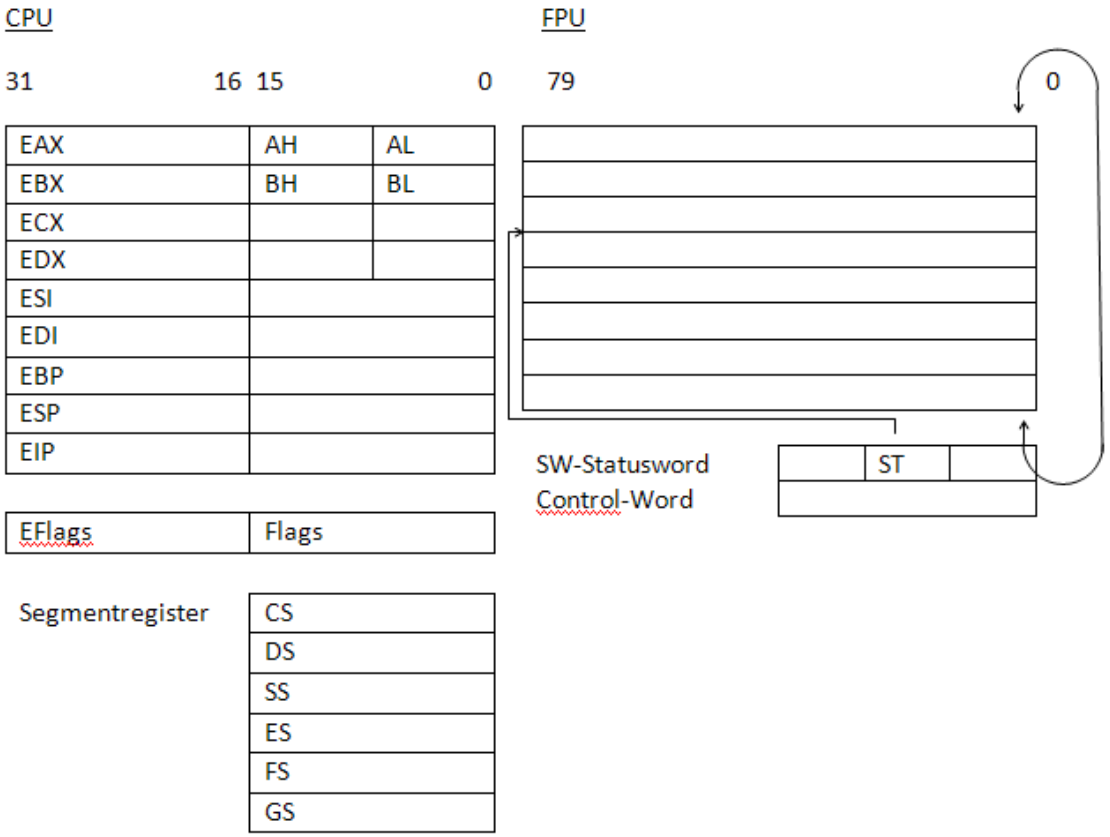
Programmiermodell – 80x86

1. Register. FPU



Programmiermodell – 80x86

1. Register: Zusammenfassung



Programmiermodell – 80x86

1. Adressierungsart

Operand Addressing
Code: CS + EIP (Code segment + Offset)
Stack: SS + ESP (Stack segment + Offset (stack top))
Immediate Operand: *constant_expression*
Register Operand: *register_name*

Programmiermodell – 80x86

1. Adressierungsart: Memory-Operand

Adressierungsart	*Beispiel*	*Hinweis*
Register	MOV AX, BX	Der Inhalt des BX-Registers soll in das AX-Register übertragen werden
Unmittelbar	MOV AX, 123h	Der hexadezimale Wert 123h soll in das AX-Register übertragen werden
Absolut	MOV AX, [123h] MOV AX, DS:[123h]	Der Inhalt der Speicherzelle 123h soll in das AX-Register übertragen werden

Indirekt	MOV AX, [BX]
	MOV AX, DS:[BX]
	
	MOV AX, [BP]
	MOV AX, SS:[BP]
		Der Inhalt der Speicherzelle, deren Adresse in BX, bzw. BP steht. Soll in das AX-Register übertragen werden

Indirekt mit Verschiebeanteil (Displacement)	MOV AX, [BX+123h] MOV AX, DS:[BX-123h]	Anstelle von BX könnten auch die Register DI und SI stehen. Der Inhalt der Speicherzelle deren Adresse sich aus der Summe: Inhalt BX bzw. BP + 123h ergibt soll nach AX gebracht werden

Indirekt mit Basisregister und Indexregister	MOV AX, [BX+SI]
	MOV AX, DS:[BX+SI]
	
	MOV AX, [BP+SI]
	MOV AX, SS:[BP+SI]
		Summenbildung wie beschrieben
		Anstelle von SI könnte auch DI verwendet werden

Indirekt mit Basisregister und Indexregister und Verschiebeanteil	MOV AX, [BX+SI+konst]
	MOV AX, DS:[BX+SI+konst]
	
	MOV AX, [BP+SI+konst]
	MOV AX, SS:[BP+SI+konst]
		Bei ausdrücklicher Nennung sind DS bzw SS auch durch DS, SS und ES ersetzbar.

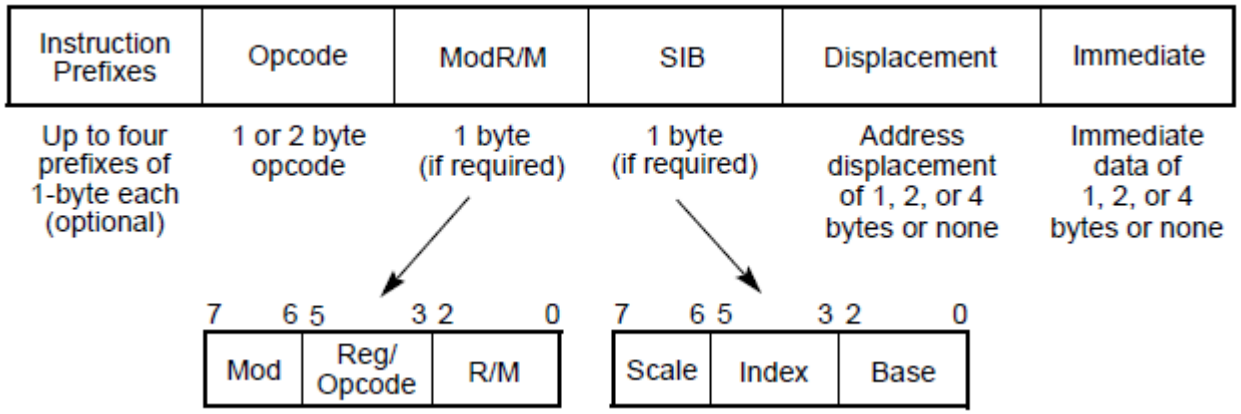
Programmiermodell – 80x86

1. Adressierungsart: Memory-Operand

Base	+(	Index	*	Scale	)+	Displacement
eax ebx ecx edx esp ebp esi edi		eax ebx ecx edx ebp esi edi		1 2 3 4		Name Number

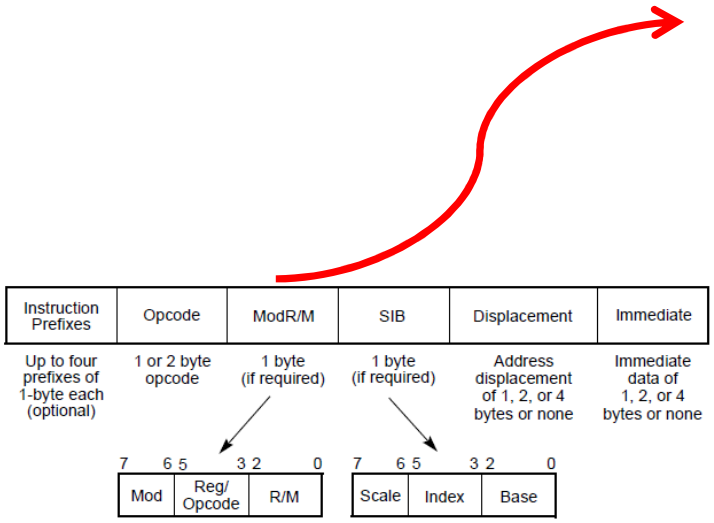
Programmiermodell – 80x86

1. Instruktionsformat



Programmiermodell – 80x86

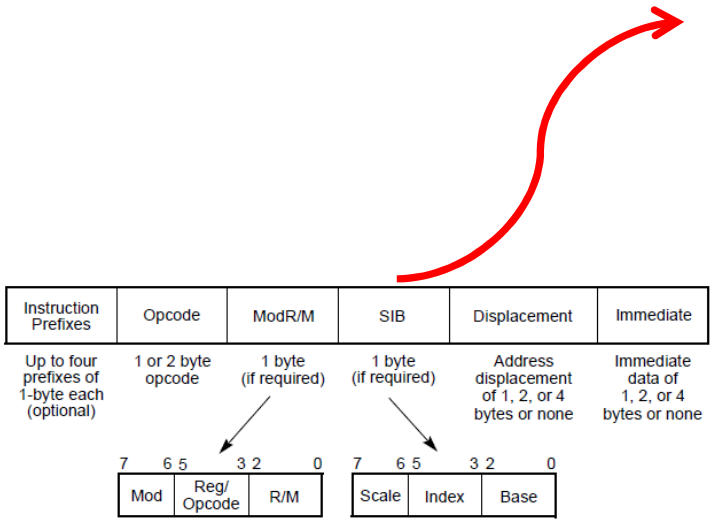
1. Instruktionsformat



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
[EAX]+disp8[3]	01	000	40	48	50	58	60	68	70	78
[ECX]+disp8		001	41	49	51	59	61	69	71	79
[EDX]+disp8		010	42	4A	52	5A	62	6A	72	7A
[EBX]+disp8		011	43	4B	53	5B	63	6B	73	7B
[--][--]+disp8		100	44	4C	54	5C	64	6C	74	7C
[EBP]+disp8		101	45	4D	55	5D	65	6D	75	7D
[ESI]+disp8		110	46	4E	56	5E	66	6E	76	7E
[EDI]+disp8		111	47	4F	57	5F	67	6F	77	7F
[EAX]+disp32	10	000	80	88	90	98	A0	A8	B0	B8
[ECX]+disp32		001	81	89	91	99	A1	A9	B1	B9
[EDX]+disp32		010	82	8A	92	9A	A2	AA	B2	BA
[EBX]+disp32		011	83	8B	93	9B	A3	AB	B3	BB
[--][--]+disp32		100	84	8C	94	9C	A4	AC	B4	BC
[EBP]+disp32		101	85	8D	95	9D	A5	AD	B5	BD
[ESI]+disp32		110	86	8E	96	9E	A6	AE	B6	BE
[EDI]+disp32		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/MM/XMM1		001	C1	C9	D1	D9	E1	E9	F1	F9
EDX/DX/DL/MM2/XMM2		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

Programmiermodell – 80x86

1. Instruktionsformat



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

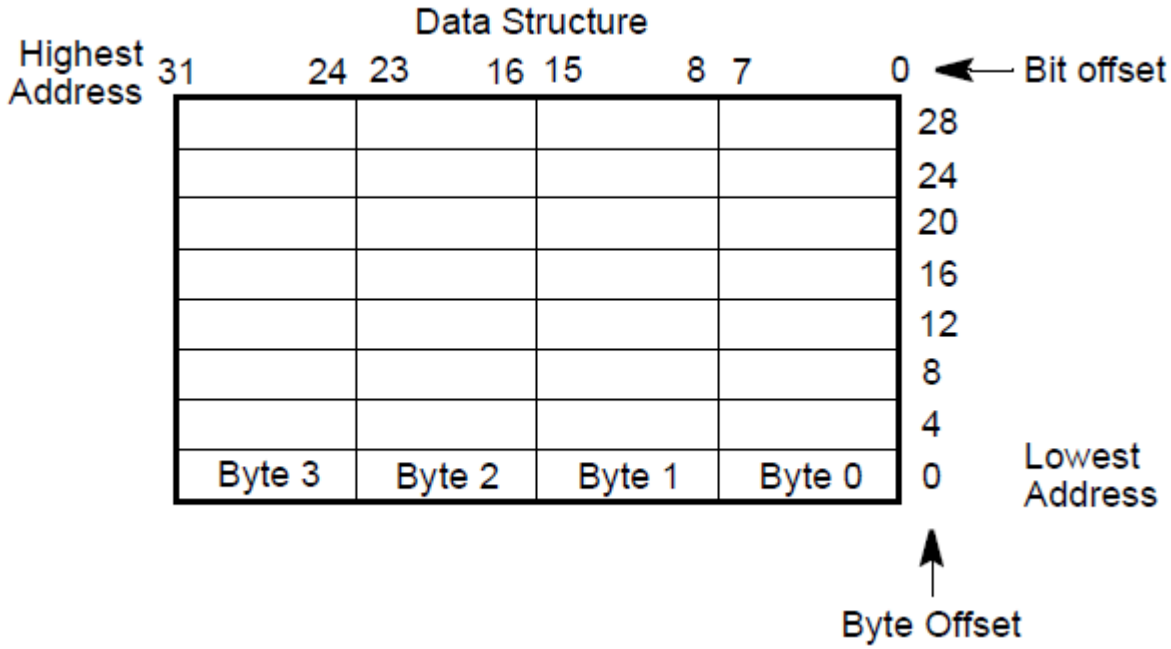
Programmiermodell – 80x86

Syntax Inline-Assembler

```
void fkt (int n){

        int lokal;

        lokal = 5;
                __asm mov DWORD PTR [lokal], 5

        n = lokal;
                __asm {
                        mov EAX, DWORD PTR [lokal]
                        mov DWORD PTR [n], EAX
                }
        return n;
}
```

Programmiermodell – 80x86

Bit and Byte Order



Programmiermodell – 80x86

HW-unterstützte Datentypen:

Anzahl Bytes	Bezeichnung	Verwendung
1	Byte, DB	char, …
2	WORD, DW	short, …
4	DWORD, DD	int, float, long
6	DF	Pointer
8	QWORD, DQ	long, longlong, double
10	TByte, DT	Extended float, precision, BCD, MMX

Programmiermodell – 80x86

Instruktionen: Zuweisung 1 : 1

MOV—Move

Opcode	Instruction	Description
88 /r	MOV *r/m8,r8*	Move *r8* to *r/m8*
89 /r	MOV *r/m16,r16*	Move *r16* to *r/m16*
89 /r	MOV *r/m32,r32*	Move *r32* to *r/m32*
8A /r	MOV *r8,r/m8*	Move *r/m8* to *r8*
8B /r	MOV *r16,r/m16*	Move *r/m16* to *r16*
8B /r	MOV *r32,r/m32*	Move *r/m32* to *r32*
8C /r	MOV *r/m16,Sreg***	Move segment register to *r/m16*
8E /r	MOV *Sreg,r/m16***	Move *r/m16* to segment register
A0	MOV AL,*moffs8**	Move byte at (*seg:offset*) to AL
A1	MOV AX,*moffs16**	Move word at (*seg:offset*) to AX
A1	MOV EAX,*moffs32**	Move doubleword at (*seg:offset*) to EAX
A2	MOV *moffs8**,AL	Move AL to (*seg:offset*)
A3	MOV *moffs16**,AX	Move AX to (*seg:offset*)
A3	MOV *moffs32**,EAX	Move EAX to (*seg:offset*)
B0+ *rb*	MOV *r8,imm8*	Move *imm8* to *r8*
B8+ *rw*	MOV *r16,imm16*	Move *imm16* to *r16*
B8+ *rd*	MOV *r32,imm32*	Move *imm32* to *r32*
C6 /0	MOV *r/m8,imm8*	Move *imm8* to *r/m8*
C7 /0	MOV *r/m16,imm16*	Move *imm16* to *r/m16*
C7 /0	MOV *r/m32,imm32*	Move *imm32* to *r/m32*

Programmiermodell – 80x86

Zuweisung **mit** Vorzeichenerweiterung

MOVSX—Move with Sign-Extension

Opcode	Instruction	Description
0F BE */r*	MOVSX *r16,r/m8*	Move byte to word with sign-extension
0F BE */r*	MOVSX *r32,r/m8*	Move byte to doubleword, sign-extension
0F BF */r*	MOVSX *r32,r/m16*	Move word to doubleword, sign-extension

Programmiermodell – 80x86

Zuweisung **ohne** Vorzeichenerweiterung

MOVZX—Move with Zero-Extend

Opcode	Instruction	Description
0F B6 */r*	MOVZX *r16,r/m8*	Move byte to word with zero-extension
0F B6 */r*	MOVZX *r32,r/m8*	Move byte to doubleword, zero-extension
0F B7 */r*	MOVZX *r32,r/m16*	Move word to doubleword, zero-extension

Programmiermodell – 80x86

Zuweisung **FPU:** Float-Wert in FPU-Register laden

FLD—Load Floating Point Value

Opcode	Instruction	Description
D9 /0	FLD *m32fp*	Push *m32fp* onto the FPU register stack.
DD /0	FLD *m64fp*	Push *m64fp* onto the FPU register stack.
DB /5	FLD *m80fp*	Push *m80fp* onto the FPU register stack.
D9 C0+i	FLD ST(i)	Push ST(i) onto the FPU register stack.

Programmiermodell – 80x86

Zuweisung **FPU:** Int-Wert in FPU-Register laden

FILD—Load Integer

Opcode	Instruction	Description
DF /0	FILD *m16int*	Push *m16int* onto the FPU register stack.
DB /0	FILD *m32int*	Push *m32int* onto the FPU register stack.
DF /5	FILD *m64int*	Push *m64int* onto the FPU register stack.

Programmiermodell – 80x86

Zuweisung **FPU:** Aus FPU-Register als float speichern

FST/FSTP—Store Floating Point Value

Opcode	Instruction	Description
D9 /2	FST *m32fp*	Copy ST(0) to *m32fp*
DD /2	FST *m64fp*	Copy ST(0) to *m64fp*
DD D0+i	FST ST(i)	Copy ST(0) to ST(i)
D9 /3	FSTP *m32fp*	Copy ST(0) to *m32fp* and pop register stack
DD /3	FSTP *m64fp*	Copy ST(0) to *m64fp* and pop register stack
DB /7	FSTP *m80fp*	Copy ST(0) to *m80fp* and pop register stack
DD D8+i	FSTP ST(i)	Copy ST(0) to ST(i) and pop register stack

Programmiermodell – 80x86

Zuweisung **FPU:** Aus FPU-Register als int speichern

FIST/FISTP—Store Integer

Opcode	Instruction	Description
DF /2	FIST *m16int*	Store ST(0) in *m16int*
DB /2	FIST *m32int*	Store ST(0) in *m32int*
DF /3	FISTP *m16int*	Store ST(0) in *m16int* and pop register stack
DB /3	FISTP *m32int*	Store ST(0) in *m32int* and pop register stack
DF /7	FISTP *m64int*	Store ST(0) in *m64int* and pop register stack

Programmiermodell – 80x86

Zuweisung Beispiele

Mov	–	Eins zu eins
Movsx	–	Vorzeichenrichtig erweitern
Movzx	–	Vorzeichenlos erweitern
Fild/Fld + Fist/Fst	–	int <-> float

Programmiermodell – 80x86

Zuweisung Beispiele

Mov	–	Eins zu eins
Movsx	–	Vorzeichenrichtig erweitern
Movzx	–	Vorzeichenlos erweitern
Fild/Fld + Fist/Fst	–	int <-> float

und

Movs	-	Move String -> struct

Programmiermodell – 80x86

Zuweisung String

MOVS/MOVSB/MOVSW/MOVSD—Move Data from String to String

Opcode	Instruction	Description
A4	MOVS m8, m8	Move byte at address DS:(E)SI to address ES:(E)DI
A5	MOVS m16, m16	Move word at address DS:(E)SI to address ES:(E)DI
A5	MOVS m32, m32	Move doubleword at address DS:(E)SI to address ES:(E)DI
A4	MOVSB	Move byte at address DS:(E)SI to address ES:(E)DI
A5	MOVSW	Move word at address DS:(E)SI to address ES:(E)DI
A5	MOVSD	Move doubleword at address DS:(E)SI to address ES:(E)DI

Programmiermodell – 80x86

Zuweisung String

MOVS/MOVSB/MOVSW/MOVSD—Move Data from String to String (Continued)

Operation

```
DEST ←SRC;
IF (byte move)
    THEN IF DF = 0
        THEN
            (E)SI ← (E)SI + 1;
            (E)DI ← (E)DI + 1;
        ELSE
            (E)SI ← (E)SI – 1;
            (E)DI ← (E)DI – 1;
        FI;
    ELSE IF (word move)
        THEN IF DF = 0
            (E)SI ← (E)SI + 2;
            (E)DI ← (E)DI + 2;
        ELSE
            (E)SI ← (E)SI – 2;
            (E)DI ← (E)DI – 2;
        FI;
    ELSE (* doubleword move*)
        THEN IF DF = 0
            (E)SI ← (E)SI + 4;
            (E)DI ← (E)DI + 4;
        ELSE
            (E)SI ← (E)SI – 4;
            (E)DI ← (E)DI – 4;
        FI;
FI;
```

Programmiermodell – 80x86

Zuweisung String

MOVS/MOVSB/MOVSW/MOVSD—Move Data from String to String (Continued)

Operation

```
DEST ←SRC;
IF (byte move)
    THEN IF DF = 0
        THEN
            (E)SI ← (E)SI + 1;
            (E)DI ← (E)DI + 1;
        ELSE
            (E)SI ← (E)SI -
            (E)DI ← (E)DI -
        FI;
    ELSE IF (word move)
        THEN IF DF = 0
            (E)SI ← (E)SI +
            (E)DI ← (E)DI + 2;
        ELSE
            (E)SI ← (E)SI – 2;
            (E)DI ← (E)DI – 2;
        FI;
    ELSE (* doubleword move*)
        THEN IF DF = 0
            (E)SI ← (E)SI + 4;
            (E)DI ← (E)DI + 4;
        ELSE
            (E)SI ← (E)SI – 4;
            (E)DI ← (E)DI – 4;
        FI;
FI;
```

The MOVS, MOVSB, MOVSW, and MOVSD instructions can be preceded by the REP prefix (see "REP/REPE/REPZ/REPNE /REPNZ—Repeat String Operation Prefix" in this chapter) for block moves of ECX bytes, words, or doublewords.