# CS 342, CCNY Spring 2021

# DOT PRODUCT$(x, y) = \sum_{k=0}^{n} x_n y_n$ COMPUTATION
# Using SIDM ( SSE vector instructions)

Reference: ***Implement a Horizontal Add/Subtract with SSE3 Instructions for dot product computation***
**https://software.intel.com/en-us/articles/implement-a-horizontal-addsubtract-with-sse3-instructions**

## Source code

```
int main(int argc, char* argv[])
{
/*
      float a[N], b[N], x = 0.0;

      for (i = 0; i < N; i++)

      x = x + a[i]*b[i];

*/

const int N = 8;
static float a[N]={1.0,2.0,1.0,2.0,1.0,2.0,1.0,2.0},
             b[N]={2.0,1.0,2.0,1.0,2.0,1.0,2.0,1.0},
        x = 0.0;
      float *aPointer = a;
      float *bPointer = b;

      __asm
            {
      pxor    xmm0, xmm0   ;initialize xmm0 to 0 , xmm0 will serve as x
      mov     eax, dword ptr[aPointer]    ;eax points to a[]
      mov     ebx, dword ptr[bPointer]    ;ebx points to b[]
      mov     ecx, N             ;number of elements in arrays

myLOOP:
     movups  xmm1, [eax]      ;four values of a in xmm1
     movups  xmm2, [ebx]      ;four values of b in xmm2
     mulps   xmm1, xmm2       ;mulitply a[i]*b[i]
     addps   xmm0, xmm1 ;add x + a[i]*b[i]


      add eax, 16       ;increment pa by 4
      add ebx, 16       ;increment pb by 4
      sub ecx, 4        ;loop-4
      jnz myLOOP   ;loop if ecx not 0

      haddps  xmm0, xmm0           ;horizontal add
      haddps  xmm0, xmm0           ;horizontal add
      movss   dword ptr[x], xmm0    ;result goes to x

        }
      return 0;
}
```

## Disassembly window

```
14: static float a[N]={1.0,2.0,1.0,2.0,1.0,2.0,1.0,2.0},
    15:          b[N]={2.0,1.0,2.0,1.0,2.0,1.0,2.0,1.0},
    16:       x = 0.0;
    20:     float *aPointer = a;
013913E5 C7 45 EC 00 80 39 01 mov         dword ptr [aPointer],1398000h
    21:     float *bPointer = b;
013913EC C7 45 E0 20 80 39 01 mov         dword ptr [bPointer],1398020h
    22:
    23:     __asm
    24:           {
    26: pxor    xmm0, xmm0 ;initialize xmm0 to 0 , xmm0 will serve as x
013913F3 66 0F EF C0          pxor        xmm0,xmm0
    27:     mov     eax, dword ptr[aPointer]     ;eax points to a[]
013913F7 8B 45 EC             mov         eax,dword ptr [aPointer]
    28:     mov     ebx, dword ptr[bPointer]     ;ebx points to b[]
013913FA 8B 5D E0             mov         ebx,dword ptr [bPointer]
    29:     mov      ecx, N        ;number of elements in arrays
013913FD 8B 4D F8             mov         ecx,dword ptr [N]
    30:
    31:     myLOOP:
    32:     movups  xmm1, [eax]   ;four values of a in xmm1
01391400 0F 10 08             movups      xmm1,xmmword ptr [eax]
    33:     movups  xmm2,   [ebx] ;four values of b in xmm2
01391403 0F 10 13             movups      xmm2,xmmword ptr [ebx]
    34:     mulps    xmm1, xmm2   ;mulitply a[i]*b[i]
01391406 0F 59 CA             mulps       xmm1,xmm2
    35:     addps   xmm0, xmm1    ;add x + a[i]*b[i]
01391409 0F 58 C1             addps       xmm0,xmm1
    36:
```

```
    38:     add eax, 16             ;increment pa by 4
0139140C 83 C0 10             add         eax,10h
    39:     add ebx, 16             ;increment pb by 4
0139140F 83 C3 10             add         ebx,10h
    40:     sub ecx, 4                  ;loop-4
01391412 83 E9 04             sub         ecx,4
    41:     jnz myLOOP              ;loop if ecx not 0
01391415 75 E9                jne         myLOOP (01391400h)
    42:
    43:     haddps  xmm0, xmm0              ;horizontal add
01391417 F2 0F 7C C0          haddps      xmm0,xmm0
    44:     haddps  xmm0, xmm0             ;horizontal add
0139141B F2 0F 7C C0          haddps      xmm0,xmm0
    45:     movss   dword ptr[x], xmm0  ;result goes to x
0139141F F3 0F 11 05 80 81 39 01 movss       dword ptr ds:[1398180h],xmm0
        48:         }
    49:
    50:     return 0;
01391427 33 C0                xor         eax,eax
    51: }
```

**Register file**

```
Registers
EAX = CCCCCCCC EBX = 7EDC8000 ECX = 00000000 EDX = 00000001 ESI = 00000000 EDI = 002DF974 EIP = 013913E5
ESP = 002DF884 EBP = 002DF974 EFL = 00000206

MM0 = 0000000000000000 MM1 = 0000000000000000 MM2 = 0000000000000000 MM3 = 0000000000000000
MM4 = 0000000000000000 MM5 = 0000000000000000 MM6 = 0000000000000000 MM7 = 0000000000000000

XMM0 = 00000000000000000000000000000000
XMM1 = 00000000000000000000000000000000
XMM2 = 00000000000000000000000000000000
XMM3 = 00000000000000000000000000000000
XMM4 = 00000000000000000000000000000000
XMM5 = 00000000000000000000000000000000
XMM6 = 00000000000000000000000000000000
XMM7 = 00000000000000000000000000000000
XMM00 = +0.00000E+000      XMM01 = +0.00000E+000      XMM02 = +0.00000E+000      XMM03 = +0.00000E+000
XMM10 = +0.00000E+000      XMM11 = +0.00000E+000      XMM12 = +0.00000E+000      XMM13 = +0.00000E+000
XMM20 = +0.00000E+000      XMM21 = +0.00000E+000      XMM22 = +0.00000E+000      XMM23 = +0.00000E+000
XMM30 = +0.00000E+000      XMM31 = +0.00000E+000      XMM32 = +0.00000E+000      XMM33 = +0.00000E+000
XMM40 = +0.00000E+000      XMM41 = +0.00000E+000      XMM42 = +0.00000E+000      XMM43 = +0.00000E+000
XMM50 = +0.00000E+000      XMM51 = +0.00000E+000      XMM52 = +0.00000E+000      XMM53 = +0.00000E+000
XMM60 = +0.00000E+000      XMM61 = +0.00000E+000      XMM62 = +0.00000E+000      XMM63 = +0.00000E+000
XMM70 = +0.00000E+000      XMM71 = +0.00000E+000      XMM72 = +0.00000E+000      XMM73 = +0.00000E+000
MXCSR = 00001F80

0x002df960 = CCCCCCCC
```

## Memory stores array

```
Memory 3
0x01398000  00 00 80 3f  ...?
0x01398004  00 00 00 40  ...@
0x01398008  00 00 80 3f  ...?
0x0139800C  00 00 00 40  ...@
0x01398010  00 00 80 3f  ...?
0x01398014  00 00 00 40  ...@
0x01398018  00 00 80 3f  ...?
0x0139801C  00 00 00 40  ...@
0x01398020  00 00 00 40  ...@
0x01398024  00 00 80 3f  ...?
0x01398028  00 00 00 40  ...@
0x0139802C  00 00 80 3f  ...?
0x01398030  00 00 00 40  ...@
0x01398034  00 00 80 3f  ...?
0x01398038  00 00 00 40  ...@
0x0139803C  00 00 80 3f  ...?
```

At the address **0x01398000 is floating point representation**

0x3f800000=

**0011 1111 1000 0000 0000 0000 0000 0000**

**+ $2^{127-127}(1+0) = 1.0$**

0x40000000=

**0100 0000 0000 0000 0000 0000 0000 0000**

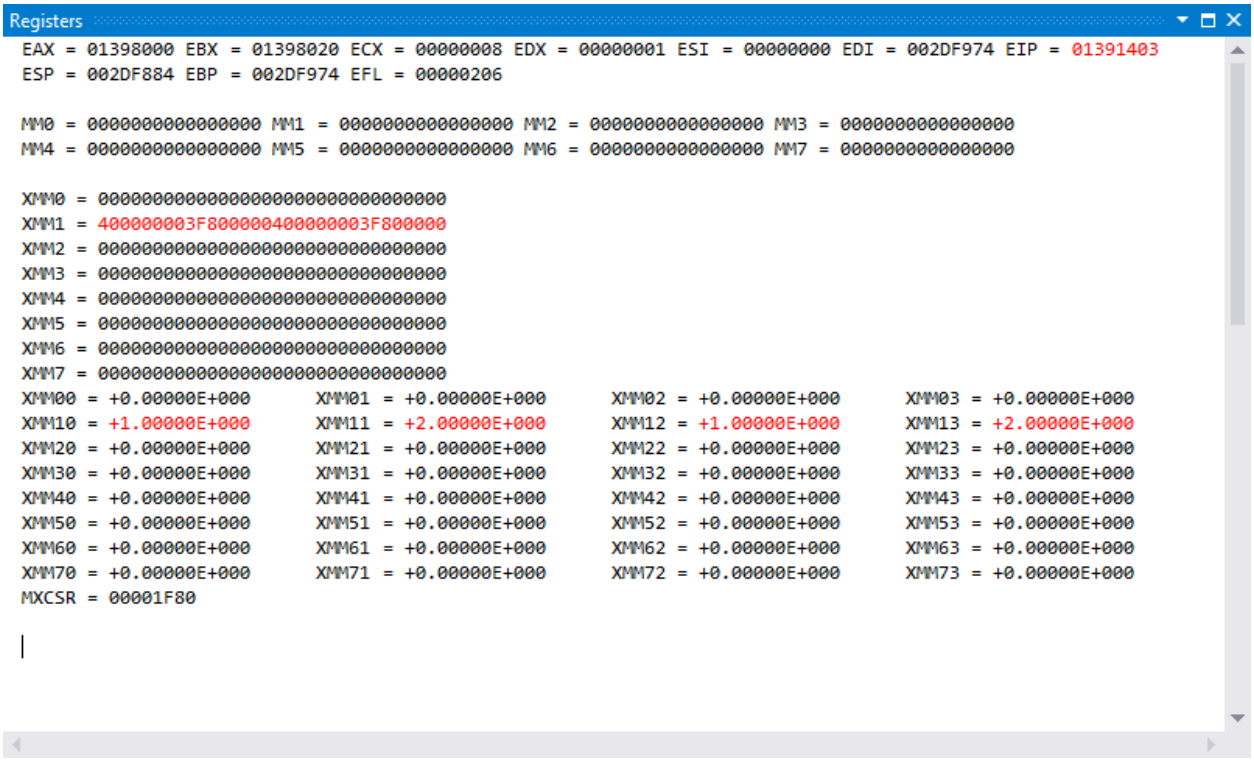
**+ $2^{128-127}(1+0) = 2.0$**

## Register file after loading first 4 elements of the array

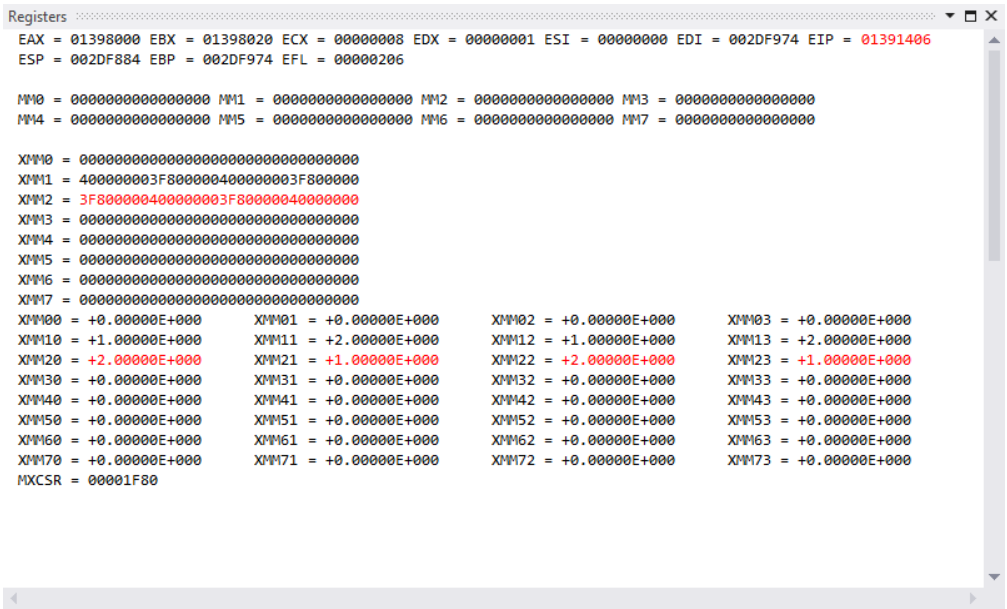
**EAX** contains the address **0X01398000** of the first element of the array a[]

**EBX** contains the address **0X01398020** of the first element of the array b[]

**ECX** contains the number of elements in each array




**AFTER LOADING THE SECOND ARRAY TO XMM2**



Register file after **01391406 0F 59 CA mulps xmm1,xmm2**

```
Registers
EAX = 01398000 EBX = 01398020 ECX = 00000008 EDX = 00000001 ESI = 00000000 EDI = 002DF974 EIP = 01391409
ESP = 002DF884 EBP = 002DF974 EFL = 00000206

MM0 = 0000000000000000 MM1 = 0000000000000000 MM2 = 0000000000000000 MM3 = 0000000000000000
MM4 = 0000000000000000 MM5 = 0000000000000000 MM6 = 0000000000000000 MM7 = 0000000000000000

XMM0 = 00000000000000000000000000000000
XMM1 = 40000000400000004000000040000000
XMM2 = 3F800000400000003F800000400000000
XMM3 = 00000000000000000000000000000000
XMM4 = 00000000000000000000000000000000
XMM5 = 00000000000000000000000000000000
XMM6 = 00000000000000000000000000000000
XMM7 = 00000000000000000000000000000000
XMM00 = +0.00000E+000      XMM01 = +0.00000E+000      XMM02 = +0.00000E+000      XMM03 = +0.00000E+000
XMM10 = +2.00000E+000      XMM11 = +2.00000E+000      XMM12 = +2.00000E+000      XMM13 = +2.00000E+000
XMM20 = +2.00000E+000      XMM21 = +1.00000E+000      XMM22 = +2.00000E+000      XMM23 = +1.00000E+000
XMM30 = +0.00000E+000      XMM31 = +0.00000E+000      XMM32 = +0.00000E+000      XMM33 = +0.00000E+000
XMM40 = +0.00000E+000      XMM41 = +0.00000E+000      XMM42 = +0.00000E+000      XMM43 = +0.00000E+000
XMM50 = +0.00000E+000      XMM51 = +0.00000E+000      XMM52 = +0.00000E+000      XMM53 = +0.00000E+000
XMM60 = +0.00000E+000      XMM61 = +0.00000E+000      XMM62 = +0.00000E+000      XMM63 = +0.00000E+000
XMM70 = +0.00000E+000      XMM71 = +0.00000E+000      XMM72 = +0.00000E+000      XMM73 = +0.00000E+000
MXCSR = 00001F80
```

**After 01391409 0F 58 C1 addps xmm0,xmm1 ;xmm1+xmm0->xmm0**

```
Registers
EAX = 01398000 EBX = 01398020 ECX = 00000008 EDX = 00000001 ESI = 00000000 EDI = 002DF974 EIP = 0139140C
ESP = 002DF884 EBP = 002DF974 EFL = 00000206

MM0 = 0000000000000000 MM1 = 0000000000000000 MM2 = 0000000000000000 MM3 = 0000000000000000
MM4 = 0000000000000000 MM5 = 0000000000000000 MM6 = 0000000000000000 MM7 = 0000000000000000

XMM0 = 40000000400000004000000040000000
XMM1 = 40000000400000004000000040000000
XMM2 = 3F800000400000003F800000400000000
XMM3 = 00000000000000000000000000000000
XMM4 = 00000000000000000000000000000000
XMM5 = 00000000000000000000000000000000
XMM6 = 00000000000000000000000000000000
XMM7 = 00000000000000000000000000000000
XMM00 = +2.00000E+000      XMM01 = +2.00000E+000      XMM02 = +2.00000E+000      XMM03 = +2.00000E+000
XMM10 = +2.00000E+000      XMM11 = +2.00000E+000      XMM12 = +2.00000E+000      XMM13 = +2.00000E+000
XMM20 = +2.00000E+000      XMM21 = +1.00000E+000      XMM22 = +2.00000E+000      XMM23 = +1.00000E+000
XMM30 = +0.00000E+000      XMM31 = +0.00000E+000      XMM32 = +0.00000E+000      XMM33 = +0.00000E+000
XMM40 = +0.00000E+000      XMM41 = +0.00000E+000      XMM42 = +0.00000E+000      XMM43 = +0.00000E+000
XMM50 = +0.00000E+000      XMM51 = +0.00000E+000      XMM52 = +0.00000E+000      XMM53 = +0.00000E+000
XMM60 = +0.00000E+000      XMM61 = +0.00000E+000      XMM62 = +0.00000E+000      XMM63 = +0.00000E+000
XMM70 = +0.00000E+000      XMM71 = +0.00000E+000      XMM72 = +0.00000E+000      XMM73 = +0.00000E+000
MXCSR = 00001F80
```

**The Dot product is stored in xmm0**

**Using the HADDPS instruction, as shown in the debug window lines 43 and 44**. **HADDPS** performs a single-precision addition on contiguous data elements. The first data element of the result is obtained by adding the first and second elements of the first operand. The second element is obtained by adding the third and fourth elements of the first operand. The third element is obtained by adding the first and second elements of the second operand. The fourth element is obtained by adding the third and fourth elements of the second operand

```
Registers
EAX = 01398020 EBX = 01398040 ECX = 00000000 EDX = 00000001 ESI = 00000000 EDI = 002DF974 EIP = 0139141F
ESP = 002DF884 EBP = 002DF974 EFL = 00000246

MM0 = 0000000000000000 MM1 = 0000000000000000 MM2 = 0000000000000000 MM3 = 0000000000000000
MM4 = 0000000000000000 MM5 = 0000000000000000 MM6 = 0000000000000000 MM7 = 0000000000000000

XMM0 = 41800000418000004180000041800000
XMM1 = 40000000400000004000000040000000
XMM2 = 3F8000004000000 3F80000040000000
XMM3 = 00000000000000000000000000000000
XMM4 = 00000000000000000000000000000000
XMM5 = 00000000000000000000000000000000
XMM6 = 00000000000000000000000000000000
XMM7 = 00000000000000000000000000000000
XMM00 = +1.60000E+001      XMM01 = +1.60000E+001      XMM02 = +1.60000E+001      XMM03 = +1.60000E+001
XMM10 = +2.00000E+000      XMM11 = +2.00000E+000      XMM12 = +2.00000E+000      XMM13 = +2.00000E+000
XMM20 = +2.00000E+000      XMM21 = +1.00000E+000      XMM22 = +2.00000E+000      XMM23 = +1.00000E+000
XMM30 = +0.00000E+000      XMM31 = +0.00000E+000      XMM32 = +0.00000E+000      XMM33 = +0.00000E+000
XMM40 = +0.00000E+000      XMM41 = +0.00000E+000      XMM42 = +0.00000E+000      XMM43 = +0.00000E+000
XMM50 = +0.00000E+000      XMM51 = +0.00000E+000      XMM52 = +0.00000E+000      XMM53 = +0.00000E+000
XMM60 = +0.00000E+000      XMM61 = +0.00000E+000      XMM62 = +0.00000E+000      XMM63 = +0.00000E+000
XMM70 = +0.00000E+000      XMM71 = +0.00000E+000      XMM72 = +0.00000E+000      XMM73 = +0.00000E+000
MXCSR = 00001F80

0x01398180 = 00000000
```

Finally at the Register XMM0 **4 copies of the result (dot product) is in floating point representation**

0x41800000=

**0100 0001 1000 0000 0000 0000 0000 0000**

**+ $2^{131-127}(1+0) = 16.0$**

0x40000000=

**0100 0000 0000 0000 0000 0000 0000 0000**

**+ $2^{128-127}(1+0) = 2.0$**