Table of Contents

Instructions: 1-7
- Basic Building Blocks:
  - load & store: 8-12
  - shuffle: 13-15
  - arithmetic: 16-18

Load Instructions
Vector Arithmetic
Reorder Instruction
Reorder Instructions: Complex Numbers
Reorder Instructions: 4x4 Transpose
Load 4 reals
Store 4 reals
Load 4 complex
Matrix Multiplication, vector loads: 5-10
Matrix Multiplication, 5 complex loads: 11-12
Matrix Multiplication, Analysis: 13
Normal Vectorization, Vector
Load code
Unroll
WHT
The Box Operator
Short Vector FFT

vector indexing:

\[
a: \begin{bmatrix}
3 & 2 & 1 & 0
\end{bmatrix}
\]

in most instructions the order of operands matters
Load Instructions (SSE and later)

16-byte aligned \( m \)

\[ a = \text{mm_load_ps}(m), \quad a = \text{mm_load_d}(m), \quad a = \text{mm_loadl}(m) \]

8-byte aligned

\[ a = \text{mm_load_ps}(a, m) \]

\[ b = \text{mm_load_d}(b, m) \]

4-byte aligned

\[ a = \text{mm_load_ss}(m) \]

Stores are analogous

Constants (Special Load Instructions) (SSE and later)

\[ C = \text{mm_set_ps}(1.0, 2.0, 3.0, 4.0); \]
\[ d = \text{mm_set_c}(1.0); \]
\[ e = \text{mm_set_s}(1.0); \]
\[ f = \text{mm_set_p}(1.0); \]
Vector arithmetic (SSE and later)

\[ c = \text{mm}_-\text{add}ps(a, b) \]

Same:
\[ c = \text{mm}_-\text{sub}ps(a, b) \]
\[ c = \text{mm}_-\text{mul}ps(a, b) \]

Scalar arithmetic (SSE and later)

\[ c = \text{mm}_-\text{add}ss(a, b) \]

Add-Sub (SSE3 and later)

\[ c = \text{mm}_-\text{add}sub-\text{ps}(a, b) \]

(alternate add & sub of vector elements)
Reorder Instructions (SSE and later)

Unpack lo

\[ c = \textproc{mm_unpacklo_ps}(a, b) \]

\[ \begin{align*}
    c_0 &= a_0 \\
    c_1 &= b_0 \\
    c_2 &= a_1 \\
    c_3 &= b_1
\end{align*} \]

Unpack hi

\[ c = \textproc{mm_unpackhi_ps}(a, b) \]

\[ \begin{align*}
    c_0 &= a_2 \\
    c_1 &= b_2 \\
    c_2 &= a_3 \\
    c_3 &= b_3
\end{align*} \]

Shuffle

\[ c = \textproc{mm_shuffle_ps}(a, b, \textproc{mm_shuffle}(i, j, i, j)) \]

\[ \begin{align*}
    c_0 &= a_i \\
    c_1 &= a_j \\
    c_2 &= b_k \\
    c_3 &= b_l
\end{align*} \]

Any element of \( b \) or any element of \( a \)
**align (SSSE3 and later)**

```
c = _mm_castps_s128 (_mm_loadr_epi8(  
  _mm_castsi128_ps(a)  
  _mm_castsi128_ps(b)))
```

**shuffle (SSSE3 and later)**

```
c is filled with any element of b or c
```

any

```
_mm_shuffle_epi8( )
```
Matrix-Vector Product (SSE4 and later)

Dot-product instruction

- mm_dp_ps(a, b, mask)

"computes the pointwise product of a and b and writes the sum of the resulting numbers into elements of c where the others are set to 0."
Example: Load 4 real numbers from arbitrary memory location (SSE)

```c
#define SCALAP_LOAD(out, m0, m1, m2, m3)
    out = _mm_load_ss(m0);
    b = _mm_load_ss(m1);
    c = _mm_shuffle_ps(o, b, _MM_SHUFFLE(1, 0, 0));
    d = _mm_load_ss(m2);
    e = _mm_load_ss(m3);
    f = _mm_shuffle_ps(c, e, _MM_SHUFFLE(1, 0, 1));
    out = _mm_shuffle_ps(c, f, _MM_SHUFFLE(1, 0, 0));
```

Note: Whenever possible, avoid this by restructuring the algorithm or data format to have aligned vector loads (see page 1)

- Equivalent to macro on page 11 (but the above is "safest")
float \theta [20] = 3 \ldots 3.

-- declspec (align(16)) g[4];
-- m128 vf;

\begin{array}{l}
g[0] = f[3];
g[1] = f[5];
g[2] = f[12];
g[3] = f[17];
\end{array}
\begin{array}{l}
vf = \text{mm_loadps}(g);
\end{array}
\begin{array}{l}
\text{// operations}
\end{array}
\begin{array}{l}
\text{mm_storeps}(g, vf);
f[7] = g[0];
f[13] = g[1];
f[17] = g[3];
\end{array}

Lods innocent, but is really bad.
Most people try that at some point.
Looks at the assembly to see for yourself.
Set load from tight and memory (SSE and later)

You can do: (see page 1)

\[-m_{128\text{ uf}} = \_mm\_set\_ps(1.0, 3.0, 2.0, 1.0)\; \rightarrow \text{vector load of 128 bit constant}\]

Compilers let you do this type of thing:

\[
\text{float} f[20] = 3 \ldots 3;
\]

\[-m_{128\text{ uf}} = \_mm\_set\_ps(f[3], f[5], f[2], f[7])\; \text{equivalent to page 8}\]

However, internally: 4 loads, 3 shuffles.

Do not use \(_mm\_set\_ps()\) on variables if you can avoid it. (see page 8)
Example: load 4 complex numbers (load 4 pairs of numbers) (SSE and later)

Strided access

Unit stride access

4 instructions

(Same for store ops)
Reorder Instruction - Examples (SSE and later)

Interleaved Complex \rightarrow \text{split complex} : L_2^8

\[
\begin{align*}
\text{c} &= \text{mm\_shuffle\_ps}(a, b, \text{MM\_SHUFFLE}(2, 0, 2, 0)), \\
\text{d} &= \text{mm\_shuffle\_ps}(a, b, \text{MM\_SHUFFLE}(3, 1, 3, 1))
\end{align*}
\]

\[c \rightarrow \text{split complex} \rightarrow \text{interleaved complex} : L_4^8\]

\[
\begin{align*}
\text{c} &= \text{mm\_unpack\_lo\_ps}(a, b) \\
\text{d} &= \text{mm\_unpack\_hi\_ps}(a, b)
\end{align*}
\]

Reverse Vector : J_4

\[
\begin{align*}
\text{b} &= \text{mm\_shuffle\_ps}(a, b, \text{MM\_SHUFFLE}(0, 1, 2, 3))
\end{align*}
\]
SSSE3:
- `psll b` - `{mm_alignr.epi8()}`
  1 instruction

```c
C = mm_shuffle_ps(b, {-mm_shuffle_ps(0, 0, {mm_shuffle_ps(1, 2, 1))}});
L = mm_shuffle_ps(b, {-mm_shuffle_ps(0, 0, {mm_shuffle_ps(3, 2, 1))}});
```

Shift by 1 (SSSE)
Transpose 4x4:

\[
\begin{bmatrix}
V_3 & V_2 & V_1 & V_0 \\
7 & 1 & 3 & 5 \\
1 & 1 & 0 & 1 \\
1 & 1 & 1 & 5 \\
0 & 1 & 0 & 0
\end{bmatrix}
\]

Defined macro (SSE and later)

```
#define _MM_TRANSPOSE4_PS(V0, V1, V2, V3) {
    _mm256_t t0, t1, t2, t3;
    t0 = _mm_shuffle_ps(V0, V1, _MM_SHUFFLE(2, 0, 2));
    t1 = _mm_shuffle_ps(V0, V1, _MM_SHUFFLE(3, 1, 3));
    t2 = _mm_shuffle_ps(V2, V3, _MM_SHUFFLE(2, 0, 2));
    t3 = _mm_shuffle_ps(V2, V3, _MM_SHUFFLE(3, 1, 3));
    V0 = _mm_shuffle_ps(t0, t2, _MM_SHUFFLE(2, 0, 2));
    V1 = _mm_shuffle_ps(t1, t3, _MM_SHUFFLE(2, 0, 2));
    V2 = _mm_shuffle_ps(t0, t2, _MM_SHUFFLE(3, 1, 3));
    V3 = _mm_shuffle_ps(t1, t3, _MM_SHUFFLE(3, 1, 3));
}
```
Matrix - Vector Product (SSE and beyond)

needs reworking

5 vector loads
4 vector multiplies
but how to do the sums?

transpose

3 adds, 8 shuffles

other solutions (SSE3 and beyond)
Matrix-Vector Product (SSE4)

\[
\begin{align*}
\begin{bmatrix} 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \end{bmatrix} &= 2 \\
\begin{bmatrix} 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \end{bmatrix} &= 0 \\
\begin{bmatrix} 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \end{bmatrix} &= 2 \\
\begin{bmatrix} 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \end{bmatrix} &= 0 \\
\begin{bmatrix} 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \end{bmatrix} &= 2 \\
\begin{bmatrix} 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \end{bmatrix} &= 0
\end{align*}
\]

4 dot product instructions
3 add instructions

or 4 dot products

+ 8 scalar stores or 5

Performance implications?
Complex Multiplication double SSE3

\[(a + ib)(c + id) = (ac - bd) + (ad + bc)i\]

\[\text{movddup} \quad \text{-mm_loadddup_pd()}
\]
\[\text{mm_movedup()}
\]
\[\text{mm_shufd_r()}\]

\[\text{mmsubmenu()}\]

\[\text{vector store}\]

SSE, SSE2: see slide