Intel MMX, SSE, SSE2, SSE3/SSSE3/SSE4 Architectures

Baha Guclu Dundar
SALUC *Lab*
Computer Science and Engineering Department
University of Connecticut

Slides 1-33 are modified from
Computer Organization and Assembly Languages Course
By Yung-Yu Chuang

Overview

- SIMD
- MMX architectures
- MMX instructions
- examples
- SSE/SSE2/SSE3

- SIMD instructions are probably the best place to use assembly since compilers usually do not do a good job on using these instructions

2

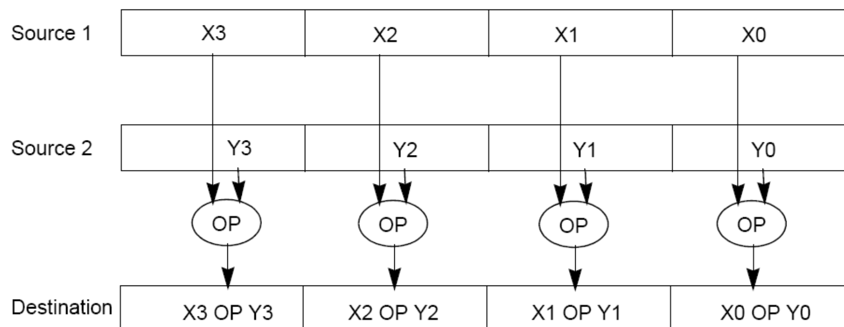
Performance boost

- Increasing clock rate is not fast enough for boosting performance
- Architecture improvements (such as pipeline/cache/SIMD) are more significant
- Intel analyzed multimedia applications and found they share the following characteristics:
  - Small native data types (8-bit pixel, 16-bit audio)
  - Recurring operations
  - Inherent parallelism

3

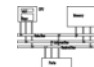
SIMD

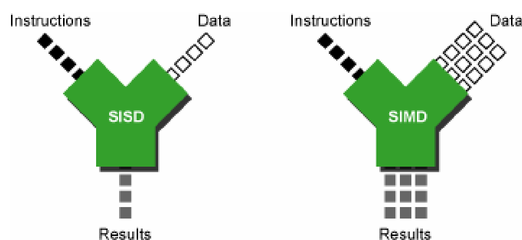
- SIMD (single instruction multiple data) architecture performs the same operation on multiple data elements in parallel
- **`PADDW MM0, MM1`**



4

SISD/SIMD





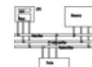
5

IA-32 SIMD development

- MMX (Multimedia Extension) was introduced in 1996 (Pentium with MMX and Pentium II).
- SSE (Streaming SIMD Extension) was introduced in 1999 with Pentium III.
- SSE2 was introduced with Pentium 4 in 2001.
- SSE3 was introduced in 2004 with Pentium 4 supporting hyper-threading technology. SSE3 adds 13 more instructions.

6

MMX



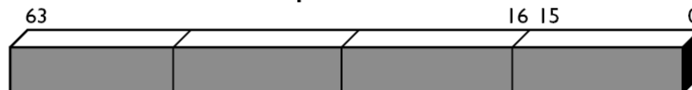
- After analyzing a lot of existing applications such as graphics, MPEG, music, speech recognition, game, image processing, they found that many multimedia algorithms execute the same instructions on many pieces of data in a large data set.
- Typical elements are small, 8 bits for pixels, 16 bits for audio, 32 bits for graphics and general computing.
- New data type: 64-bit packed data type. Why 64 bits?
  - Good enough
  - Practical

7

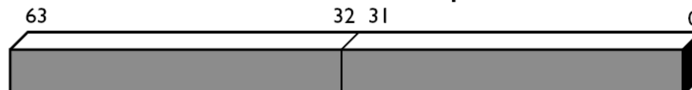
MMX data types

Packed Byte: 8 bytes packed into 64 bits

Packed Word: 4 words packed into 64 bits



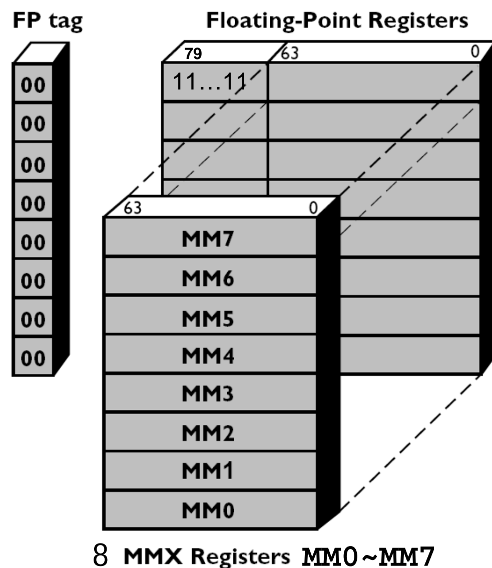
Packed Doubleword: 2 doublewords packed into 64 bits



Packed Quadword: One 64-bit quantity

8

MMX integration into IA



NaN or infinity as real because bits 79-64 are ones.

Even if MMX registers are 64-bit, they don't extend Pentium to a 64-bit CPU since only logic instructions are provided for 64-bit data.

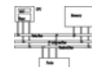
9

Compatibility

- To be fully compatible with existing IA, no new mode or state was created. Hence, for context switching, no extra state needs to be saved.
- To reach the goal, MMX is hidden behind FPU. When floating-point state is saved or restored, MMX is saved or restored.
- It allows existing OS to perform context switching on the processes executing MMX instruction without be aware of MMX.
- However, it means MMX and FPU can not be used at the same time. Big overhead to switch.

10

Compatibility



- Although Intel defenses their decision on aliasing MMX to FPU for compatibility. It is actually a bad decision. OS can just provide a service pack or get updated.
- It is why Intel introduced SSE later without any aliasing

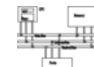
11

MMX instructions

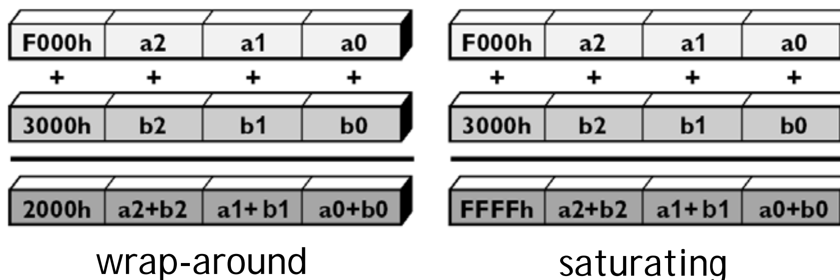
- 57 MMX instructions are defined to perform the parallel operations on multiple data elements packed into 64-bit data types.
- These include **`add`**, **`subtract`**, **`multiply`**, **`compare`**, and **`shift`**, **`data conversion`**, **`64-bit data move`**, **`64-bit logical operation`** and **`multiply-add`** for multiply-accumulate operations.
- All instructions except for data move use MMX registers as operands.
- Most complete support for 16-bit operations.

12

Saturation arithmetic



- Useful in graphics applications.
- When an operation overflows or underflows, the result becomes the largest or smallest possible representable number.
- Two types: signed and unsigned saturation



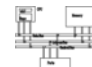
13

MMX instructions

Category		Wraparound	Signed Saturation	Unsigned Saturation
Arithmetic	Addition	PADDB, PADDW, PADDD	PADDSB, PADDSW	PADDUSB, PADDUSW
	Subtraction	PSUBB, PSUBW, PSUBD	PSUBSB, PSUBSW	PSUBUSB, PSUBUSW
	Multiplication	PMULL, PMULH		
	Multiply and Add	PMADD		
Comparison	Compare for Equal	PCMPEQB, PCMPEQW, PCMPEQD		
	Compare for Greater Than	PCMPGTPB, PCMPGTPW, PCMPGTPD		
Conversion	Pack		PACKSSWB, PACKSSDW	PACKUSWB
Unpack	Unpack High	PUNPCKHBW, PUNPCKHWD, PUNPCKHDQ		
	Unpack Low	PUNPCKLBW, PUNPCKLWD, PUNPCKLDQ		

14

MMX instructions



		Packed	Full Quadword
Logical	And And Not Or Exclusive OR		PAND PANDN POR PXOR
Shift	Shift Left Logical Shift Right Logical Shift Right Arithmetic	PSLLW, PSLLD PSRLW, PSRLD PSRAW, PSRAD	PSLLQ PSRLQ
		Doubleword Transfers	**Quadword Transfers**
Data Transfer	Register to Register Load from Memory Store to Memory	MOVD MOVD MOVD	MOVQ MOVQ MOVQ
Empty MMX State		EMMS	

Call it before you switch to FPU from MMX;
Expensive operation

15

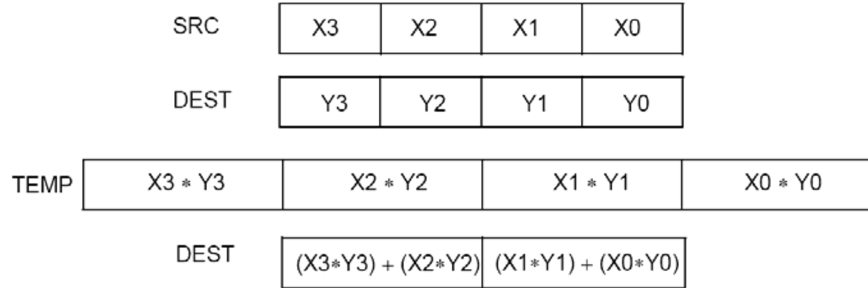
Arithmetic

- **`PADDB/PADDW/PADDD`**: add two packed numbers, no EFLAGS is set, ensure overflow never occurs by yourself
- Multiplication: two steps
- **`PMULLW`**: multiplies four words and stores the four lo words of the four double word results
- **`PMULHW/PMULHUW`**: multiplies four words and stores the four hi words of the four double word results. **`PMULHUW`** for unsigned.

16

Arithmetic

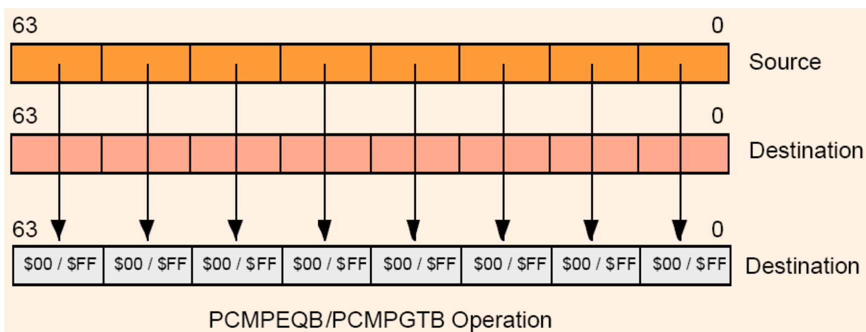
- **PMADDWD**



17

Comparison

- No CFLAGS, how many flags will you need? Results are stored in destination.
- EQ/GT, no LT

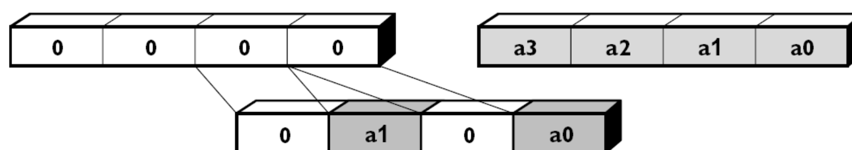


18

Change data types

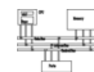
- Pack: converts a larger data type to the next smaller data type.
- Unpack: takes two operands and interleave them. It can be used for expand data type for immediate calculation.

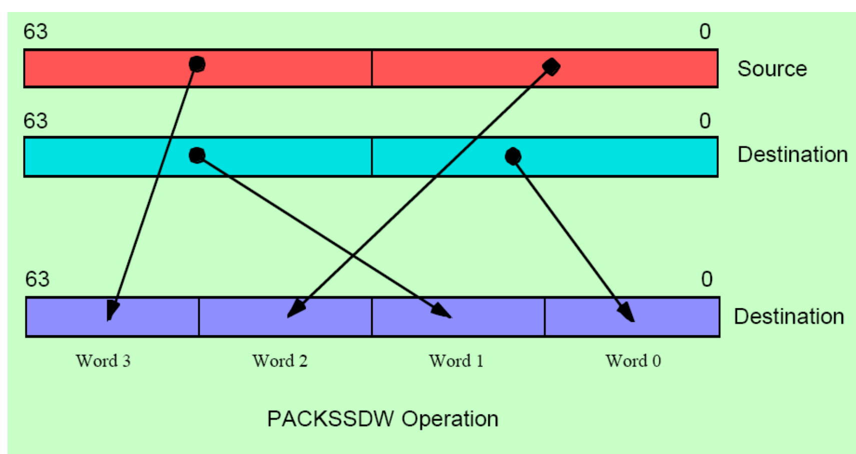
Unpack low-order words into doublewords



19

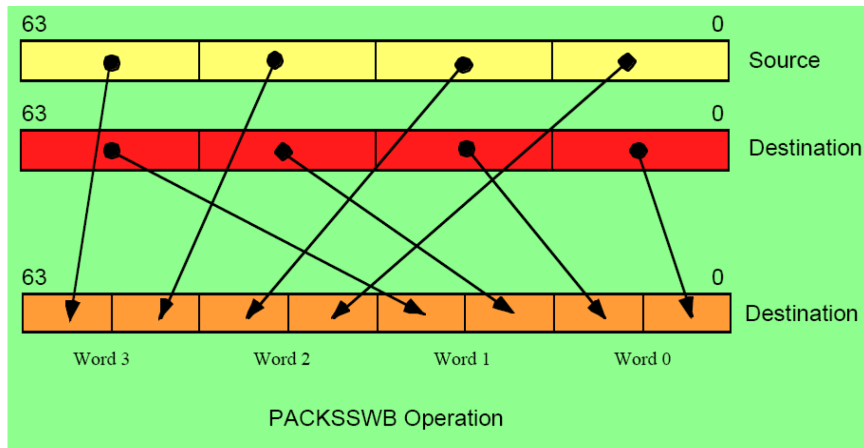
Pack with signed saturation





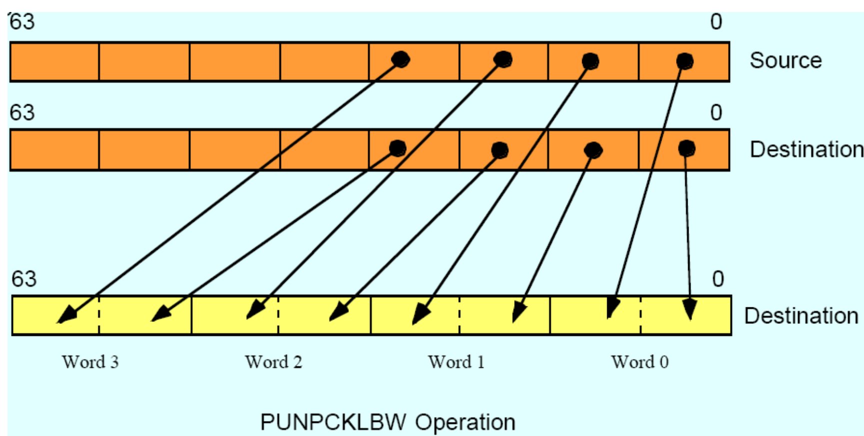
20

Pack with signed saturation



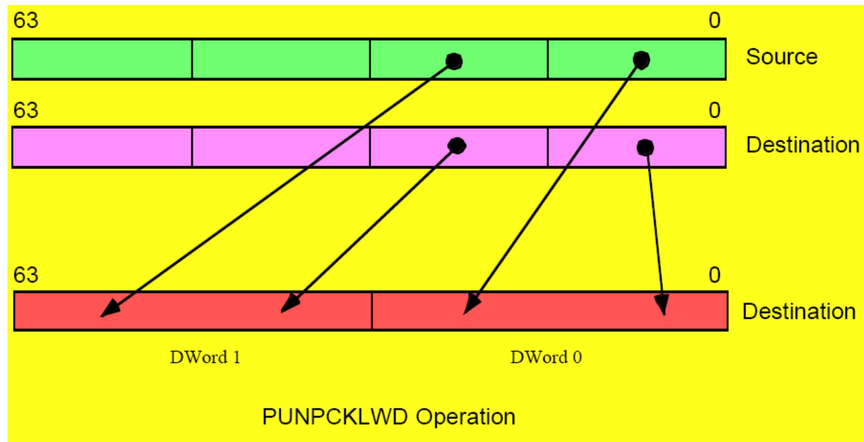
21

Unpack low portion



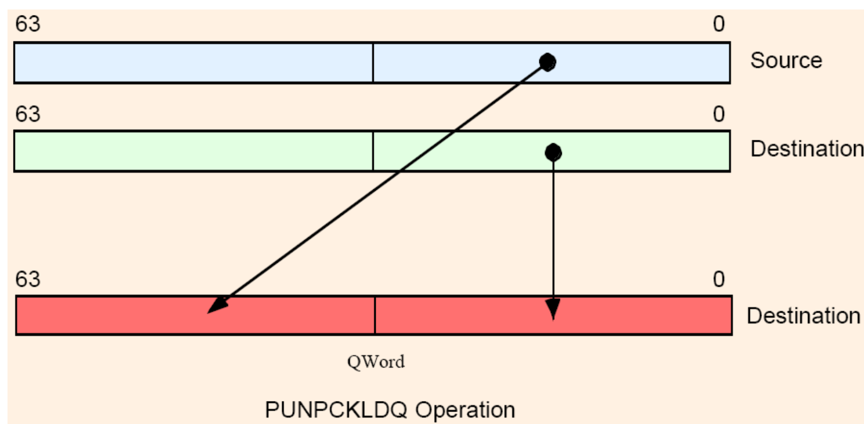
22

Unpack low portion



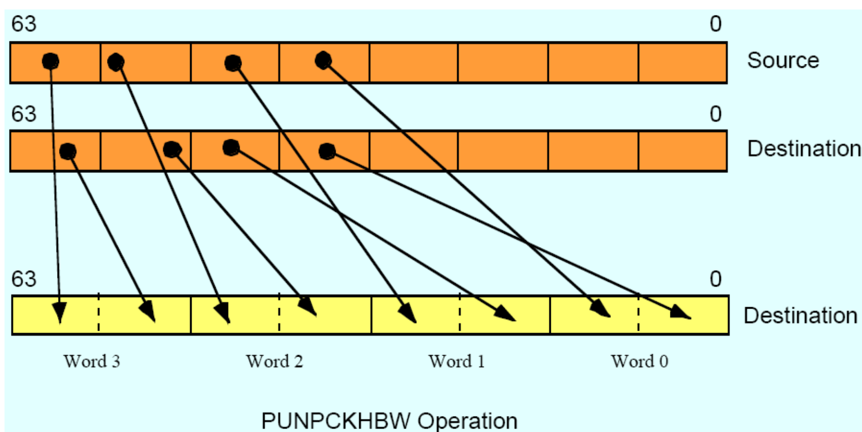
23

Unpack low portion



24

Unpack high portion



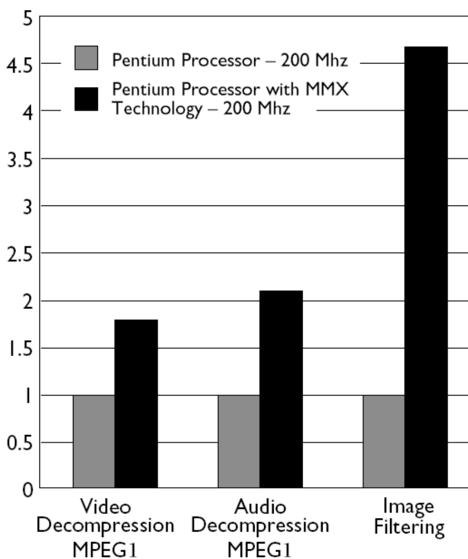
25

Performance boost (data from 1996)

Benchmark kernels: FFT, FIR, vector dot-product, IDCT, motion compensation.

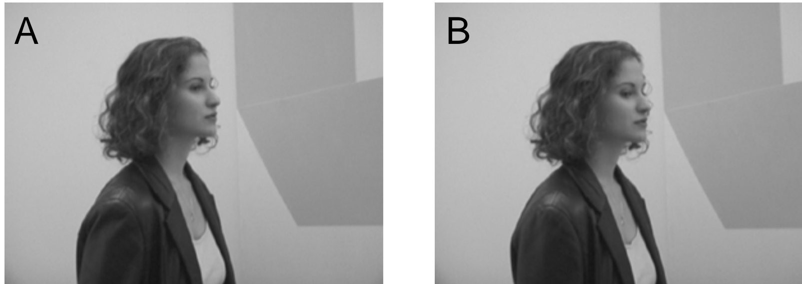
65% performance gain

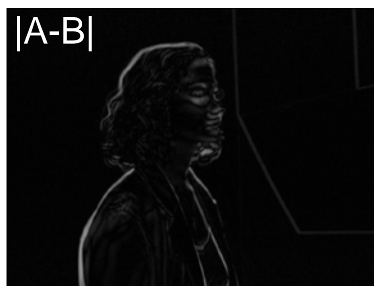
Lower the cost of multimedia programs by removing the need of specialized DSP chips



26

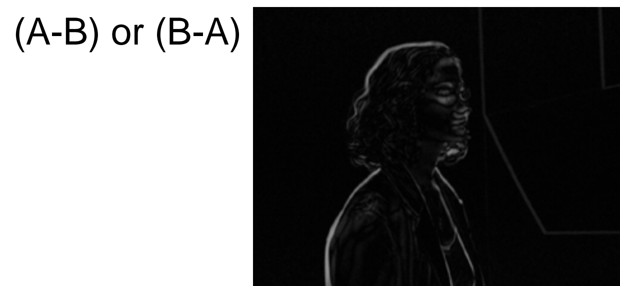
Application: frame difference





27

Application: frame difference



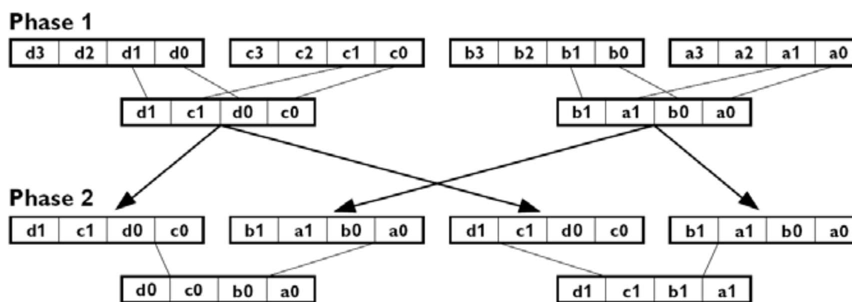
28

Application: frame difference

```
MOVQ      mm1, A //move 8 pixels of image A
MOVQ      mm2, B //move 8 pixels of image B
MOVQ      mm3, mm1  // mm3=A
PSUBSB    mm1, mm2  // mm1=A-B
PSUBSB    mm2, mm3  // mm2=B-A
POR       mm1, mm2  // mm1=|A-B|
```

29

Application: matrix transport



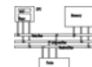
Note: Repeat for the other rows to generate ([d_3, c_3, b_3, a_3] and [d_2, c_2, b_2, a_2]).

MMX code sequence operation:

```
movq        mm1, row1    ; load pixels from first row of matrix
movq        mm2, row2    ; load pixels from second row of matrix
movq        mm3, row3    ; load pixels from third row of matrix
movq        mm4, row4    ; load pixels from fourth row of matrix
punpcklwd   mm1, mm2     ; unpack low order words of rows 1 & 2, mm 1 = [b1, a1, b0, a0]
punpcklwd   mm3, mm4     ; unpack low order words of rows 3 & 4, mm3 = [d1, c1, d0, c0]
movq        mm5, mm1     ; copy mm1 to mm5
punpckldq   mm1, mm3     ; unpack low order doublewords -> mm2 = [d0, c0, b0, a0]
punpckhdq   mm5, mm3     ; unpack high order doublewords -> mm5 = [d1, c1, b1, a1]
```

30

Application: matrix transport



```
char M1[4][8];// matrix to be transposed
char M2[8][4];// transposed matrix
int n=0;
for (int i=0;i<4;i++)
  for (int j=0;j<8;j++)
    { M1[i][j]=n; n++; }
__asm{
//move the 4 rows of M1 into MMX registers
movq mm1,M1
movq mm2,M1+8
movq mm3,M1+16
movq mm4,M1+24
```

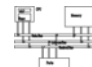
31

Application: matrix transport

```
//generate rows 1 to 4 of M2
punpcklbw mm1, mm2
punpcklbw mm3, mm4
movq mm0, mm1
punpcklwd mm1, mm3 //mm1 has row 2 & row 1
punpckhwd mm0, mm3 //mm0 has row 4 & row 3
movq M2, mm1
movq M2+8, mm0
```

32

Application: matrix transport



```
//generate rows 5 to 8 of M2
movq mm1, M1 //get row 1 of M1
movq mm3, M1+16 //get row 3 of M1
punpckhbw mm1, mm2
punpckhbw mm3, mm4
movq mm0, mm1
punpcklwd mm1, mm3 //mm1 has row 6 & row 5
punpckhwd mm0, mm3 //mm0 has row 8 & row 7
//save results to M2
movq M2+16, mm1
movq M2+24, mm0
emms
} //end
```

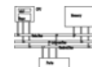
33

SSE

- Adds eight 128-bit registers
- Allows SIMD operations on packed single-precision floating-point numbers
- Most SSE instructions require 16-aligned addresses
- Allows 70 new instructions

34

Advantages of SSE

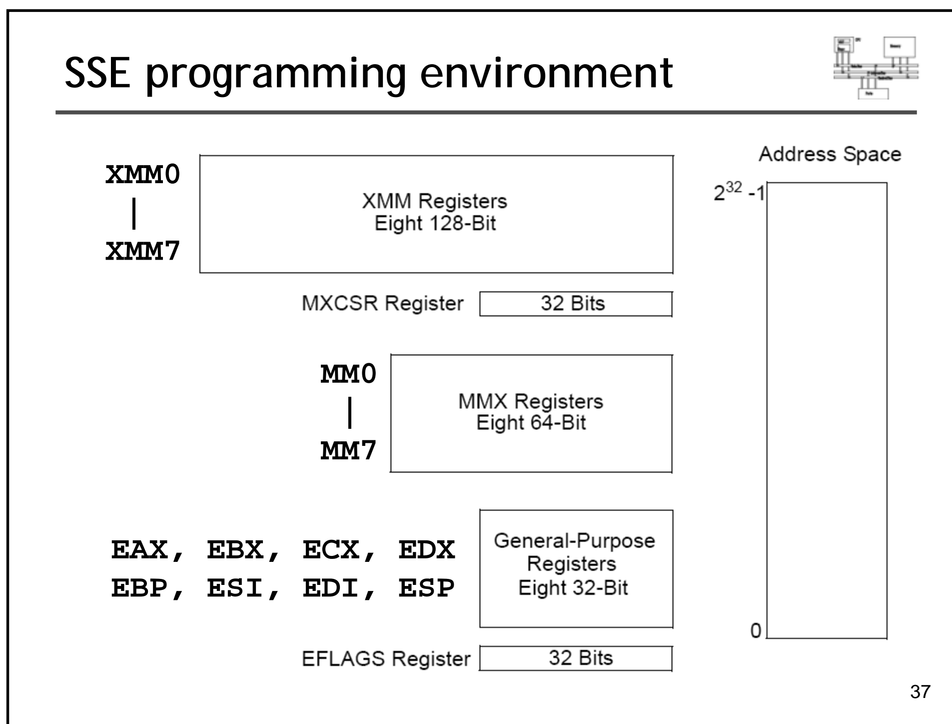


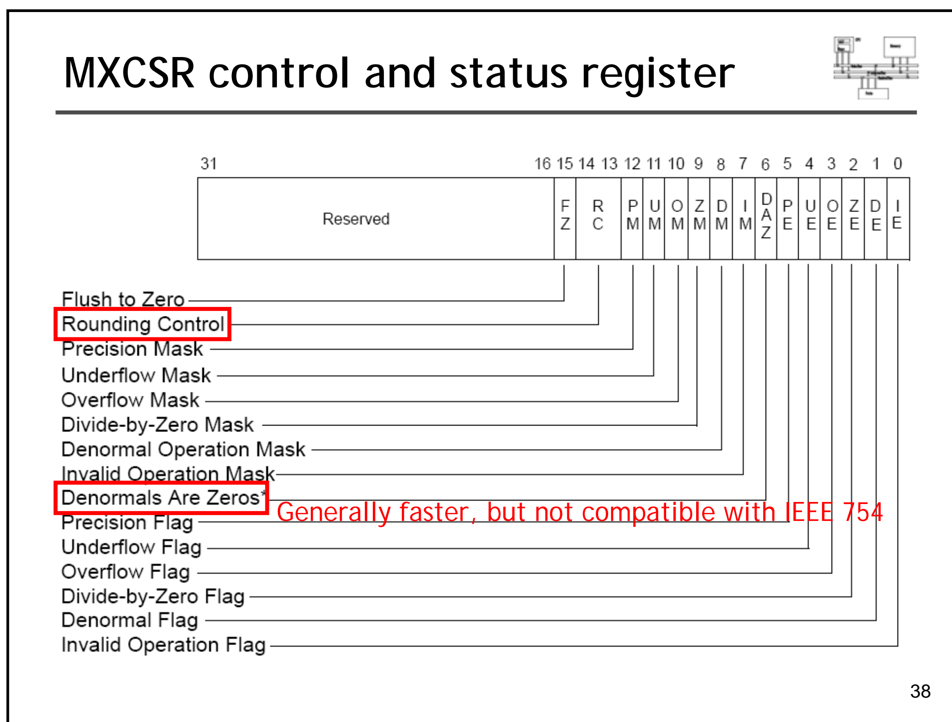
In MMX

- An application cannot execute MMX instructions and perform floating-point operations simultaneously.
- A large number of processor clock cycles are needed to change the state of executing MMX instructions to the state of executing FP operations and vice versa.

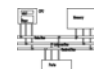
SSE features

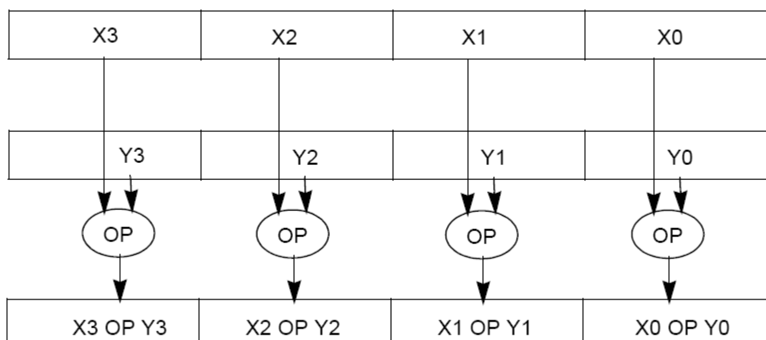
- Add eight 128-bit data registers (XMM registers) in non-64-bit mode.
- 32-bit MXCSR register (control and status)
- Add a new data type: 128-bit packed single-precision floating-point (4 FP numbers.)
- Instruction to perform SIMD operations on 128-bit packed single-precision FP and additional 64-bit SIMD integer operations.
- Instructions that explicitly prefetch data, control data cacheability and ordering of store


SSE programming environment
XMM0
XMM7
XMM Registers
Eight 128-Bit
MXCSR Register
32 Bits
MM0
MM7
MMX Registers
Eight 64-Bit
EAX, EBX, ECX, EDX
EBP, ESI, EDI, ESP
General-Purpose
Registers
Eight 32-Bit
EFLAGS Register
32 Bits
Address Space
2^32 -1
0
37


MXCSR control and status register
31
16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Reserved
FZ
RC
PM
UM
OM
ZM
DM
IM
DAZ
PE
UE
OE
ZE
DE
IE
Flush to Zero
Rounding Control
Precision Mask
Underflow Mask
Overflow Mask
Divide-by-Zero Mask
Denormal Operation Mask
Invalid Operation Mask
Denormals Are Zeros*
Generally faster, but not compatible with IEEE 754
Precision Flag
Underflow Flag
Overflow Flag
Divide-by-Zero Flag
Denormal Flag
Invalid Operation Flag
38

SSE packed FP operation

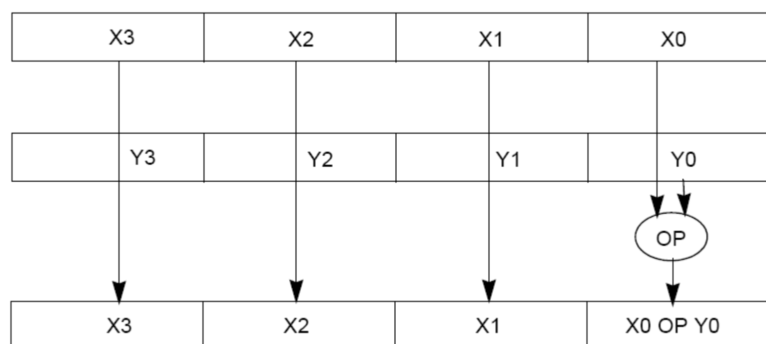




- **`ADDPS/SUBPS`**: packed single-precision FP

39

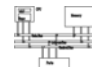
SSE scalar FP operation



- **`ADDSS/SUBSS`**: scalar single-precision FP
  used as FPU?

40

SSE Instruction Set



Floating point instructions

- Memory-to-Register / Register-to-Memory / Register-to-Register data movement
  - Scalar – MOVSS
  - Packed – MOVAPS, MOVUPS, MOVLPS, MOVHPS, MOVLHPS, MOVHLPS
- Arithmetic
  - Scalar – ADDSS, SUBSS, MULSS, DIVSS, RCPSS, SQRTSS, MAXSS, MINSS, RSQRTSS
  - Packed – ADDPS, SUBPS, MULPS, DIVPS, RCPPS, SQRTPS, MAXPS, MINPS, RSQRTPS
- Compare
  - Scalar – CMPSS, COMISS, UCOMISS
  - Packed – CMPPS
- Data shuffle and unpacking
  - Packed – SHUFPS, UNPCKHPS, UNPCKLPS
- Data-type conversion
  - Scalar – CVTSI2SS, CVTSS2SI, CVTTSS2SI
  - Packed – CVTPI2PS, CVTPS2PI, CVTTPS2PI
- Bitwise logical operations
  - Packed – ANDPS, ORPS, XORPS, ANDNPS

41

SSE Instruction Set

Integer instructions

- Arithmetic
  - PMULHUW, PSADBW, PAVGB, PAVGW, PMAXUB, PMINUB, PMAXSW, PMINSW
- Data movement
  - PEXTRW, PINSRW
- Other
  - PMOVMSKB, PSHUFW

Other instructions

- MXCSR management
  - LDMXCSR, STMXCSR
- Cache and Memory management
  - MOVNTQ, MOVNTPS, MASKMOVQ, PREFETCH0, PREFETCH1, PREFETCH2, PREFETCHNTA, SFENCE

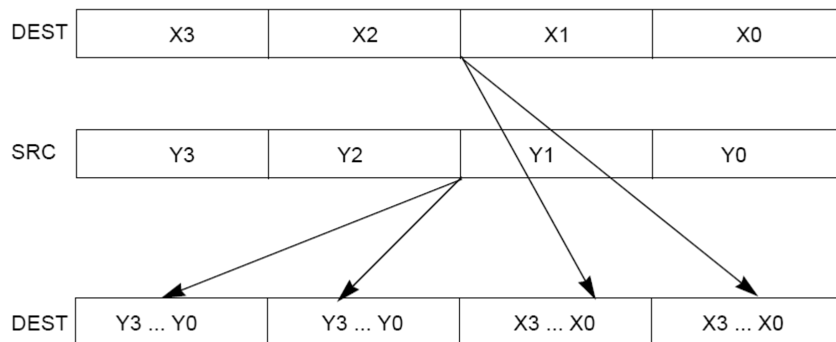
42

SSE Packed Shuffle (**SHUFPS**)

SHUFPS xmm1, xmm2, imm8

Select[1..0] decides which DW of DEST to be copied to the 1st DW of DEST

...



43

SSE Shuffle (**SHUFPS**)

```
CASE (SELECT[1:0]) OF
   0:  DEST[31:0] ← DEST[31:0];
   1:  DEST[31:0] ← DEST[63:32];
   2:  DEST[31:0] ← DEST[95:64];
   3:  DEST[31:0] ← DEST[127:96];
ESAC;
CASE (SELECT[3:2]) OF
   0:  DEST[63:32] ← DEST[31:0];
   1:  DEST[63:32] ← DEST[63:32];
   2:  DEST[63:32] ← DEST[95:64];
   3:  DEST[63:32] ← DEST[127:96];
ESAC;
CASE (SELECT[5:4]) OF
   0:  DEST[95:64] ← SRC[31:0];
   1:  DEST[95:64] ← SRC[63:32];
   2:  DEST[95:64] ← SRC[95:64];
   3:  DEST[95:64] ← SRC[127:96];
ESAC;
CASE (SELECT[7:6]) OF
   0:  DEST[127:96] ← SRC[31:0];
   1:  DEST[127:96] ← SRC[63:32];
   2:  DEST[127:96] ← SRC[95:64];
   3:  DEST[127:96] ← SRC[127:96];
ESAC;
```

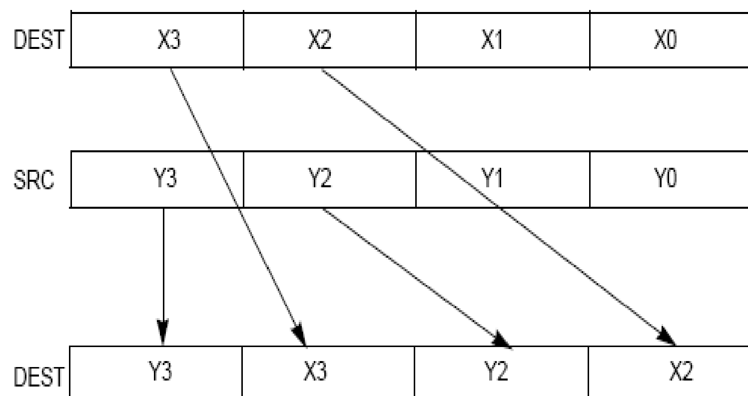
44

Example (cross product)

```
/* cross */
__m128 _mm_cross_ps( __m128 a , __m128 b ) {
  __m128 ea , eb;
  // set to a[1][2][0][3] , b[2][0][1][3]
  ea = _mm_shuffle_ps( a, a, _MM_SHUFFLE(3,0,2,1) );
  eb = _mm_shuffle_ps( b, b, _MM_SHUFFLE(3,1,0,2) );
  // multiply
  __m128 xa = _mm_mul_ps( ea , eb );
  // set to a[2][0][1][3] , b[1][2][0][3]
  a = _mm_shuffle_ps( a, a, _MM_SHUFFLE(3,1,0,2) );
  b = _mm_shuffle_ps( b, b, _MM_SHUFFLE(3,0,2,1) );
  // multiply
  __m128 xb = _mm_mul_ps( a , b );
  // subtract
  return _mm_sub_ps( xa , xb );
}
```

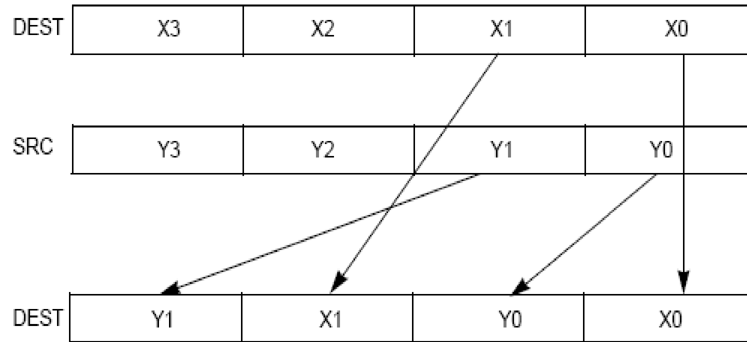
45

High Unpack and interleave Shuffle (**UNPCKHPS**)



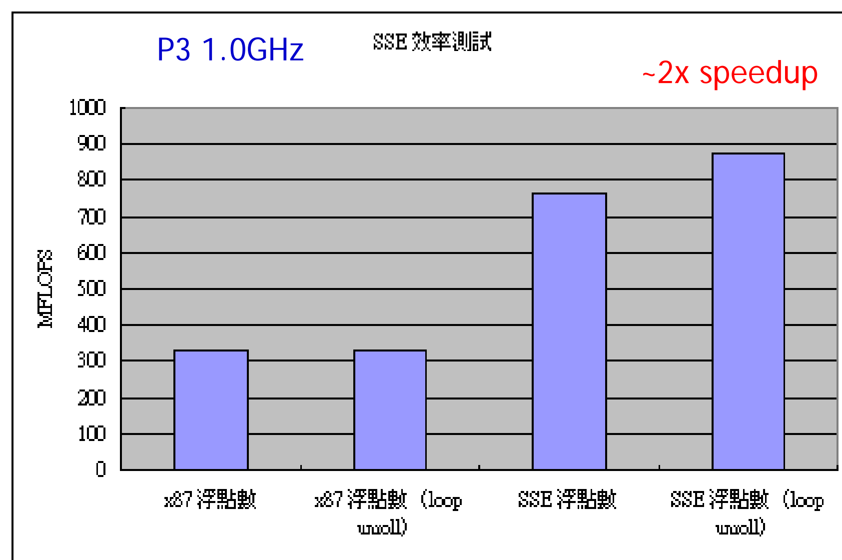
46

Low Unpack and interleave Shuffle (UNPCKHPS)



47

SSE examples (1,024 FP additions)



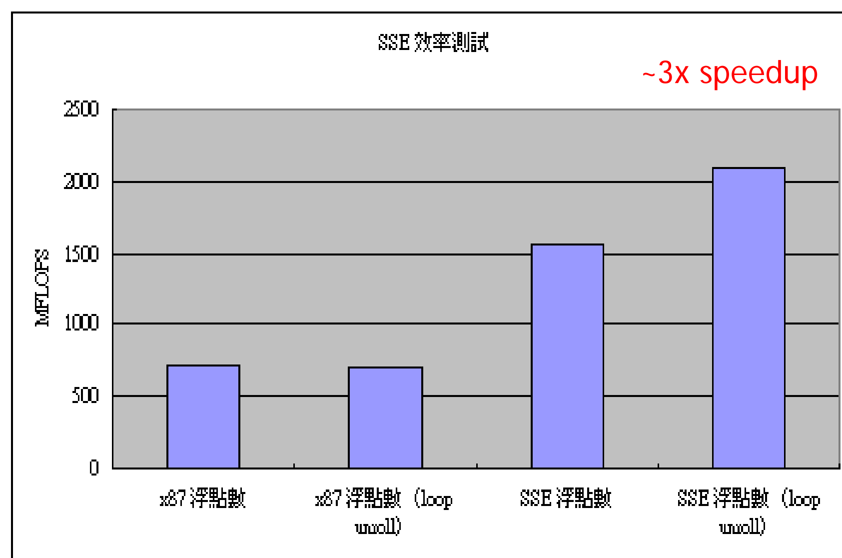
48

Inner product

```
__m128 x1 = _mm_load_ps(vec1_x);
__m128 y1 = _mm_load_ps(vec1_y);
__m128 z1 = _mm_load_ps(vec1_z);
__m128 x2 = _mm_load_ps(vec2_x);
__m128 y2 = _mm_load_ps(vec2_y);
__m128 z2 = _mm_load_ps(vec2_z);
__m128 t1 = _mm_mul_ps(x1, x2);
__m128 t2 = _mm_mul_ps(y1, y2);
t1 = _mm_add_ps(t1, t2);
t2 = _mm_mul_ps(z1, z2);
t1 = _mm_add_ps(t1, t2);
_mm_store_ps(output, t1);
```

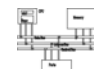
49

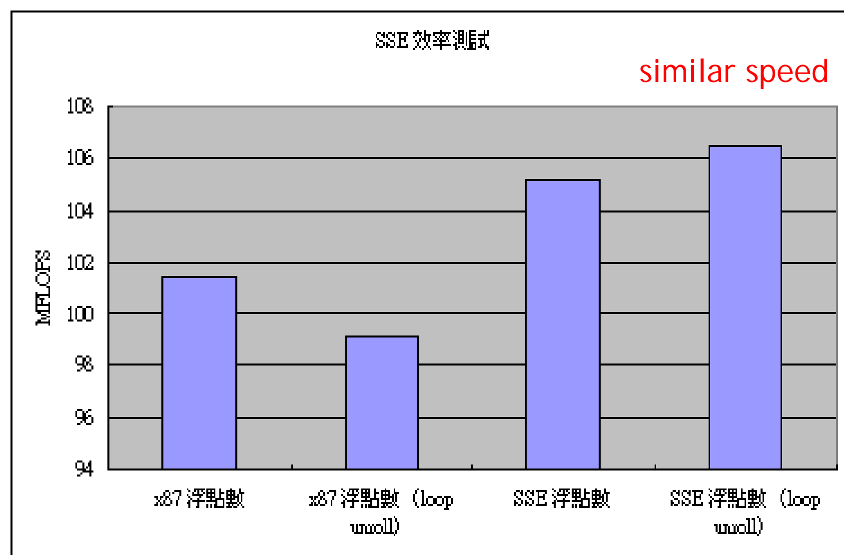
Inner product (1,024 3D vectors)



50

Inner product (102,400 3D vectors)





51

SSE2

- Introduced into the IA-32 architecture in the Pentium 4 and Intel Xeon processors in 2001.
- Allowing advanced graphics such as 3-D graphics, video decoding/encoding, speech recognition
- AMD didn't support SSE2 until 2003, with their Opteron and Athlon64 processors

52

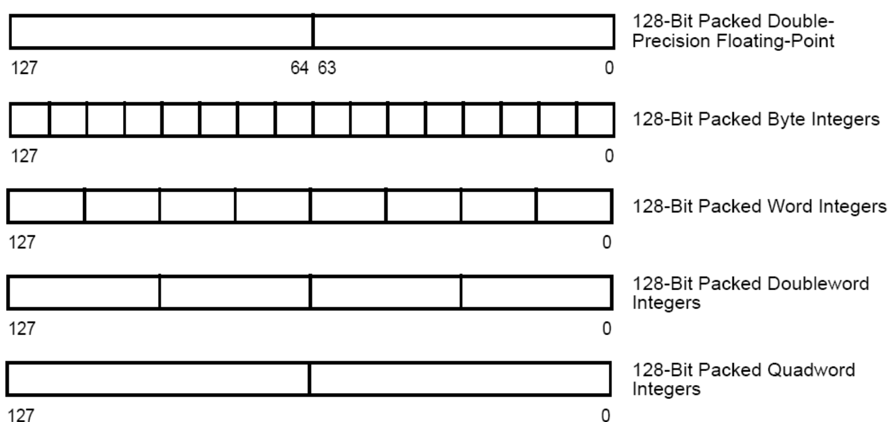
What is new in SSE2?

- Provides ability to perform SIMD operations on 128-bit double-precision FP.
- Provides greater throughput by operating on 128-bit packed integers, useful for RC5 and RSA. XMM registers are also used for 128-bit packed integer data.
- Offers more flexibility in big numbers.
- 144 new instructions

53

SSE2 features

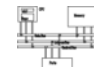
- Add data types and instructions for them

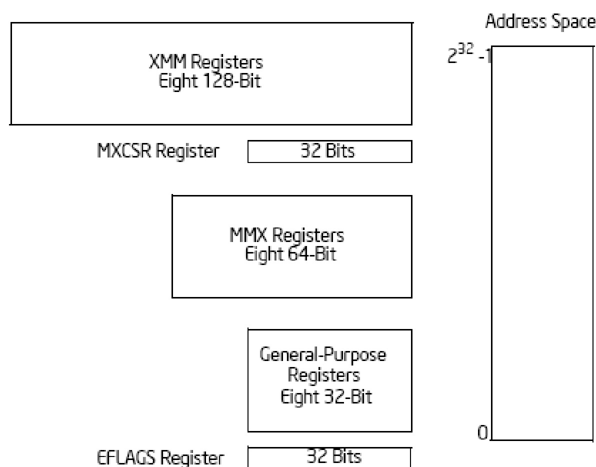


- Programming environment unchanged (also packed and scalar)

54

SSE2 Programming Environment





55

SSE2 Instructions

ARITHMETIC:

addpd - Adds 2 64bit doubles.
addsd - Adds bottom 64bit doubles.
subpd - Subtracts 2 64bit doubles.
subsd - Subtracts bottom 64bit doubles.
mulpd - Multiplies 2 64bit doubles.
mulsd - Multiplies bottom 64bit doubles.
divpd - Divides 2 64bit doubles.
divsd - Divides bottom 64bit doubles.
maxpd - Gets largest of 2 64bit doubles for 2 sets.
maxsd - Gets largest of 2 64bit doubles to bottom set.
minpd - Gets smallest of 2 64bit doubles for 2 sets.
minsd - Gets smallest of 2 64bit values for bottom set.
paddb - Adds 16 8bit integers.
paddw - Adds 8 16bit integers.
paddd - Adds 4 32bit integers.
paddq - Adds 2 64bit integers.
paddsb - Adds 16 8bit integers with saturation.
paddsw - Adds 8 16bit integers using saturation.
paddusb - Adds 16 8bit unsigned integers using saturation.
paddusw - Adds 8 16bit unsigned integers using saturation.
psubb - Subtracts 16 8bit integers.
psubw - Subtracts 8 16bit integers.
psubd - Subtracts 4 32bit integers.
psubq - Subtracts 2 64bit integers.
psubsb - Subtracts 16 8bit integers using saturation.
psubsw - Subtracts 8 16bit integers using saturation.
psubusb - Subtracts 16 8bit unsigned integers using saturation.
psubusw - Subtracts 8 16bit unsigned integers using saturation.
pmaddwd - Multiplies 16bit integers into 32bit results and adds results.
pmulhw - Multiplies 16bit integers and returns the high 16bits of the result.
pmullw - Multiplies 16bit integers and returns the low 16bits of the result.
pmuludq - Multiplies 2 32bit pairs and stores 2 64bit results.
rcpps - Approximates the reciprocal of 4 32bit singles.
rcpss - Approximates the reciprocal of bottom 32bit single.
sqrtpd - Returns square root of 2 64bit doubles.
sqrtsd - Returns square root of bottom 64bit double.

56

SSE2 Instructions

Logic:
- andnpd - Logically NOT ANDs 2 64bit doubles.
- andnps - Logically NOT ANDs 4 32bit singles.
- andpd - Logically ANDs 2 64bit doubles.
- pand - Logically ANDs 2 128bit registers.
- pandn - Logically Inverts the first 128bit operand and ANDs with the second.
- por - Logically ORs 2 128bit registers.
- pslldq - Logically left shifts 1 128bit value.
- psllq - Logically left shifts 2 64bit values.
- pslld - Logically left shifts 4 32bit values.
- psllw - Logically left shifts 8 16bit values.
- psrad - Arithmetically right shifts 4 32bit values.
- psraw - Arithmetically right shifts 8 16bit values.
- psrldq - Logically right shifts 1 128bit values.
- psrlq - Logically right shifts 2 64bit values.
- psrld - Logically right shifts 4 32bit values.
- psrlw - Logically right shifts 8 16bit values.
- pxor - Logically XORs 2 128bit registers.
- orpd - Logically ORs 2 64bit doubles.
- xorpd - Logically XORs 2 64bit doubles.

57

SSE2 Instructions

Compare:
- cmppd - Compares 2 pairs of 64bit doubles.
- cmpsd - Compares bottom 64bit doubles.
- comisd - Compares bottom 64bit doubles and stores result in EFLAGS.
- ucomisd - Compares bottom 64bit doubles and stores result in EFLAGS. (QNaNs don't throw exceptions with ucomisd, unlike comisd.
- pcmpxxb - Compares 16 8bit integers.
- pcmpxxw - Compares 8 16bit integers.
- pcmpxxd - Compares 4 32bit integers.
- Compare Codes (the xx parts above):
- eq - Equal to.
- lt - Less than.
- le - Less than or equal to.
- ne - Not equal.
- nlt - Not less than.
- nle - Not less than or equal to.
- ord - Ordered.
- unord - Unordered.

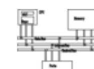
58

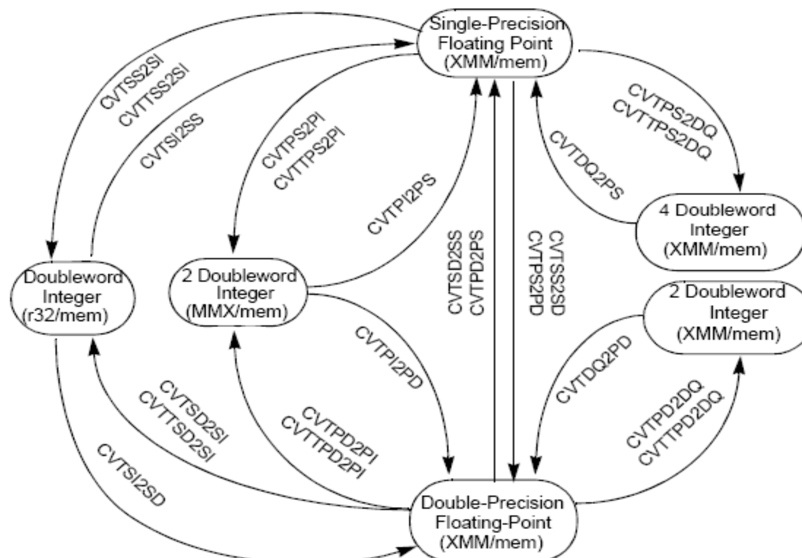
SSE2 Instructions

Conversion:
- cvtdq2pd - Converts 2 32bit integers into 2 64bit doubles.
- cvtdq2ps - Converts 4 32bit integers into 4 32bit singles.
- cvtpd2pi - Converts 2 64bit doubles into 2 32bit integers in an MMX register.
- cvtpd2dq - Converts 2 64bit doubles into 2 32bit integers in the bottom of an XMM register.
- cvtpd2ps - Converts 2 64bit doubles into 2 32bit singles in the bottom of an XMM register.
- cvtpi2pd - Converts 2 32bit integers into 2 32bit singles in the bottom of an XMM register.
- cvtps2dq - Converts 4 32bit singles into 4 32bit integers.
- cvtps2pd - Converts 2 32bit singles into 2 64bit doubles.
- cvtsd2si - Converts 1 64bit double to a 32bit integer in a GPR.
- cvtsd2ss - Converts bottom 64bit double to a bottom 32bit single. Tops are unchanged.
- cvtsi2sd - Converts a 32bit integer to the bottom 64bit double.
- cvtsi2ss - Converts a 32bit integer to the bottom 32bit single.
- cvtss2sd - Converts bottom 32bit single to bottom 64bit double.
- cvtss2si - Converts bottom 32bit single to a 32bit integer in a GPR.
- cvttpd2pi - Converts 2 64bit doubles to 2 32bit integers using truncation into an MMX register.
- cvttpd2dq - Converts 2 64bit doubles to 2 32bit integers using truncation.
- cvttps2dq - Converts 4 32bit singles to 4 32bit integers using truncation.
- cvttps2pi - Converts 2 32bit singles to 2 32bit integers using truncation into an MMX register.
- cvttsd2si - Converts a 64bit double to a 32bit integer using truncation into a GPR.
- cvttss2si - Converts a 32bit single to a 32bit integer using truncation into a GPR.

59

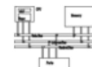
SSE2 Instructions





60

SSE2 Instructions



Load/Store:
(is "minimize cache pollution" the same as "without using cache"??)
movq - Moves a 64bit value, clearing the top 64bits of an XMM register.
movsd - Moves a 64bit double, leaving tops unchanged if move is between two XMMregisters.
movapd - Moves 2 aligned 64bit doubles.
movupd - Moves 2 unaligned 64bit doubles.
movhpd - Moves top 64bit value to or from an XMM register.
movlpd - Moves bottom 64bit value to or from an XMM register.
movdq2q - Moves bottom 64bit value into an MMX register.
movq2dq - Moves an MMX register value to the bottom of an XMM register. Top is cleared to zero.
movntpd - Moves a 128bit value to memory without using the cache. NT is "Non Temporal."
movntdq - Moves a 128bit value to memory without using the cache.
movnti - Moves a 32bit value without using the cache.
maskmovdqu - Moves 16 bytes based on sign bits of another XMM register.
pmovmskb - Generates a 16bit Mask from the sign bits of each byte in an XMM register.

61

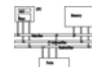
SSE2 Instructions

Shuffling:
pshufd - Shuffles 32bit values in a complex way.
pshufhw - Shuffles high 16bit values in a complex way.
pshuflw - Shuffles low 16bit values in a complex way.
unpckhpd - Unpacks and interleaves top 64bit doubles from 2 128bit sources into 1.
unpcklpd - Unpacks and interleaves bottom 64bit doubles from 2 128 bit sources into 1.
punpckhbw - Unpacks and interleaves top 8 8bit integers from 2 128bit sources into 1.
punpckhwd - Unpacks and interleaves top 4 16bit integers from 2 128bit sources into 1.
punpckhdq - Unpacks and interleaves top 2 32bit integers from 2 128bit sources into 1.
punpckhqdq - Unpacks and interleaces top 64bit integers from 2 128bit sources into 1.
punpcklbw - Unpacks and interleaves bottom 8 8bit integers from 2 128bit sources into 1.
punpcklwd - Unpacks and interleaves bottom 4 16bit integers from 2 128bit sources into 1.
punpckldq - Unpacks and interleaves bottom 2 32bit integers from 2 128bit sources into 1.
punpcklqdq - Unpacks and interleaces bottom 64bit integers from 2 128bit sources into 1.
packssdw - Packs 32bit integers to 16bit integers using saturation.
packsswb - Packs 16bit integers to 8bit integers using saturation.
packuswb - Packs 16bit integers to 8bit unsigned integers unsing saturation.

Cache Control:
clflush - Flushes a Cache Line from all levels of cache.
lfence - Guarantees that all memory loads issued before the lfence instruction are completed before anyloads after the lfence instruction.
mfence - Guarantees that all memory reads and writes issued before the mfence instruction are completed before any reads or writes after the mfence instruction.
pause - Pauses execution for a set amount of time.

62

SSE3/SSSE3/SSE4



- Introduced for Pentium 4 processor supporting Hyper-Threading Technology in 2004.
- The Intel Xeon processor 5100 series, Intel Core 2 processor families introduced Supplemental Streaming SIMD Extensions 3 (SSSE3)
- SSE4 are introduced in Intel processor generations built from 45nm process technology in 2006
- SSE3/SSSE3/SSE4 do not introduce new data types. XMM registers are used to operate on packed integer data, single precision floating-point data, or double-precision floating-point data.

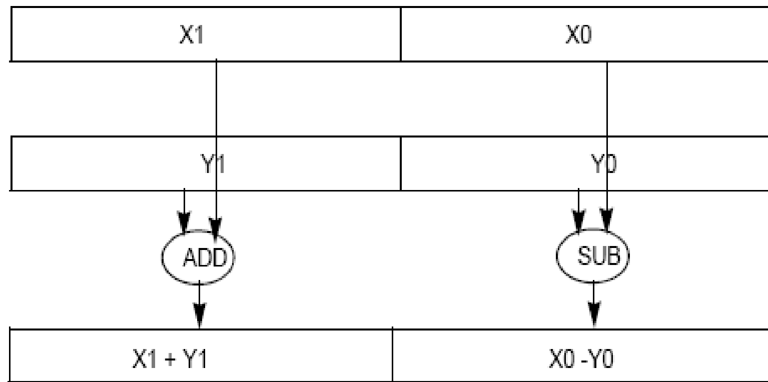
63

SSE3

- 13 new instructions
- Some instructions does horizontal operations (operating across a single register instead of down through multiple registers) and asymmetric processing
- Unaligned access instructions are new type of instructions.
- Process control instructions to boost performance with Intel's hyper-threading feature.
- AMD started to support SSE3 in 2005

64

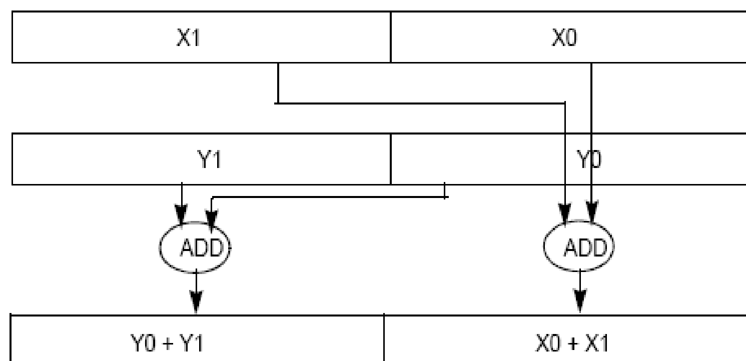
SSE3 Instructions



Asymmetric processing: ADDSUBPD

65

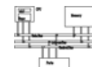
SSE3 Instructions



Horizontal data movement: HADDPD

66

SSE3 Instructions



Arithmetic:
- addsubpd - Adds the top two doubles and subtracts the bottom two.
- addsubps - Adds top singles and subtracts bottom singles.
- haddpd - Top double is sum of top and bottom, bottom double is sum of second operand's top and bottom.
- haddps - Horizontal addition of single-precision values.
- hsubpd - Horizontal subtraction of double-precision values.
- hsubps - Horizontal subtraction of single-precision values.

Load/Store:
- lddqu - Loads an unaligned 128bit value.
- movddup - Loads 64bits and duplicates it in the top and bottom halves of a 128bit register.
- movshdup - Duplicates the high singles into high and low singles.
- movsldup - Duplicates the low singles into high and low singles.
- fisttp - Converts a floating-point value to an integer using truncation.

Process Control:
- monitor - Sets up a region to monitor for activity.
- mwait - Waits until activity happens in a region specified by monitor.

67

SSSE3

- New 32 instructions designed for to accelerate a variety of multimedia and signal processing applications.
- Employs SIMD integer data.
- The operands of these instructions are packed integers of byte, word, or double word sizes.
- The operands are stored as 64 or 128 bit data in MMX registers, XMM registers, or memory.

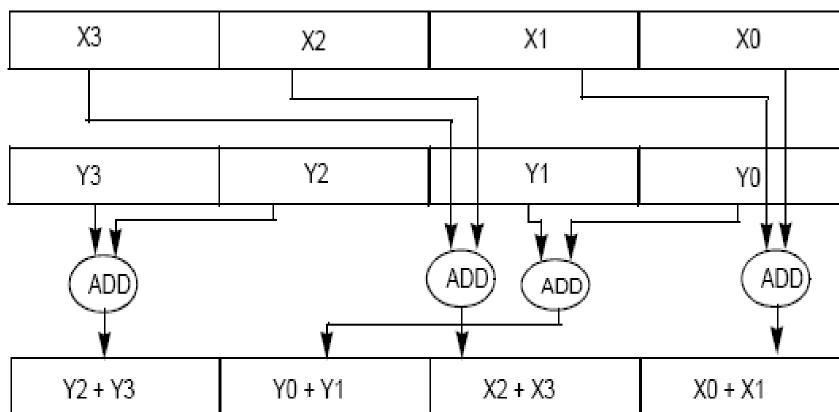
68

SSSE3 Instructions

- Twelve instructions that perform horizontal addition or subtraction operations.
- Six instructions that evaluate the absolute values.
- Two instructions that perform multiply and add operations and speed up the evaluation of dot products.
- Two instructions that accelerate packed-integer multiply operations and produce integer values with scaling.
- Two instructions that perform a byte-wise, in-place shuffle according to the second shuffle control operand.
- Six instructions that negate packed integers in the destination operand if the signs of the corresponding element in the source operand is less than zero.
- Two instructions that align data from the composite of two operands.

69

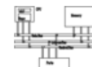
SSSE3 Instructions



Horizontal data movement: PHADDD

70

SSE4



- SSE4 comprises of two sets of extensions
  - SSE4.1: targeted to improve the performance of media, imaging and 3D graphics. It also adds instructions for improving compiler vectorization and significantly increase support for packed dword computation. It has 47 new instructions.
  - SSE4.2: improves performance in string and text processing. It has 7 new instructions.
- SSE4 instructions do not use MMX registers. Two of the SSE4.2 instructions operate on general-purpose registers; the rest of SSE4.2 instruction and SSE4.1 instructions operate on XMM registers.

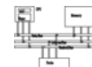
71

SSE4.1 Instructions

- Two instructions perform packed dword multiplies.
- Two instructions perform floating-point dot products with input/output selects.
- One instruction performs a load with a streaming hint.
- Six instructions simplify packed blending.
- Eight instructions expand support for packed integer MIN/MAX.
- Four instructions support floating-point round with selectable rounding mode and precision exception override.
- Seven instructions improve data insertion and extractions from XMM registers
- Twelve instructions improve packed integer format conversions (sign and zero extensions).
- One instruction improves SAD (sum absolute difference) generation for small block sizes.
- One instruction aids horizontal searching operations.
- One instruction improves masked comparisons.
- One instruction adds qword packed equality comparisons.
- One instruction adds dword packing with unsigned saturation.

72

SSE4.2 Instructions



- String and text processing that can take advantage of single-instruction multiple data programming techniques.
- Application-targeted accelerator (ATA) instructions.
- A SIMD integer instruction that enhances the capability of the 128-bit integer SIMD capability in SSE4.1.

73

References

- *Intel MMX for Multimedia PCs*, CACM, Jan. 1997
- Chapter 11 *The MMX Instruction Set*, The Art of Assembly
- Chap. 9, 10, 11 of IA-32 Intel Architecture Software Developer's Manual: Volume 1: Basic Architecture
- http://www.csie.ntu.edu.tw/~r89004/hive/sse/page_1.html

74