

Rechnerarchitektur

Peter Luksch

Peter.Luksch@cs.tum.edu

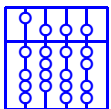
wwwbode.cs.tum.edu

Lehr- und Forschungseinheit Informatik X
Rechnertechnik und Rechnerorganisation
Parallelrechner

WS 2001/2002
30. Oktober 2001

Befehlstypen

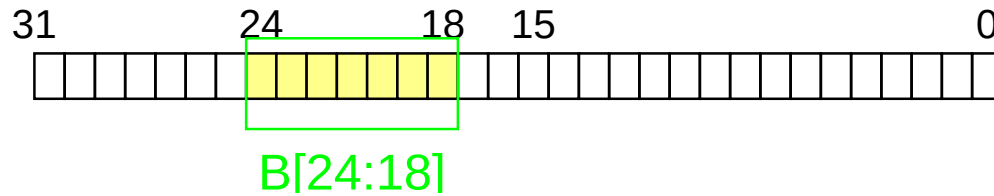
- ❑ Load/Store (integer, floating point)
- ❑ Arithmetik: ganzzahlig und Gleitkomma: ADD, MULT, FADD, FMUL, ...
- ❑ Logische Operationen: AND, OR, ...
- ❑ Schieben, Rotieren
- ❑ ggf. Bitfeldoperationen
- ❑ Verzweigungen



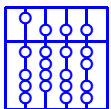
Datenformate

- ❑ vorzeichenlose/vorzeichenbehaftete ganze Zahlen
(*signed/unsigned integers*)
8, 16, 32, 64 bit

- ❑ Bitfelder



- ❑ rationale und reelle Zahlen

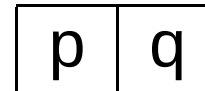


Rationale und reelle Zahlen

rationale Zahlen

$$x \in \mathbb{Q} \Leftrightarrow x = \frac{p}{q}, \quad p, q \in \mathbb{Z}$$

symbolische Repräsentation:



Festkommadarstellung

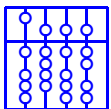
$$\frac{11}{2} = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} = (101.1)_2$$

0	0000000000000000101	100000000000000000
---	---------------------	--------------------

Zahlbereich (Betrag) Festkommdarstellung: $2^{-16} \dots 2^{15}$

Zahlbereich (Betrag) ganze Zahlen: $2^0 : 2^{31}$

⇒ Gewinn ?! Skalierung mit konstantem Faktor



Gleitkommazahlen

$$x = \pm S \times 2^E$$

Normalisierung: Komma gleitet zur Position hinter der ersten '1'

$$\Rightarrow 1 \leq S < 2.$$

$$S = \underbrace{b_0}_{=1}.b_1b_2b_3 \dots b_{p-1}$$

p : *precision*, Genauigkeit

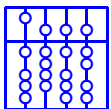
Darstellung:

$\pm$	E	S
-------	-----	-----

z.B.: $(71)_{10} = (1.000111)_2 \times 2^6$

0	00000110	000111000000000000000000
	8bit	23 bit

Maschinengenauigkeit $\epsilon = (0.00 \dots 1)_2 = 2^{-(p-1)}$



IEEE Floating Point Standard

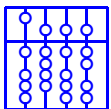
1979: Komitee

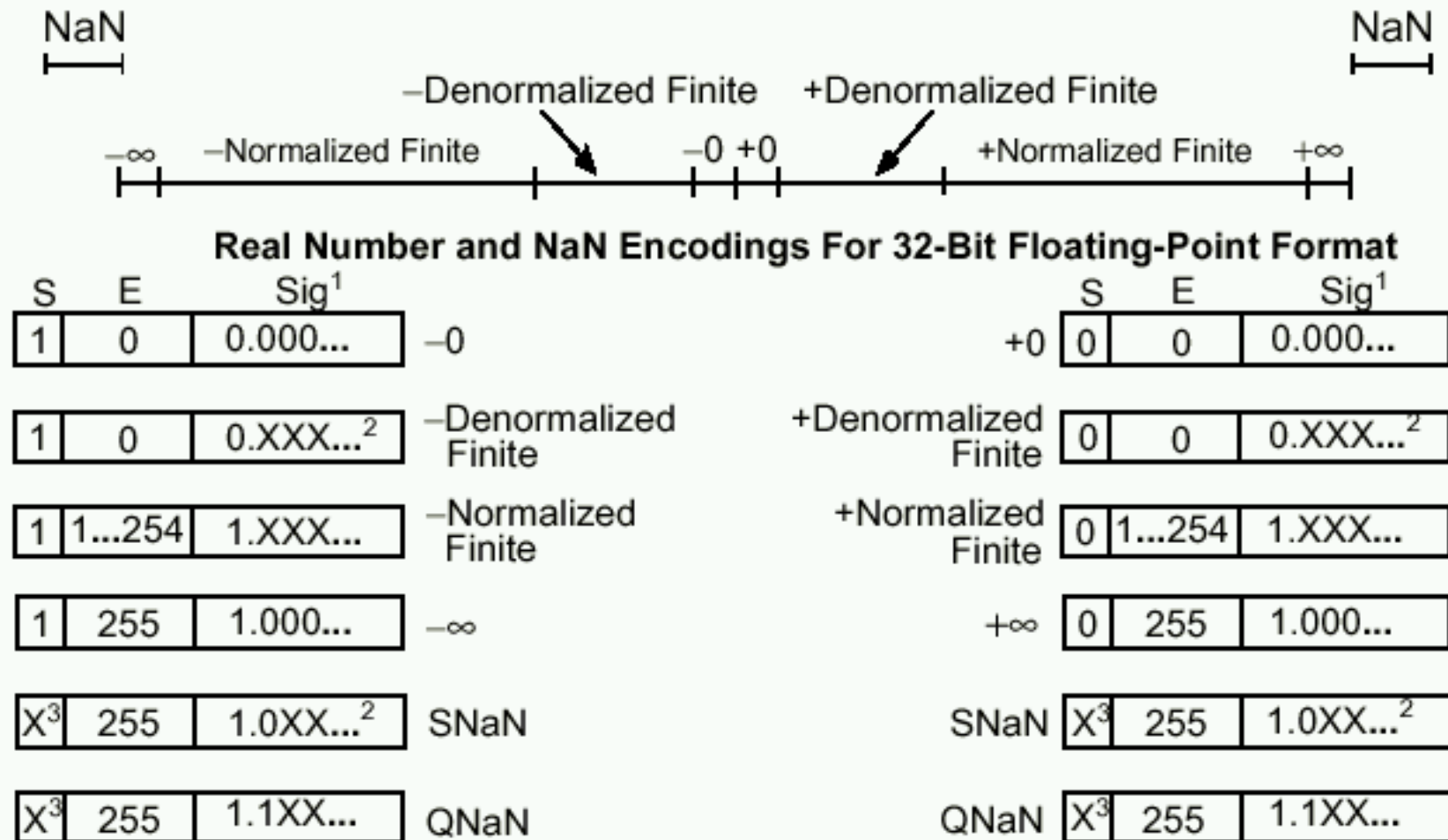
(William Kahan, UC Berkeley + viele Rechnerhersteller)

Single Precision

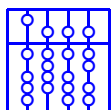
$\pm$	$a_1 \dots a_8$	$b_1 \dots \dots \dots b_{23}$
	E	S

$\pm$	E	S	Wert
X	0	0	0
0	$(255)_{10}$	0	$+\infty$
1	$(255)_{10}$	0	$-\infty$
X	$(255)_{10}$	$\neq 0$	NaN (Not a Number)

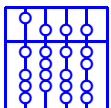
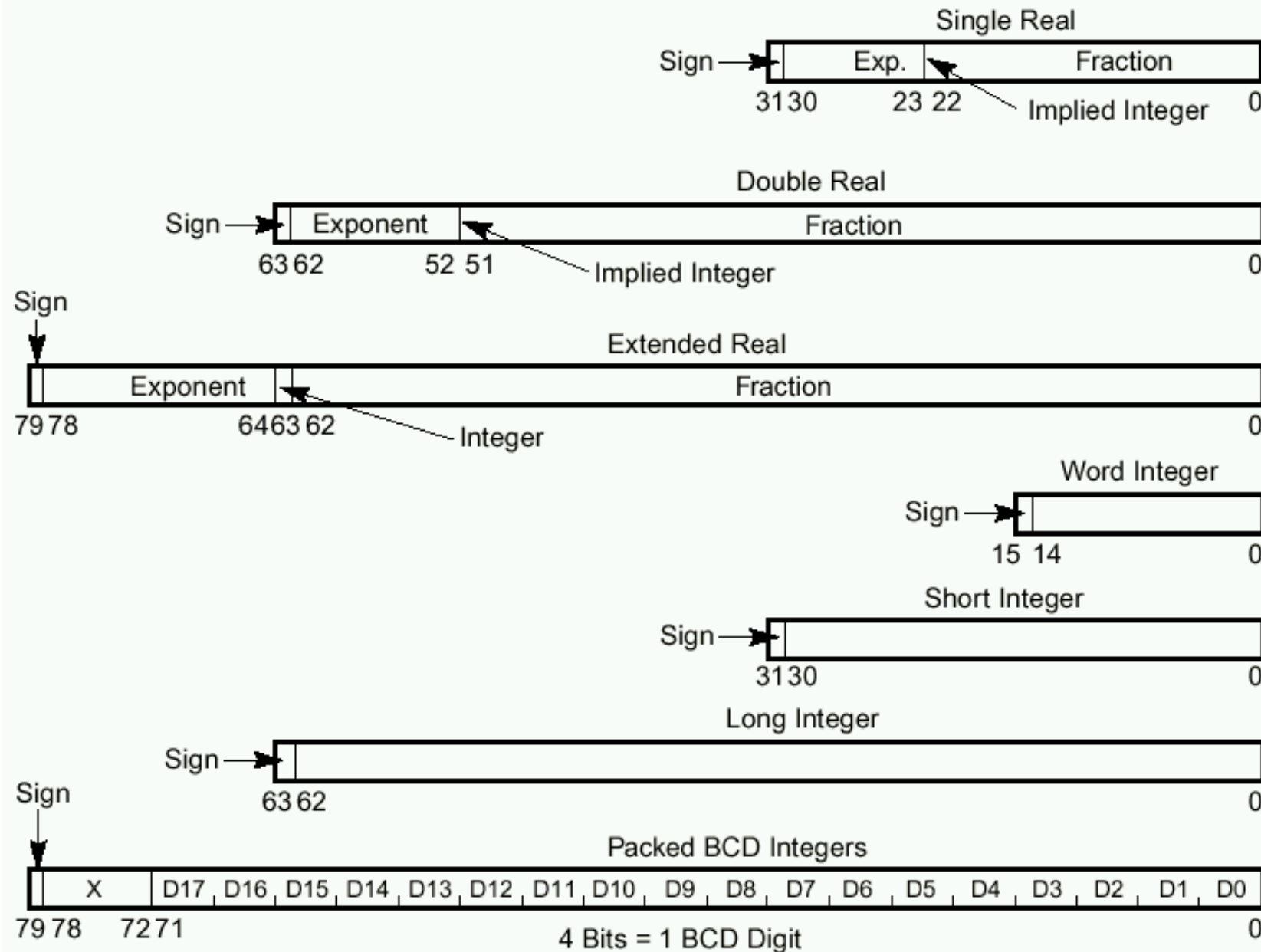


**NOTES:**

1. Integer bit of fraction implied for single-precision floating-point format.
2. Fraction must be non-zero.
3. Sign bit ignored.

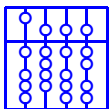


Datenformate der IA-32 FPU



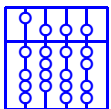
IEEE Floating Point Standard

Parameter	single	double	extended	Quad-Precision
Exp. Bias	+127	+1023	+16383	+16383
max. Exponent	+127	+1023	+16383	+16383
min. Exponent	-126	-1022	-16382	-16382
$x_{\min}$	$1,38 \cdot 10^{-38}$	$2,23 \cdot 10^{-308}$	$3,37 \cdot 10^{-4932}$	$3,37 \cdot 10^{-4932}$
$x_{\max}$	$3,40 \cdot 10^{38}$	$1,97 \cdot 10^{308}$	$1,18 \cdot 10^{4932}$	$1,18 \cdot 10^{4932}$
Precision (bits)	24	53	64	113
Mantisse (bits)	23	52	64	112
Exponent (bits)	8	11	15	15
Speicherbed. (bits)	32	64	80	128



Zusammenfassung Befehlssatzarchitekturen

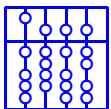
- ❑ Trends der Vergangenheit, Gegenwart, Zukunft
- ❑ Gemeinsamkeiten gegenwärtiger Befehlssatzarchitekturen
- ❑ Datenformate (nicht Datentypen!)
- ❑ Es folgen Fallstudien



Fallstudie CISC:

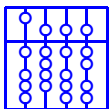
Die IA-32 Befehlssatzarchitektur

- ❑ komplexer Befehlssatz
 - ❑ Pentium III: 340 Befehlstypen
 - ❑ Pentium 4: + 144 Befehlstypen
- ❑ variables Befehlsformat

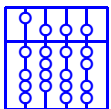


Befehlssatz

- ❑ CMOVcc: bedingter MOVE Befehl
- ❑ XCHG: vertausche Operanden
Speicheroperand $\Rightarrow$ LOCK (Bus für andere Busteilnehmer gesperrt)
- ❑ CMPXCHG: compare and change $\Rightarrow$ Sempahor
- ❑ XADD: vertauscht 2 Operanden, speichert ihre Summe in Zieloperanden $\Rightarrow$ Realisierung von Synchronisationsoperationen in SMP Programmen
- ❑ PUSHA, POPA: push/pop aller Standardregister



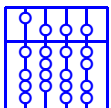
- ❑ CBW: convert byte to word
- ❑ Dezimalarithmetik mit BCD-Zahlen
- ❑ LOOP Schleife mit automatisch dekrementiertem Zähler
- ❑ String-Befehle: MOVS, CMPS, LODS, ...
REP Präfix: wiederhole Befehl bis ECX=0
- ❑ IN, OUT
INS, OUTS: Datenblöcke Port $\leftrightarrow$ Speicher
typische Verwendung mit REP Präfix und Registerindirekter Adressierung (ESI, EDI)
- ❑ XLATB Tabellen-Lookup
EBX: Tabellenanfang, AL: Tabellenindex
 $AL := [EBX+AL]$



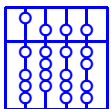
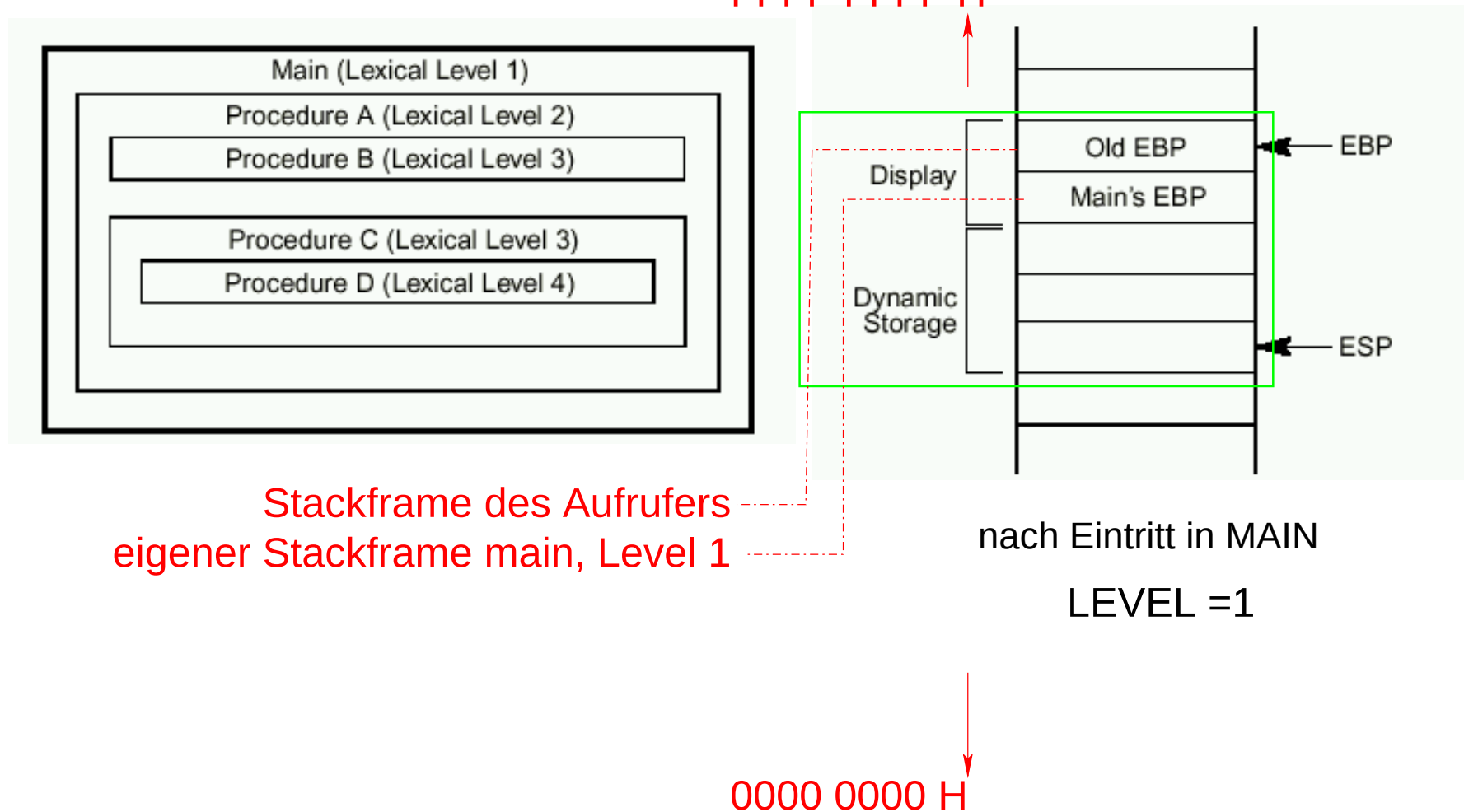
□ ENTER, LEAVE: Prozedur-Prolog, -Epilog. ENTER STORAGE, LEVEL

The following pseudo code shows the formal definition of the ENTER instruction. STORAGE is the number of bytes of dynamic storage to allocate for local variables, and LEVEL is the lexical nesting level.

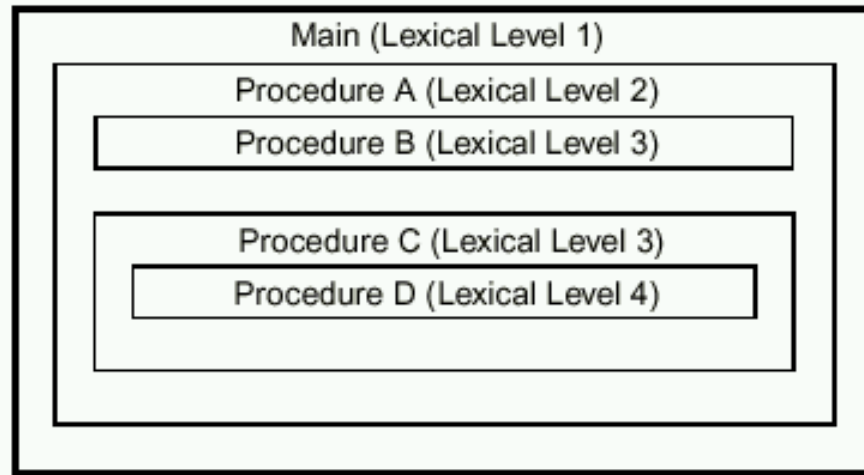
```
PUSH EBP;  
FRAME_PTR ← ESP;  
IF LEVEL > 0  
    THEN  
        DO (LEVEL – 1) times  
            EBP ← EBP – 4;  
            PUSH Pointer(EBP); (* doubleword pointed to by EBP *)  
        OD;  
    PUSH FRAME_PTR;  
FI;  
EBP ← FRAME_PTR;  
ESP ← ESP – STORAGE;
```



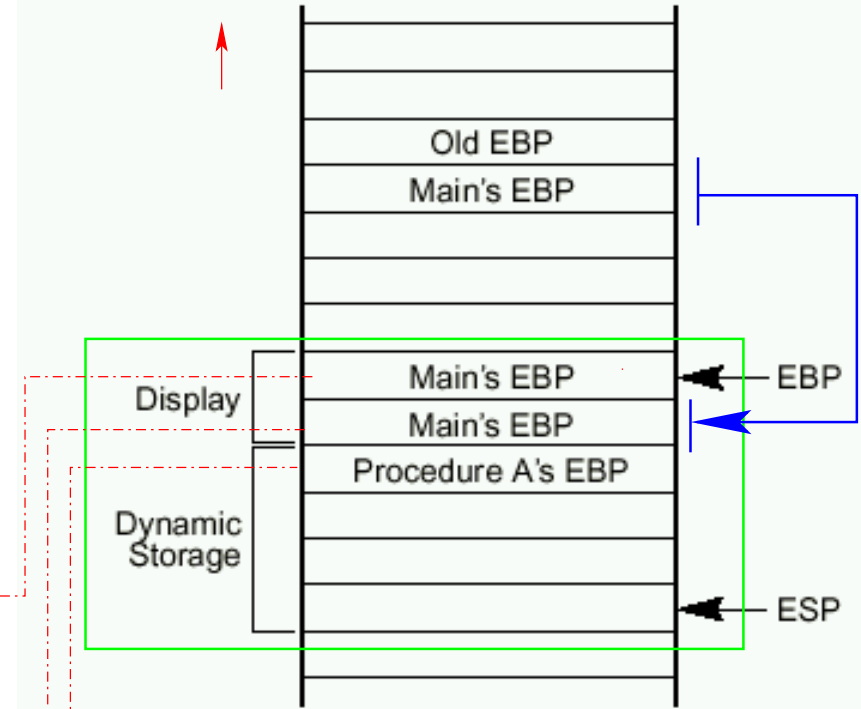
Prozeduraufrufe für blockstrukturierte Sprachen: Der ENTER Befehl am Beispiel



Prozeduraufrufe für blockstrukturierte Sprachen: Der ENTER Befehl am Beispiel



FFFF FFFF H



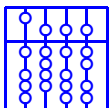
EBP bei Aufruf von A()
Stackframe des Aufrufers

Stackframe statischer Vorgänger (Level 1)

EBP unmittelbar nach Eintritt in A()
eigener Stackframe

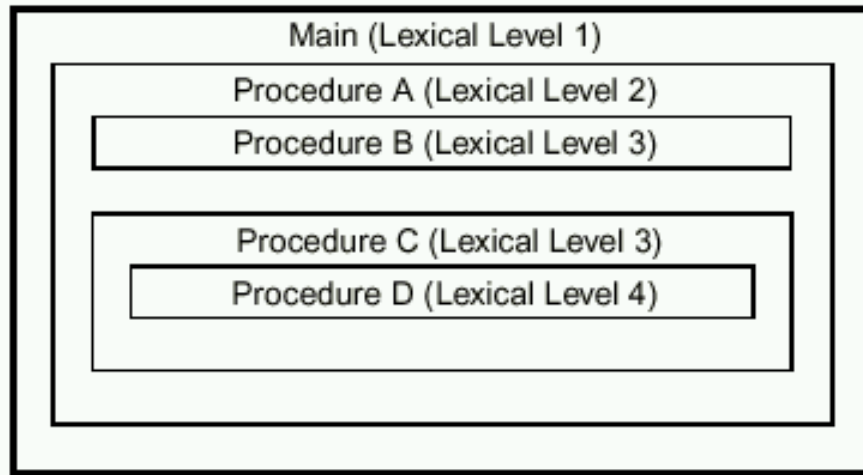
nach Eintritt in Prozedur A

0000 0000 H

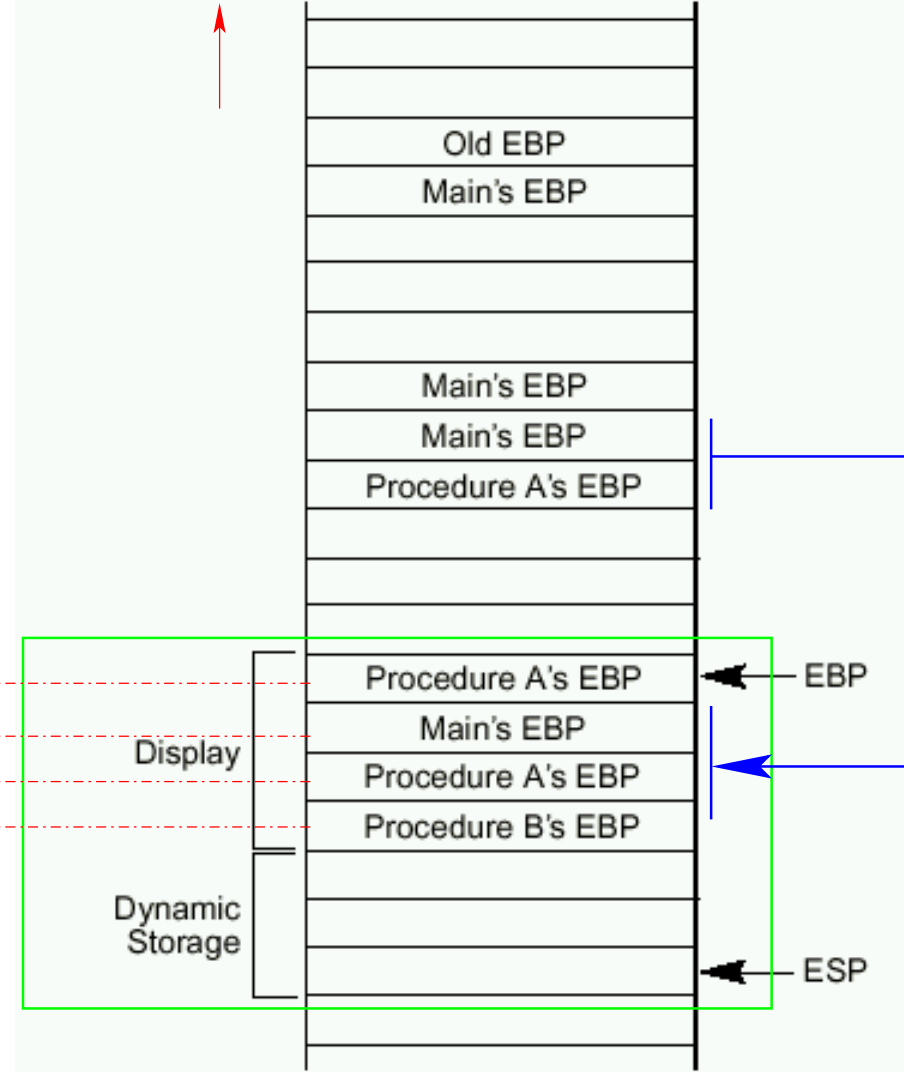


Prozeduraufrufe für blockstrukturierte Sprachen: Der ENTER Befehl am Beispiel

FFFF FFFF H

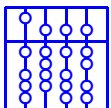


Stackframe Aufrufer
 Stackframe statischer Vorgänger Level 1
 Stackframe statischer Vorgänger Level 2
 eigener Stackframe (Level 3)

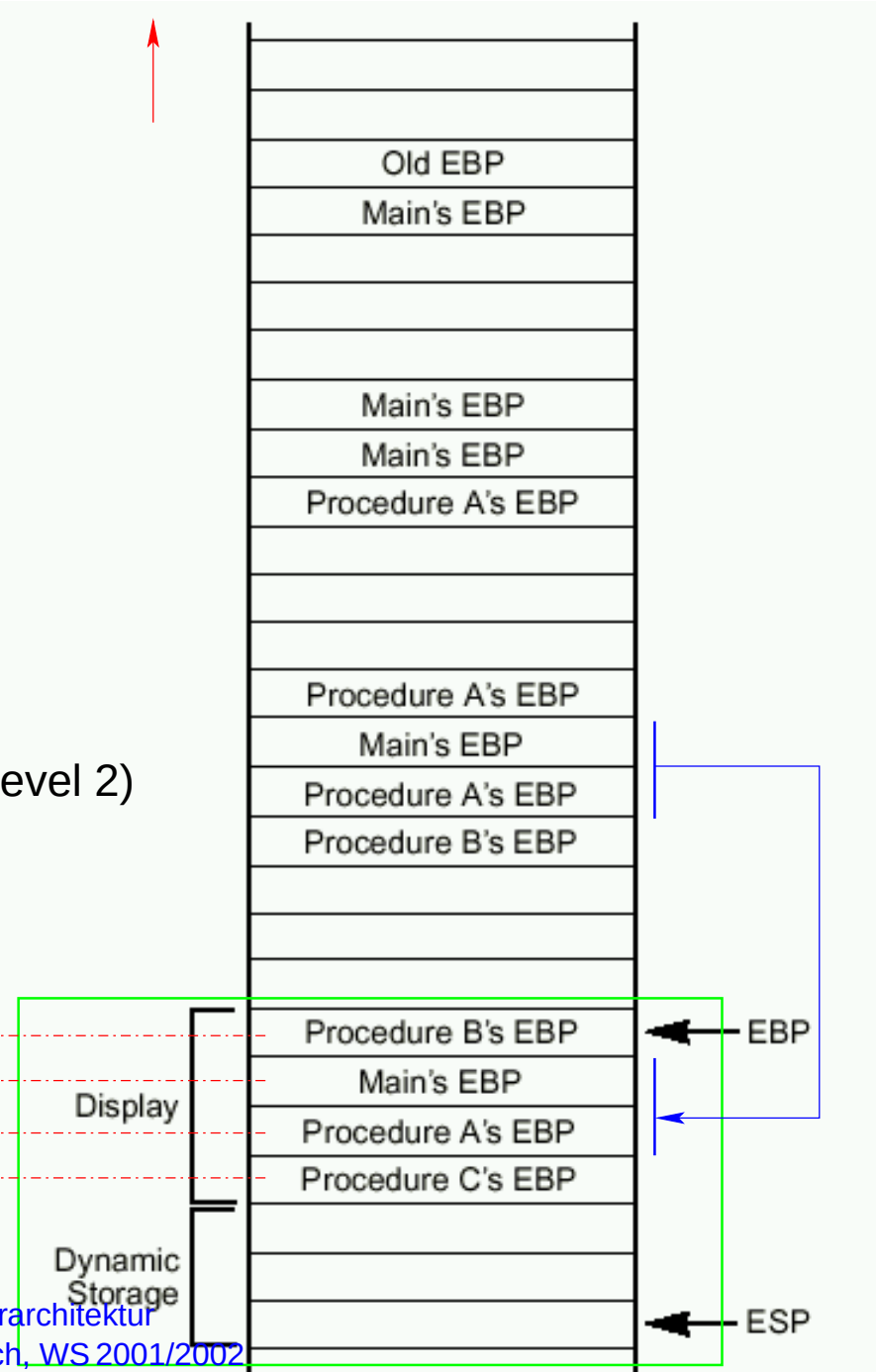


nach Eintritt in Prozedur B

0000 0000 H



FFFF FFFF H



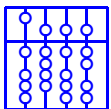
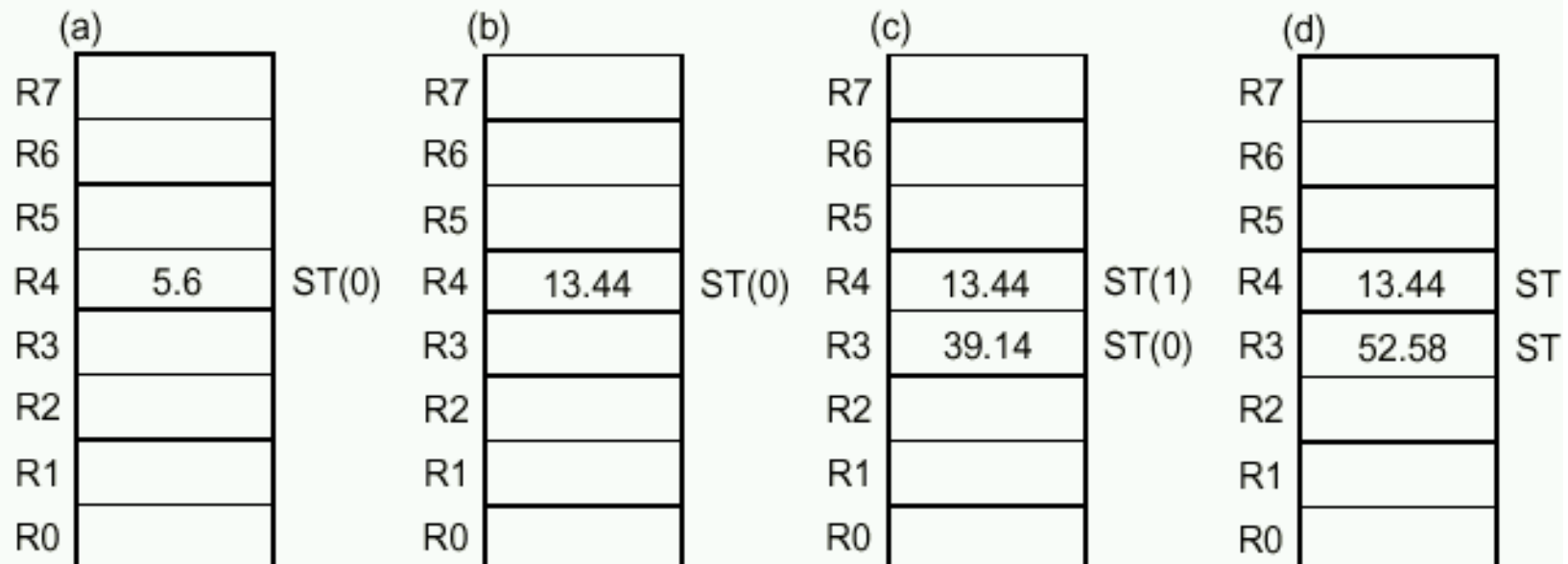
Die x87 FPU: 0-Adreß-Befehle

Computation

Dot Product = $(5.6 \times 2.4) + (3.8 \times 10.3)$

Code:

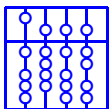
```
FLD  value1 ; (a) value1=5.6
FMUL value2 ; (b) value2=2.4
FLD  value3 ; value3=3.8
FMUL value4 ; (c) value4=10.3
FADD ST(1)  ; (d)
```



Die x87 FPU: 0-Adreß-Befehle

□ FPU-Befehle:

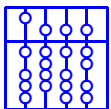
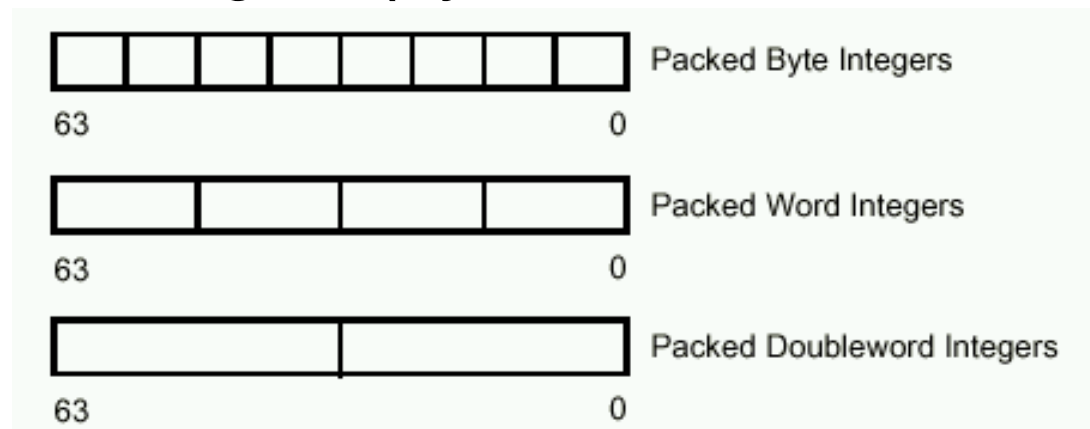
- Addition, Subtraktion, Multiplikation, Division
- Trigonometrische Funktionen
- Logarithmus (Basis 2)
- Vergleiche (Ergebnis im FPU Statuswort).
Erst ab P6 Vergleichsbefehle die CPU Statuswort setzen.



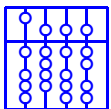
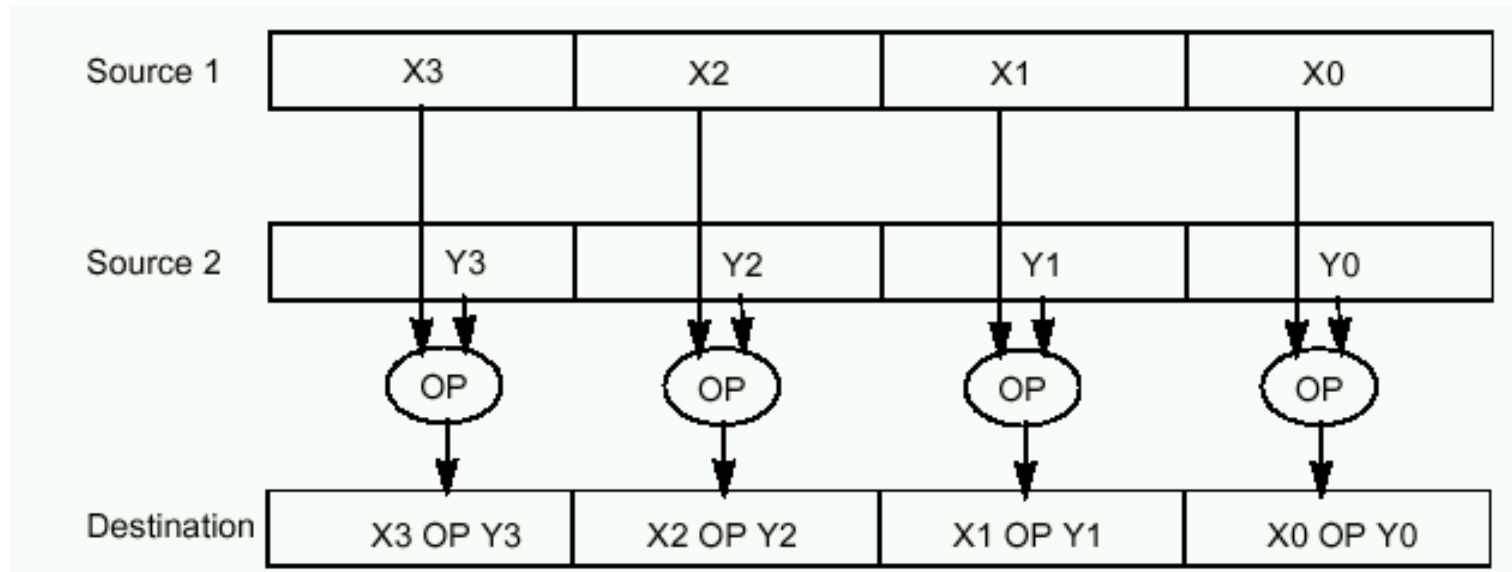
MMX Technologie

Ziel: Leistungssteigerung für Multimedia und Kommunikationsanwendungen

Datentypen: *packed integers* (Byte, 16-bit-, 32-bit-Wort)



MMX: SIMD Operationen



MMX: Arithmetik

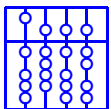
- wraparound arithmetic

ignoriert Bereichsüberschreitungen (overflow)

- Sättigungsarithmetik (*saturation arithmetic*)

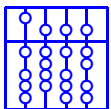
- vorzeichenbehaftet (*signed*):

- *positive overflow*: Ergebnis: größte darstellbare positive Zahl (7FFFH für 16-bit-Worte)
 - *negative overflow*: Ergebnis: betragsgrößte darstellbare negative Zahl (8000H für 16-bit-Worte)



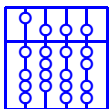
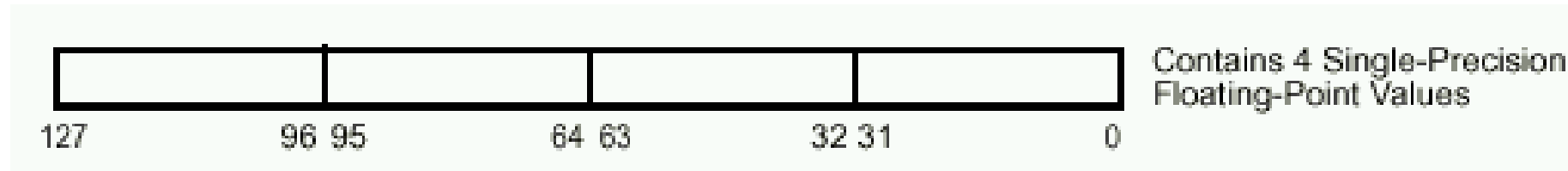
MMX Befehle

- Grundrechenarten auf *packed integers*
- Vergleichsbefehle ($=$, $\neq$, $<$, $>$, $\leq$, $\geq$)
Ergebnis: Maske (1: Bedingung erfüllt)
- Packen und Entpacken von/zu Standard-Integer Format
- logische Operationen (bit-weise UND, ODER, etc.)
- Schiebebefehle

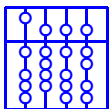
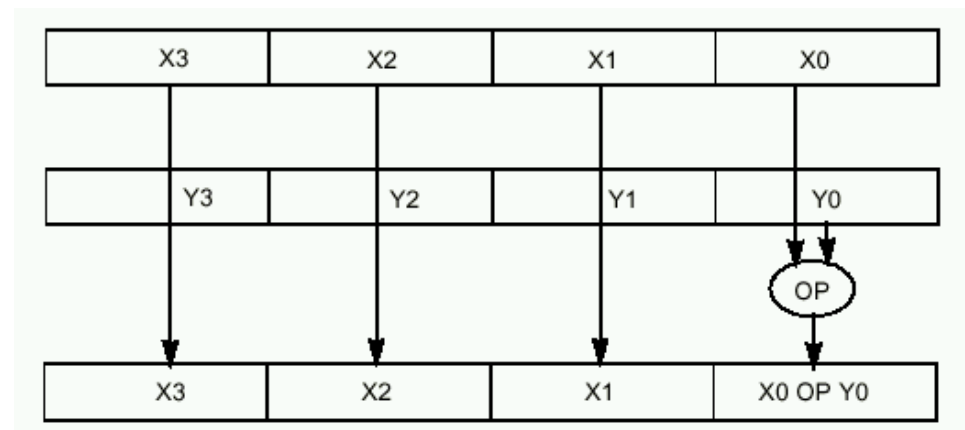
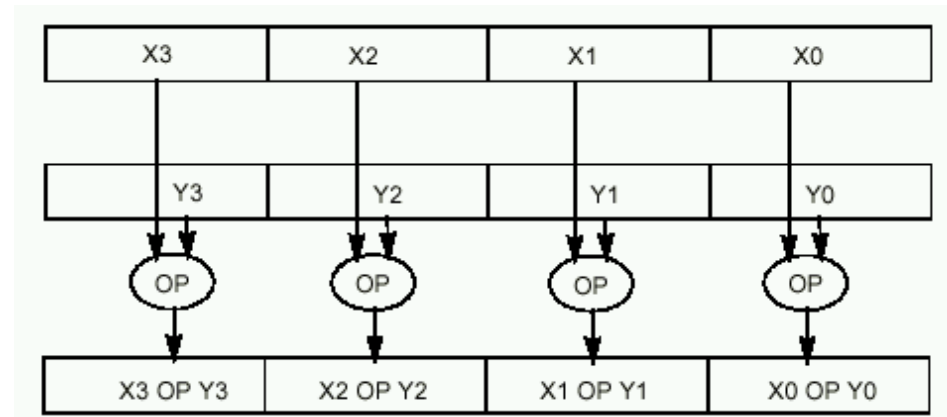


SSE Befehlssatzerweiterung

- ❑ Streaming SIMD Extensions
- ❑ SIMD auf Gleitkommazahlen mit einfacher Genauigkeit (*single precision floating point numbers*)
- ❑ *packed single precision floating point*

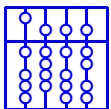


SSE Gleitkommaoperationen

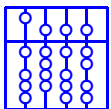
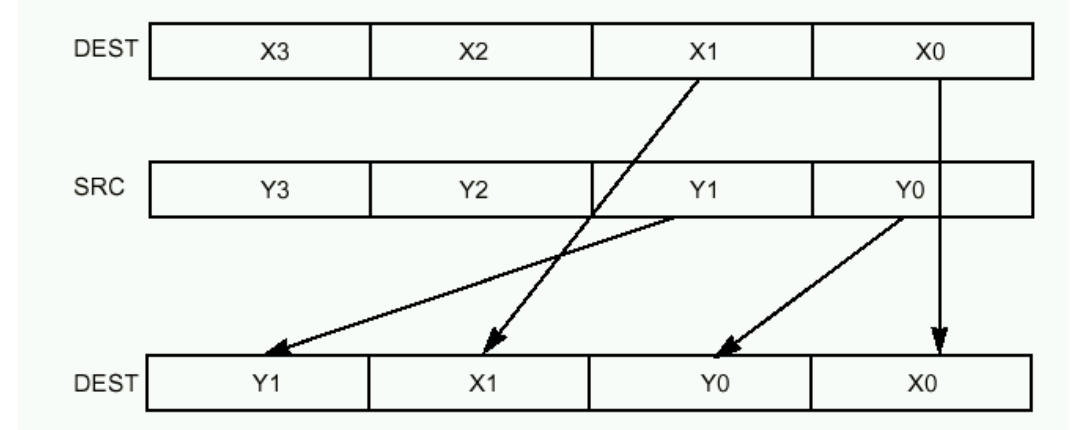
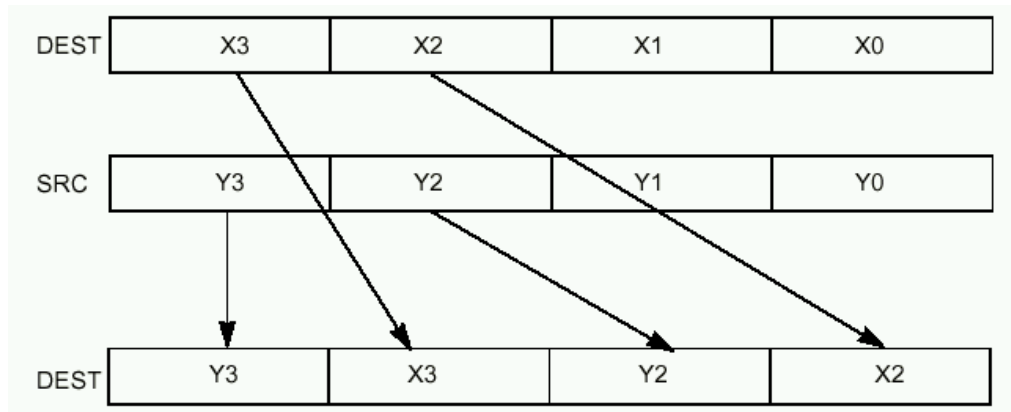


SSE-Befehle

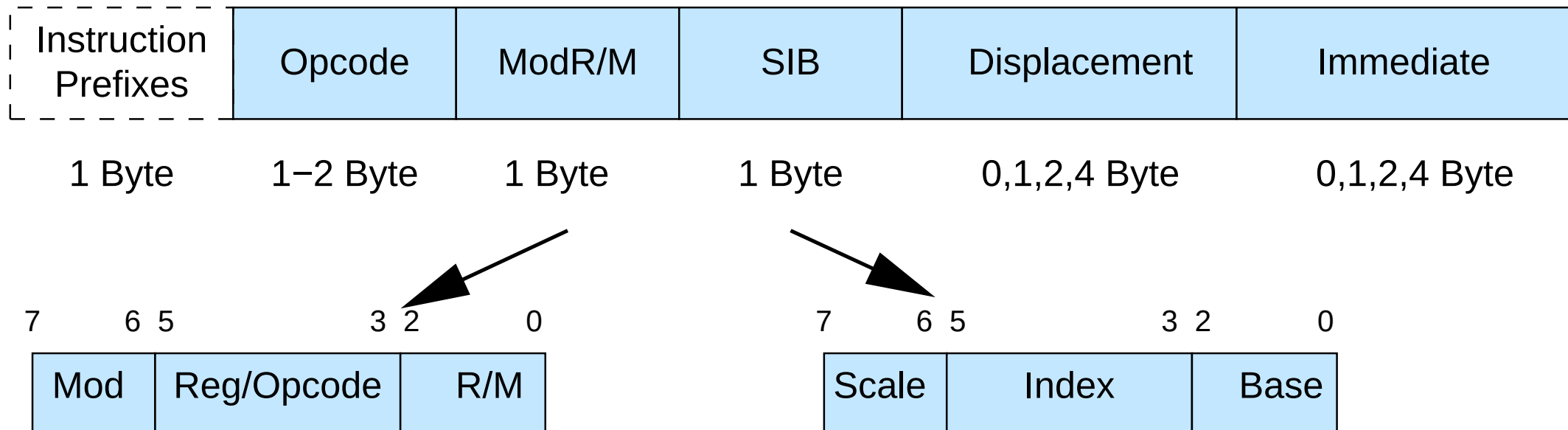
- ❑ $+$, $-$, $\times$, $\div$ auf *packed single precision floating point values*
- ❑ $\sqrt{\cdot}$, $\frac{1}{x}$
- ❑ Minimum, Maximum
- ❑ AND, ANDNOT, OR, XOR
- ❑ Vergleichsbefehle: $=$, $\neq$, $<$, $>$, $\leq$, $\geq$, $\nless$, $\ngtr$, $\dots$, ungeordnet
- ❑ Shuffle-Befehle: beliebige Vertauschung der Werte innerhalb des 128-Bit Registers
- ❑ Unpack und Interleave



SSE: Unpack and Interleave



IA-32 Befehlsformat

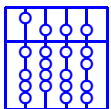


Präfix: z.B. REP (Repeat [String Instruction] until ECX=0)

Opcode: 400+ Befehle $\Rightarrow$ 1-2 Bytes

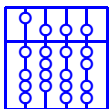
ModR/M codiert 24 Adressierungsarten + 8 Register

SIB: für Adressierungsarten *base+index*, *scale+index*



Befehlsdekodierung

- ❑ variable Befehlslänge $\Rightarrow$ Befehls Grenzen lokalisieren
- ❑ Befehlslänge und Bedeutung folgender Felder erst *nach* Dekodierung des Opcode bekannt
- ❑ Existenz von Displacement/Immediate Feldern erst *nach* Dekodierung der Adressierungsart bekannt
- ❑ Dekodierung zeitaufwendig
- ❑ Dekodierzeit variabel



IA-32 Opcode Mapping (kleiner Ausschnitt)

Table A-1. One-Byte Opcode Map (Continued)

	8	9	A	B	C	D	E	F
0	OR						PUSH	2-byte
	Eb,Gb	Ev,Gv	Gb,Eb	Gv,Ev	AL,lb	eAX,lv	CS	Escape
1	SBB						PUSH	POP
	Eb,Gb	Ev,Gv	Gb,Eb	Gv,Ev	AL,lb	eAX,lv	DS	DS
2	SUB						SEG	DAS
	Eb,Gb	Ev,Gv	Gb,Eb	Gv,Ev	AL,lb	eAX,lv	=CS	
3	CMP	SEG	AAS					
	Eb,Gb	Ev,Gv	Gb,Eb	Gv,Ev	AL,lb	eAX,lv	=DS	
4	DEC General-Purpose Register							
	eAX	eCX	eDX	eBX	eSP	eBP	eSI	eDI
5	POP Into General-Purpose Register							
	eAX	eCX	eDX	eBX	eSP	eBP	eSI	eDI
6	PUSH	IMUL	PUSH	IMUL	INSB	INSW/D	OUTSB	OUTSW/D
	lv	Gv,Ev,lv	lb	Gv,Ev,lb	Yb,DX	Yv,DX	Dx,Xb	DX,Xv
7	Short-Displacement Jump on Condition (Jb)							
	JS	JNS	JP	JNP	JL	JNL	JLE	JNLE

