Hinweise – 80x86-Architektur

Dr.-Ing. Volkmar Sieh

Department Informatik 4
Systemsoftware
Friedrich-Alexander-Universität Erlangen-Nürnberg

WS 2025/2026

80x86 – Dokumentation

- Dokumentation: „Intel 64 and IA-32 Architectures Software Developer's Manual“
  - Volume 1: Basic Architecture
  - Volume 2A, 2B, 2C: Instruction Set Architecture
  - Volume 3A, 3B, 3C: System Programming Guide
  - `http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html`

80x86-Architektur

- 80x86-CPUs haben lange Historie (8080, 8086, 80286, 80386, 80486, Pentium, ...)
- Daher vieles nicht logisch aufgebaut, sondern historisch gewachsen (kompatibel).
- Heute würde man vieles anders machen...

Hier nur „kleine" Untermenge der 80x86-Befehle und Fähigkeiten:

- nur Protected Mode
- nur 32-Bit-Mode
- keine Erweiterungen (FPU, MMX, SSE, ...)

Aufbau Opcodes

Befehle unterschiedlich lang (1 Byte bis 16 Bytes)

Ein-Byte-Opcode: lediglich Opcode-Byte; Beispiele:

c3	ret
f4	hlt
fa	cli
fb	sti
...	...

Aufbau Opcodes

Zwei-Byte-Opcode; Prefix 0x0f und Opcode-Byte; Beispiele:

0f	a2	cpuid
0f	aa	rsm
...	...	...

Aufbau Opcodes

Opcode, Register kodiert in Opcode, Bit 0:2; Beispiele:

	40	incl	%eax	48	decl	%eax
	41	incl	%ecx	49	decl	%ecx
	...	...	...	...	...	...
	50	pushl	%eax	58	popl	%eax
	51	pushl	%ecx	59	popl	%ecx
	...	...	...	...	...	...
0f	c8	bswap	%eax			
0f	c9	bswap	%ecx			
...	...	...	...	...	...	...

Register-Kodierung:

0:	eax	1:	ecx	2:	edx	3:	ebx
4:	esp	5:	ebp	6:	esi	7:	edi

Aufbau Opcodes

Prefix-Gruppen:

0xF0, 0xF2, 0xF3: Lock- und Repeat-Prefixe

0x26, 0x2E, 0x36, 0x3E, 0x64, 0x65: ändert Segment

0x66: ändert Operandengröße (16-Bit $\leftrightarrow$ 32-Bit)

0x67: ändert Adressierungsart (alt $\leftrightarrow$ neu)

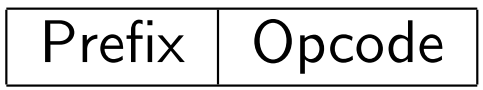
Beispiele:

im 16-Bit-Mode-Segment:

```
      40   incw   %ax
66    40   incl   %eax
```

im 32-Bit-Mode-Segment:

```
      40   incl   %eax
66    40   incw   %ax
```



Prefix: aus jeder Gruppe oben max. ein Prefix-Byte erlaubt
=> max. vier Prefix-Bytes vor Opcode.

Aufbau Opcodes

Opcode mit ModR/M-Byte; Beispiele:

```
21 07                   and   %eax, (%edi)
23 07                   and   (%edi), %eax
01 43 04                add   %eax, 0x4(%ebx)
29 8a 00 01 00 00       sub   %ecx, 0x100(%edx)
31 d5                   xor   %edx, %ebp
```

Prefix (optional)	Opcode	ModR/M	Displacement (gem. ModR/M)

Bit 7-6: Adressierungsart:

- 0: Register indirekt (kein Displacement)
- 1: Register indirekt (8-Bit-Displacement)
- 2: Register indirekt (32-Bit-Displacement)
- 3: Register

Bit 5-3: einzelnes Register

Bit 2-0: Basis-Register bzw. Register

Aufbau Opcodes

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

Aufbau Opcodes

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]+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

Aufbau Opcodes

Adressierungsart „Register indirekt mit Index und Scale-Faktor”:

```
88 04 0b    mov    %al,(%ebx, %ecx, 1)
03 0c 98    add    (%eax, %ebx, 4), %ecx
```

Codierung über sogenanntes „SIB”-Byte.

Prefix (optional)	Opcode	ModR/M	SIB (gem. ModR/M)	Displacement (gem. ModR/M

Aufbau Opcodes

Wird %esp als Register für die Register-indirekt-Adressierungsart verwendet, so folgt ein SIB-Byte:

Aufbau SIB-Byte:

Bit 7-6: Scale-Faktor:

Bit 7-6	Faktor
00	1
01	2
10	4
11	8

Bit 5-3: Index-Register

Bit 2-0: Basis-Register

Aufbau Opcodes

Spezialfall: wird als Indexregister-Nummer die Nummer des Registers %esp angegeben, wird **kein** Indexregister verwendet (Index: 0).

=> z.B. Adressierungsart `4(%esp)` existiert eigentlich nicht

Ersatz: `4(%esp , , )`

Spezialfall: wird als Basisregister-Nummer die Nummer des Registers %ebp angegeben, wird **kein** Basisregister verwendet (Basis: 0).

Anwendung (Beispiel): `array( , %eax, 4)`

Aufbau Opcodes

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

Aufbau Opcodes

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*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

Aufbau Opcodes

Immediates: zusätzliche Konstanten; Beispiele:

	04 01	addb	$0x1, %al
66	83 c0 01	addw	$0x1, %ax
	83 c0 01	addl	$0x1, %eax
66	05 00 01	addw	$0x100, %ax
	05 00 00 01 00	addl	$0x10000, %eax
	b0 12	movb	$0x12, %al
66	81 45 04 34 12	addw	$0x1234, 0x4(%ebp)

Befehls-Opcodes (max.):

Prefix	Opcode	ModR/M	SIB	Displacement	Immediate

Opcode Gruppen

- Manche Opcodes können für mehrere Befehle stehen
- Opcode wird durch MOD/RM Byte, Bits 3-5 erweitert
- Genaueres: Anhang A.4 des Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 2b.
- Beispiel: `81 F9 00 02 00 00: cmpl $0x0200, %ecx`

Prozessor Flags

- Flags sind spezielles Prozessorregister
- Befehle verändern eine Menge an Flags
- Zugriff lediglich indirekt, durch
  - Opcodes, die Flags auswerten (z.B. bedingte Sprünge)
  - Stack (z.B. `pushf`)
  - Opcodes, die einzelne Flags manipulieren (z.B. `sti, cli`)
- Für diese Aufgabe genügen Status-Flags
- Genaueres: Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 1, Kapitel 3.4.3.

64-Bit-Modus

... und 64-Bit-Modus...?

64-Bit-Modus

Neu:

- alle Register jetzt 64-Bit breit (`%rax`, `%rbx`, ...)
- 8 weitere Register (`%r8`, `%r9`, ...)
- Program-Counter-indirekte Adressierung (`disp32(%rip)`)
- ...

64-Bit-Modus

Codierung:

8-Bit-REX-Prefix (statt INC/DEC-Befehlen):

0100	W	R	X	B

- 0100: ehemalige INC/DEC-Bits
- W: „wide”-Instruktion (64-Bit Operanden)
- R: zusätzliches 3. Bit der Registernummer in ModR/M
- X: zusätzliches 3. Bit der Registernummer in SIB-Index
- B: zusätzliches 3. Bit der Registernummer in SIB-Basis

64-Bit-Modus

Zusätzlich: Es existiert jetzt eine PC-relative Adressierungsart:

z.B.:

```
movb $1, buf(%rip)
```