

SIMD: Media processing

"Media processing" mit dem PC ?!

- steigende Anforderungen für Audio, Video, Image, 3D
- grosse Datenmengen
- aber oft mit geringer Genauigkeit (8 bit .. 16 bit, 32 bit FP)
- x86-FPU ausgereizt

=> Trick: vorhandene ALUs/Datenpfade für SIMD verwenden

Befehlssatzerweiterungen:

- | | | |
|-----------|-------------------------------------|------|
| • MMX | "multimedia extension" | 1996 |
| • 3Dnow! | | 1998 |
| • ISSE | "internet SIMD streaming extension" | 1999 |
| • AltiVec | (PowerPC G4, Macintosh) | 1999 |
| • ISSE2 | | 2000 |

PC-Technologie | SS 2001 | 18.214

SIMD: Flynn-Klassifikation

SISD "single instruction, single data"

=> jeder klassische PC

SIMD "single instruction, multiple data"

=> Feldrechner/Parallelrechner

=> z.B. Connection-Machine 2: 64K Prozessoren

=> eingeschränkt: MMX&Co: 2-8 fach parallel

MIMD "multiple instruction, multiple data"

=> Multiprozessormaschinen

=> z.B. vierfach PentiumPro-Server

MISD :-)

PC-Technologie | SS 2001 | 18.214

SIMD: Literatur

MMX: "The MMX technology page has been removed"

- developer.intel.com/drg/mmx/manuals/
- developer.intel.com/drg/mmx/appnotes/
- Linux "parallel-processing-HOWTO"
- IEEE Micro 8/96 S.42, c't 01/97 S.228ff

ISSE: Intel website:

- developer.intel.com/software/idap/resources/technical_collateral/pentiumiii/
- c't 04/00 S.314 (ISSE/3Dnow/AltiVec)

3D Now! AMD website:

- www.amd.com/K6/K6docs/, www.amd.com/swdev/
- c't 15/98 S.186 ff
- IEEE Micro 3/4-99 S.37ff

PC-Technologie | SS 2001 | 18.214

Amdahl's Gesetz

"Speedup" durch Parallelisierung?

[Gene Amdahl, 1967]

System 1: berechnet Funktion X, zeitlicher Anteil $0 < F < 1$

System 2: Funktion X' ist schneller als X mit "speedup" SX:

$$SX = \text{Zeitbedarf}(X) / \text{Zeitbedarf}(X')$$

Amdahl's Gesetz:
$$S_{\text{gesamt}} = \frac{1}{(1-F) + F/SX}$$

=> Optimierung lohnt nur für häufige Operationen !!

=> Beispiele:

$$SX = 10, F = 0.1, S_{\text{gesamt}} = 1 / (0.9 + 0.01) = 1.09$$

$$SX = 2, F = 0.5, S_{\text{gesamt}} = 1 / (0.5 + 0.25) = 1.33$$

$$SX = 2, F = 0.9, S_{\text{gesamt}} = 1 / (0.1 + 0.45) = 1.82$$

$$SX = 1.1, F = 0.98, S_{\text{gesamt}} = 1 / (0.02 + 0.89) = 1.10$$

PC-Technologie | SS 2001 | 18.214

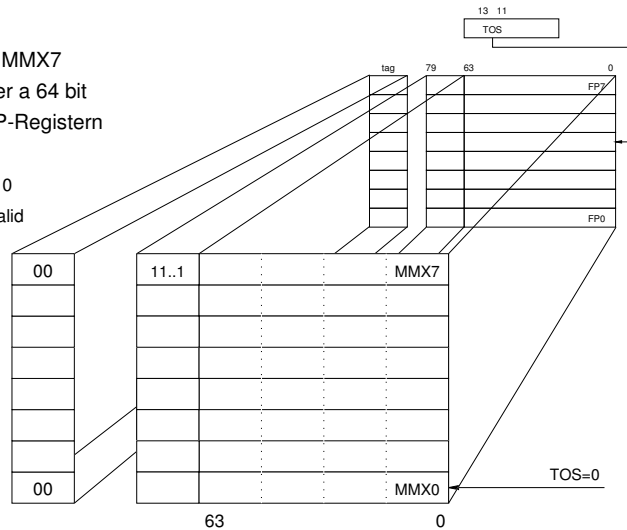
MMX: Entwurfsentscheidungen

Kompatibilität zu alten Betriebssystemen / Apps:

- keine neuen Register möglich => FP-Register nutzen
- keine neuen Exceptions => Überlauf ignorieren
- bestehende Datenpfade nutzen => saturation arithmetic
- möglichst wenig neue Opcodes => 64 bit
- alte Prozessoren und neue Software => Code doppelt
- => MMX DLLs
- Test-Applikationen: => 16 bit dominiert
- (audio/image/MPEG-1/3D-Graphik/...) => MMX DLLs
- keine Tools => Assembler
- optimierte Libraries verfügbar

MMX: Register

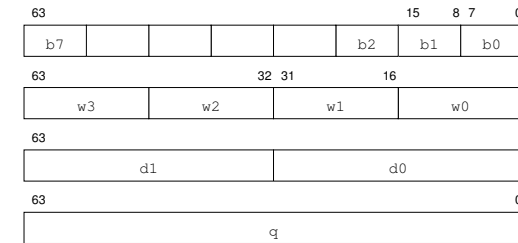
- MMX0 .. MMX7
- 8 Register a 64 bit
- in den FP-Registern
- FP NaN
- FP TOS = 0
- tag 00 = valid



MMX: Datenformate

64-bit Register, 4 Datentypen:

- packed byte *8 / packed word *4 / packed doubleword *2 / quadword
- Zugriff abhängig vom Befehl

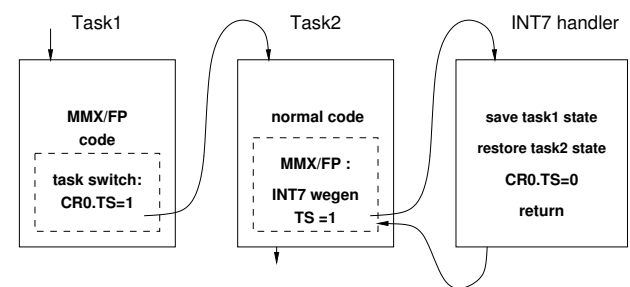


MMX: Befehlssatz

EMMS (FSAV / FRESTOR)		clear MMX state (handle FP regs)
MOVQ	mm1, mm2/mem32	move 32 bit data
MOVQ	mm1, mm2/mem64	move 64 bit data
PACKSSWB	mm1, mm2/mem64	pack 8*16 into 8*8 signed saturate
PUNPCKH	mm1, mm2/mem64	fancy unpacking (see below)
PACKSSDW	mm1, mm2/mem64	pack 4*32 into 4*16 signed saturate
PAND	mm1, mm2/mem64	mm1 AND mm2/mem64 / auch OR/XOR/NAND
PCMPEQB	mm1, mm2/mem64	8*a==b, create bit mask / auch GT
PADDB	mm1, mm2/mem64	8*add 8 bit data
PSUBD	mm1, mm2/mem64	2*sub 32 bit data / signed wrap
PSUBUSD	mm1, mm2/mem64	2*sub 32 bit data / unsigned saturate
PSLL	mm1, mm2/mem64/imm8	shift left mm1 / auch PSRA/PSRL
PMULL/HW	mm1, mm2/mem64	4*mul 16*16 store low/high 16 bits
PMADDWD	mm1, mm2/mem64	MAC 4*16 -> 2*32
insgesamt 57 Befehle		(Varianten B/W/D S/US)

MMX: Multitasking ...

Interaktion mit Betriebssystem / Taskwechsel:

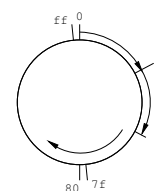


- FP-Register nur bei Bedarf sichern
- vorhandene FP INT7 Routine funktioniert auch für MMX
- keine Anpassung des Betriebssystems notwendig

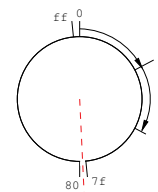
MMX: "Saturation Arithmetic"

was soll bei einem Überlauf passieren?

- wrap-around
..., 125, 126, 127, -128, -127, ...



- saturation
..., 125, 126, 127, 127, 127, ...
- Zahlenkreis "aufgeschnitten"
- gut für DSP-Anwendungen



paddw (wrap around):

a3	a2	a1	7FFFh
+	+	+	+
b3	b2	b1	0004h

a3+b3	a2+b2	a1+b1	8003h

paddsw (saturating):

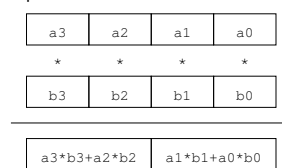
a3	a2	a1	7FFFh
+	+	+	+
b3	b2	b1	0003h

a3+b3	a2+b2	a1+b1	7FFFh

MMX: "packed multiply add word"

für Skalarprodukte:

pmaddwd



```

vector_x_matrix_4x4( MMX64* v, MMX64 *m ) {
    MMX64 v0101, v2323, t0, t1, t2, t3;

    v0101 = punpckldq( v, v ); // unpack v0/v1
    v2323 = punpckhdq( v, v ); // unpack v2/v3

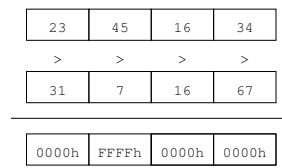
    t0 = pmaddwd( v0101, m[0] ); // v0|v1 * first 2 rows
    t1 = pmaddwd( v2323, m[1] ); // v2|v3 * first 2 rows
    t2 = pmaddwd( v0101, m[2] ); // v0|v1 * last 2 rows
    t3 = pmaddwd( v2323, m[3] ); // v2|v3 * last 2 rows

    t0 = padd( t0, t1 ); // add
    t2 = padd( t2, t3 ); //
    v = packssdw( t0, t2 ); // pack 32->16, saturate
}
  
```

MMX: "packed compare"

Vergleiche / Sprungbefehle:

pcmpgtw:



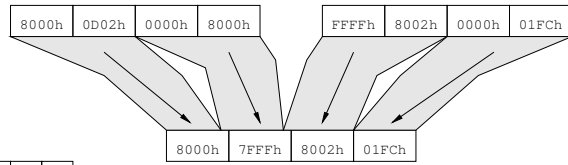
- schlecht parallelisierbar
- Pipeline-Abhängigkeiten

=> keine Sprungbefehle in MMX
=> compare-Operationen setzen Bitmasken

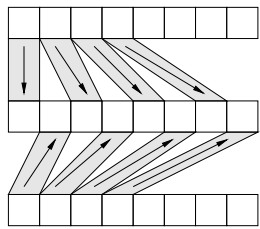
- Bitmasken für logische Ops verwendbar
- Beispiel: chroma-keying

MMX: packssdw / punpckhbw

packssdw: pack with saturation 32 -> 16 signed data:



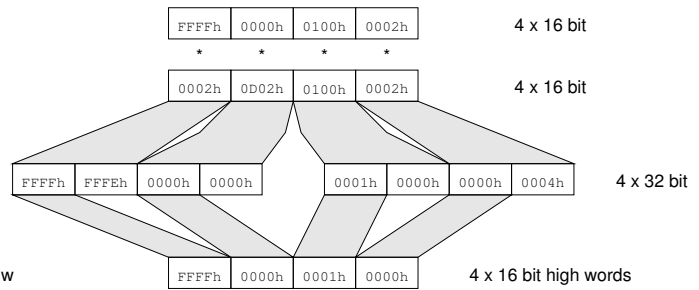
punpckhbw:



punpcklbw: lower 32 bits

MMX: pmullw / pmullhw

pmull[hw]: multiply 4 words, write low/high byte of results:

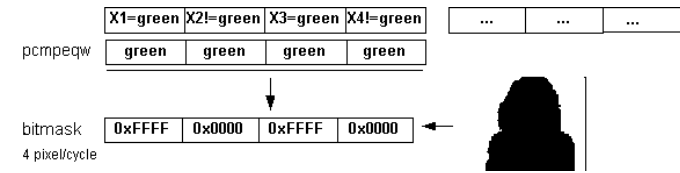


mit Packbefehlen kombinieren, wenn 32-bit Resultate gewünscht

MMX: Chroma Keying (1)

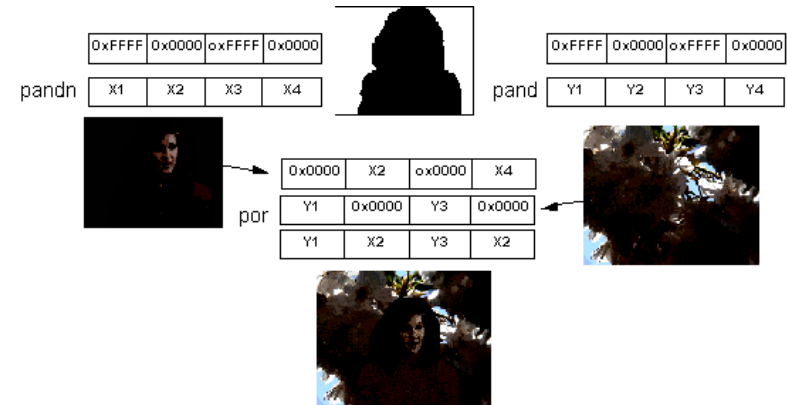
"Wetterbericht":

- MMX berechnet 4 Pixel / Takt
- keine Branch-Befehle
- Schritt 1: Maske erstellen (high-color: 16 bit/pixel)



[Intel MMX appnote]

MMX: Chroma Keying (2)



3Dnow! Befehlssatz

Table 2. 3DNow!™ Floating-Point Instructions

Operation	Function	Opcode Suffix
PAVGUSB	Packed 8-bit Unsigned Integer Averaging	BFh
PFADD	Packed Floating-Point Addition	9Eh
PFSUB	Packed Floating-Point Subtraction	9Ah
PFSUBR	Packed Floating-Point Reverse Subtraction	AAh
PFACC	Packed Floating-Point Accumulate	AEh
PFCMPGE	Packed Floating-Point Comparison, Greater or Equal	90h
PFCMPGT	Packed Floating-Point Comparison, Greater	A0h
PFCMPEQ	Packed Floating-Point Comparison, Equal	80h
PFMIN	Packed Floating-Point Minimum	94h
PFMAX	Packed Floating-Point Maximum	A4h
PI2FD	Packed 32-bit Integer to Floating-Point Conversion	0Dh
PF2ID	Packed Floating-Point to 32-bit Integer	1Dh
PFRCP	Packed Floating-Point Reciprocal Approximation	96h
PFRSQRT	Packed Floating-Point Reciprocal Square Root Approximation	97h
PFMUL	Packed Floating-Point Multiplication	84h
PFRCPT1	Packed Floating-Point Reciprocal First Iteration Step	A6h
PFRSQIT1	Packed Floating-Point Reciprocal Square Root First Iteration Step	A7h
PFRCPT2	Packed Floating-Point Reciprocal/Reciprocal Square Root Second Iteration Step	B6h
PMULHRW	Packed 16-bit Integer Multiply with rounding	B7h

PC-Technologie | SS 2001 | 18.214

3Dnow! Prefetch

Speicherzugriffe in Multimedia-Applikationen:

- reguläre Speicherzugriffsmuster
- ungewöhnliche Lokalität
- viele Daten werden (pro Frame) nur einmal benötigt
- aber regelmässig (in jedem Frame)
- Performance stark von optimaler Cache-Ausnutzung abhängig

=> prefetch-Befehl

- quasi normaler Ladebefehl, aber ohne Zielregister
- gewünschte Daten werden in L1/L2-Cache geladen
- löst keine Exceptions / Page Faults aus

=> "memory streaming"

=> auch für andere Anwendung gut nutzbar (etwa Numerik)

PC-Technologie | SS 2001 | 18.214

3Dnow! Division / Quadratwurzel

- Rechenwerk für Division / Sqrt ist sehr aufwendig
- möglichst wenig Chipfläche für 3Dnow!
- teilweise nur geringe Genauigkeit benötigt
- etwa Shading/Beleuchtungsberechnung für 3D-Graphik

=> Division und Quadratwurzel per Approximation

- erster Befehl liefert 14/15 bit Approximation
- aus Lookup-Table und Interpolation
- mit vollem Takt
- zusätzliche Befehle für Newton-Iteration
- quadratische Konvergenz: zwei Iterationsschritte für volle Genauigkeit
- wenig Hardwareaufwand
- voll in Pipeline integriert, maximaler Durchsatz

PC-Technologie | SS 2001 | 18.214

3Dnow! Division / Quadratwurzel

- Hardware-Dividerer ist sehr aufwendig
- oft wird nicht die volle Genauigkeit benötigt, z.B. Beleuchtungsberechnung bei 3D-Graphik
- Iteration zur Berechnung von $1/b$:

$$x' = x * (2 - b*x)$$

(14-Bit Precision)	MOVD	MM0, [mem]	:	0		w	
	PFRCP	MM0, MM0	:	1/w		1/w	(approx.)
	MOVQ	MM2, [mem]	:	y		x	
	PFMUL	MM2, MM0	:	y/w		x/w	
(24-Bit Precision)	MOVD	MM0, [mem]	:	0		w	
	PFRCP	MM1, MM0	:	1/w		1/w	(approx.)
	PUNPCKLDQ	MM0, MM0	:	w		w	(MMX instruction)
	PFRCPT1	MM0, MM1	:	1/w		1/w	(intermed.)
	MOVQ	MM2, [mem]	:	y		x	
	PFRCPT2	MM0, MM1	:	1/w		1/w	(full prec.)
PFMUL	MM2, MM0	:	y/w		x/w		

PC-Technologie | SS 2001 | 18.214

3Dnow! Quadratwurzel

- Iteration zur Berechnung von $1/\sqrt{b}$:

$$x' = 0.5 * x * (3 - b*x^2)$$

- separate Befehle zur ersten Schätzung (14 bit)
- zwei weitere Befehle zur ersten und zweiten Iteration
- Resultat mit 24-bit Genauigkeit
- abschließende Multiplikation für \sqrt{x} statt $1/\sqrt{x}$

(24-Bit Precision)

```

MOVD    MM0, [mem]    ;          0 | a
PFRSQRT MM1, MM0     ; 1/(sqrt a) | 1/(sqrt a) (approx.)
MOVQ    MM2, MM1     ; X_0 = 1/(sqrt a) (approx.)
PFMUL   MM1, MM1     ; X_0 * X_0 | X_0 * X_0 step 1
PUNPCKLDQ MM0, MM0  ;          a | a (MMX instr.)
PFRSQRT MM1, MM0     ; (intermediate) step 2
PFRCPIT2 MM1, MM2    ; 1/(sqrt a) (full prec.) step 3
PFMUL   MM0, MM1     ; (sqrt a) | (sqrt a)
    
```

3Dnow! Div/Sqrt-Applet

The 3Dnow! Applet:

x: 3.14159264
initial x: 0.02
Start Iteration...

iterated 1/x:	iterated 1/sqrt(x):
0.02	0.02
0.038743362944	0.02998743362944
0.0727104399789801	0.0449387922170619
0.128905319599754	0.0672656279747555
0.20560819541825084	0.10042037030165464
0.27840641013778783	0.1490398662589241
0.3133075688371535	0.21835951335847084
0.31823127488298486	0.3111848093041706
exact 1/x:	exact 1/sqrt(x):
0.31830988756072465	0.5641895847680323

- Eingabeparameter:
 - Argument x
 - Startwert x0
- Ausgabe:
 - Iterationswerte x0,x1,...
 - exakte Lösung
- quadratische Konvergenz
- stark abhängig vom Startwert

3D Now! Apfelmännchen

```

Function IterPasD
  (I,R :Double; Grenze, Tiefe :Paratyp):Paratyp;
  var A,B,C:double;
Begin
  Count:= 0;
  A:=0; B:=0;
  Repeat
    C:= SQR(A) - SQR(B) + R;
    B:= 2*A*B + I;
    A:= C;
    INC (Count);
  Until (abs (A) >Grenze) or (Abs (B) > Grenze)
    or (Count=Tiefe);
IterpasD:=Count;
End;
    
```

3D Now! Apfelmännchen

```

; Quadriere (A + jB)**2 = A**2 - B**2 + j 2*A*B
; Entry MM0 ;A | B
; MM1 ;1 | -1
; MM2 ;R | I
;
loop:
  MOVQ MM3,MM0 ;MM3=A | B
  MOVQ MM4,MM0 ; oh weh
  PSLLQ MM3,32 ; das Vertauschen ist
  PSRLQ MM4,32 ; sehr mühsam ...
  POR MM3,MM4 ;MM3=B | A

  PFMUL MM3,MM0 ;MM3= A*B | A*B
  PFMUL MM0,MM0 ;MM0= A**2 | B**2
  PFMUL MM0,MM1 ;MM0= A**2 | -B**2
  PFACC MM0,MM3 ;MM0= A**2 - B**2 | A*B+A*B
  PFADD MM0,MM2 ;MM0= A**2 - B**2 + R | 2*A*B+I
  ; = A(n+1) | = B(n+1)
  ;iA = INT(A) | iB = Int(B)

  PF2ID MM4,MM0
  MOVQ iA,MM4
; Sieh nach, ob A oder B > GRENZE ist
...
dec CX ; iteration counter
jnz loop
    
```


ISSE: Entwurfsentscheidungen

- Markt fordert 3D
- mindestens doppelte FP-Performance notwendig

2-fach oder 4-fach SIMD?

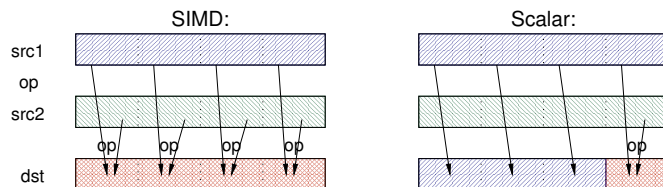
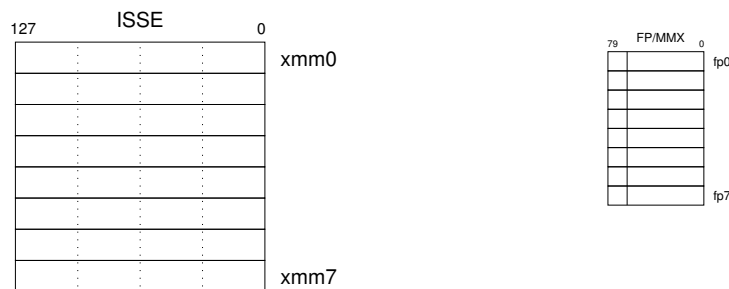
- 128-bit machbar (FP bereits 80-bit)
 - bereits 2 64-bit ALUs auf dem Prozessor
- => 4-fach SIMD

"already register-starved IA32 architecture"

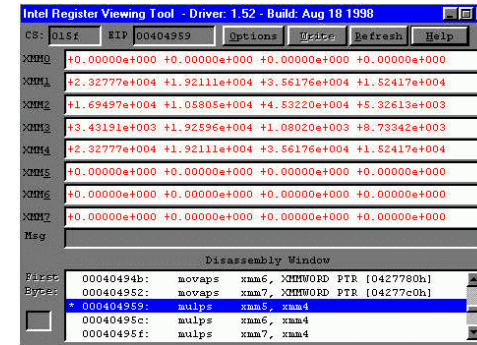
- => neue Register, 128-bit
- => erfordert OS-Unterstützung

- 70 neue Befehle
- sowohl "packed" als auch "scalar ISSE instructions"

ISSE: Register



ISSE: Register Viewing Tool



Softwareentwicklung für MMX / ISSE / 3Dnow:

- nur rudimentäre Compiler- und Tool-Unterstützung
- oft handoptimierter Assembler wg. bester Performance

ISSE: "Streaming"

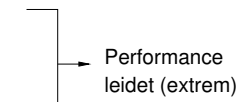
typisch für Medienverarbeitung:

- hohe Datenmenge / Datenrate
- geringe Lokalität: viele Daten (Pixel) werden nur 1x benötigt

=> Cache-"Pollution"

=> herkömmliche Cache-Strategien nutzlos

=> ALUs müssen auf die Daten warten



- 1GHz, 8x SIMD, 100 nsec Speicher: 800 OPs / 1 Zugriff

Streaming:

- Cache-Nutzung anpassen
- Prefetch: Daten rechtzeitig anfordern
- Speicherlatenz fast perfekt versteckt

(für Media-Apps.)

ISSE: Prefetch-Befehl

PREFETCHh—Prefetch Data Into Caches

Opcode	Instruction	Description
0F 18 /1	PREFETCHT0 m8	Move data from m8 closer to the processor using T0 hint.
0F 18 /2	PREFETCHT1 m8	Move data from m8 closer to the processor using T1 hint.
0F 18 /3	PREFETCHT2 m8	Move data from m8 closer to the processor using T2 hint.
0F 18 /0	PREFETCHNTA m8	Move data from m8 closer to the processor using NTA hint.

Description

Fetches the line of data from memory that contains the byte specified with the source operand to a location in the cache hierarchy specified by a locality hint:

- T0 (temporal data)—prefetch data into all cache levels.
- T1 (temporal data with respect to first level cache)—prefetch data in all cache levels except 0th cache level
- T2 (temporal data with respect to second level cache)—prefetch data in all cache levels, except 0th and 1st cache levels.
- NTA (non-temporal data with respect to all cache levels)—prefetch data into non-temporal cache structure. (This hint can be used to minimize pollution of caches.)

The source operand is a byte memory location. (The locality hints are encoded into the machine level instruction using bits 3 through 5 of the ModR/M byte. Use of any ModR/M value other than the specified ones will lead to unpredictable behavior.)

If the line selected is already present in the cache hierarchy at a level closer to the processor, no data movement occurs. Prefetches from uncacheable or WC memory are ignored.

The PREFETCHh instruction is merely a hint and does not affect program behavior. If executed, this instruction moves data closer to the processor in anticipation of future use.

ISSE: Programmierung

Intel VTune Performance Enhancement Environment:

- optimierender Compiler mit ISSE-Unterstützung:
 - Intrinsics
 - Vector Class Library
 - Vectorization
 - Intel Performance Library Suite
- C-Funktionen, Compiler inlining
- Klassen, inlining durch Compiler
- optimierender Compiler
- erfordert 16-Byte Alignment aller Datentypen
- umfangreiche Profiling-Tools
- sehr teuer

ISSE: Programmierung mit "Intrinsics"

```
float xa[SIZE], xb[SIZE], xc[SIZE];
float q;

void do_c_triad() {
    for( int j=0; j < SIZE; j++ ) {
        xa[j] = xb[j] + q*xc[j];
    }
}
```

ISSE-Programmierung mit "Intrinsics" und VTUNE:

```
#define VECTOR_SIZE 4
__declspec(align(16)) float xa[SIZE], xb[SIZE], xc[SIZE];
float q;

void do_intrin_triad() {
    __m128 tmp0, tmp1;

    tmp1 = _mm_set_ps1(q);
    for( int j=0; j < SIZE; j+= VECTOR_SIZE) {
        tmp0 = _mm_mul_ps( *((__m128 *) &xc[j]), tmp1 );
        *((__m128 *) &xa[j]) =
            _mm_add_ps(tmp0, *((__m128 *) &xb[j]);
    }
}
```

ISSE: AoS / SoA

Array of Structures:

```
struct
{
    float A, B, C;
} AoS_data[1000];
```

- Daten lokal
- Anordnung schlecht für SIMD

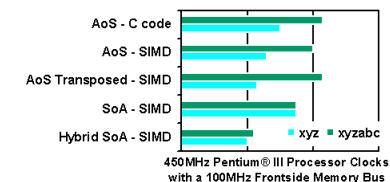
Structure of Arrays:

```
struct
{
    float A[1000], B[1000], C[1000];
} SoA_data;
```

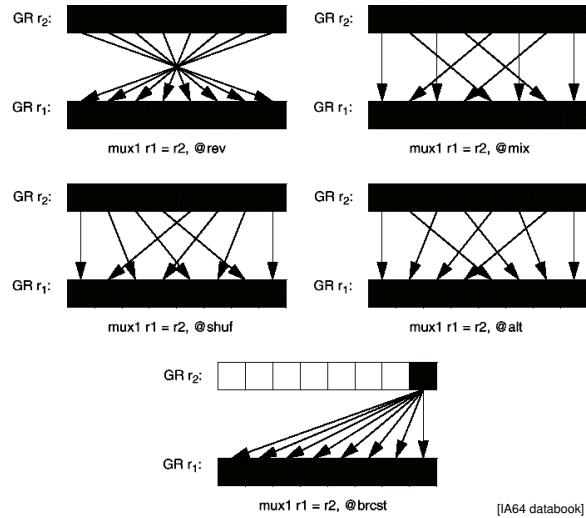
- Anordnung optimal für SIMD
- aber im Speicher "verstreut"

=> Hybrid SoA - SIMD

```
struct
{
    float A[8], B[8], C[8];
} Hybrid_data[125];
```

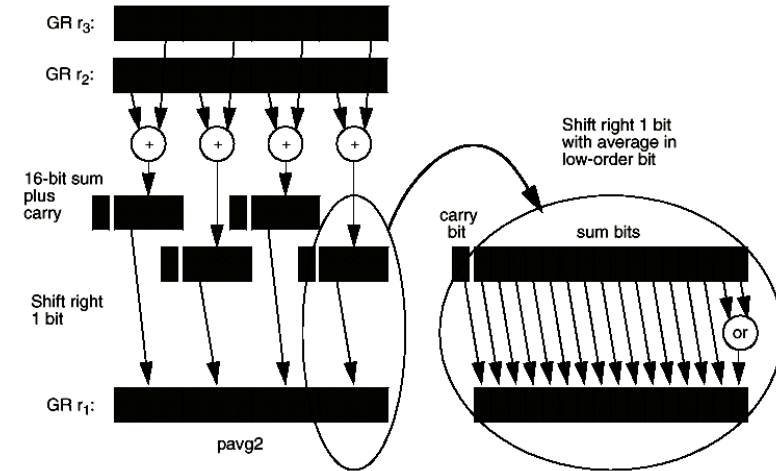


ISSE2: mux1-Befehl (IA64)



PC-Technologie | SS 2001 | 18.214

ISSE2: pavg2-Befehl (IA64)



PC-Technologie | SS 2001 | 18.214

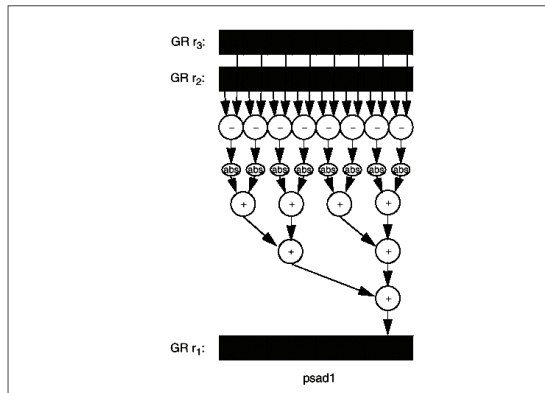
ISSE2: psad1-Befehl (IA64)

Parallel Sum of Absolute Difference

Format: (qp) psad1 r1 = r2, r3

Description: The unsigned 8-bit elements of GR r2 are subtracted from the unsigned 8-bit elements of GR r3. The absolute value of each difference is accumulated across the elements and placed in GR r1.

Figure 7-36. Parallel Sum of Absolute Difference Example



PC-Technologie | SS 2001 | 18.214

ISSE: FIR-Filter

Nutzen von MMX/ISSE für Filter?

- MMX und ISSE für 16-bit Integer
- ISSE für 32-bit Gleitkommawerte
- maximal vierfache Leistung gegenüber skalarem Code

aber:

- erfordert Operanden-Alignment (16-Byte Grenzen)
- z.B. durch Duplizierung der Daten/Koeffizienten-Arrays
- Arraygrößen Vielfache von 4
- Multiplikation parallel, aber Akkumulation schwierig

=> siehe Intel Appnote
"32-bit FP FIR Filter implemented using SSE"

PC-Technologie | SS 2001 | 18.214