Accessibility: adjust monitor distances

This is the size most subtitles will be

- Most text will be approximately this size

- Some will be larger

  - Some slightly smaller

- Images like this aren't meant to be read, they're only to illustrate
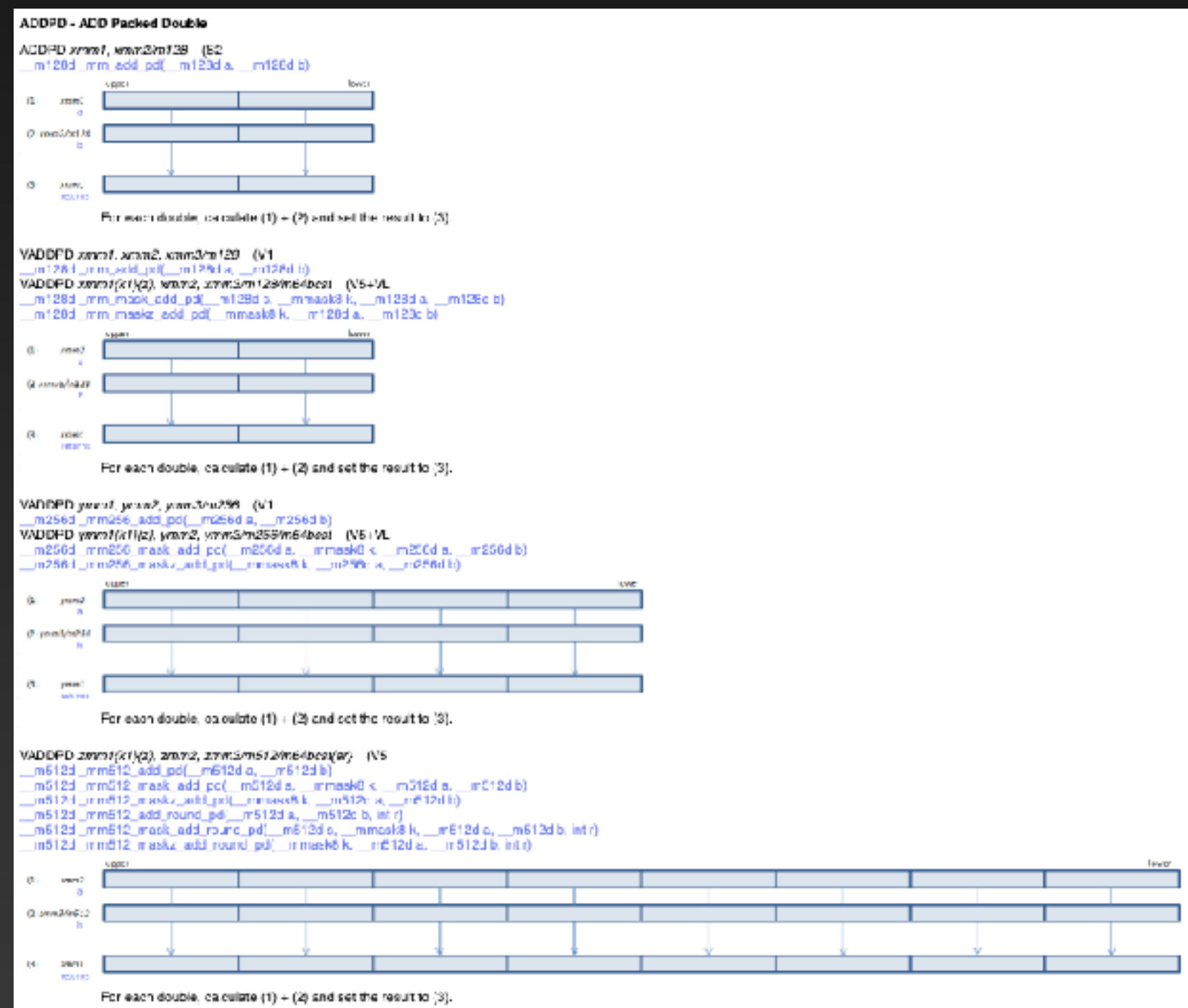

- Code will typically look like this

  - Assembly will look like this:
    ```
    fld     QWORD PTR [rsi+rcx*8]
    fadd    QWORD PTR [rdx+rcx*8]
    fstp    QWORD PTR [rdx+rcx*8]
    ```

  - C and other high level languages will look like this
    ```
    void add(int n, double *x, double *y)
        {
            for (int i = 0; i < n; i++)
            y[i] = x[i] + y[i];
        }
    ```

AVX10 for HPC:

A reasonable solution to the 7 levels of AVX-512 folly

Felix LeClair for EasyBuild, Q4 2023

On the docket:
Topics covered

1. What is an ISA?

2. x87 "It's not a great floating point unit, but it's what we have"

3. MMX "What is SIMD?"

4. SSE 1-2 "SIMD and Real Types"

5. SSE 3-4.2 ""

6. Check-in 1

7. AVX1

8. AVX2

9. Check in 2, electric boogaloo

10. AVX "3" or "I can't believe it's not AVX512"

11. AVX "3.5" or "Whatever Phi was"

12. AVX "4" or AVX512 VL, DQ, BW

13. AVX "5" or AVX512 IFMA, VBMI

14. AVX "6" or AVX512BF16

15. AVX "7" or AVX512 VPOPCNTDQ, VNNI, VBMI2, BITALG

16. AVX "8" or VP2INTERSECT

17. AVX "9" or FP16

18. AVX "10" or "The One we care about"

The ISA is the language your computer speaks. Just like natural languages, it can have several dialects (variants) and regional terms (extensions) only some computers understand.

Robert Clausecker

For programmers: ISA : CPU :: Language standard : compiler

The ISA is the language the chip understands.

Alexander Monakov

CHAOSHACKER

[An ISA] is a mapping between integers and operations

Violet the Chromatic Bat

[An ISA] is like a box of chocolates

Brandon Biggs

An ISA is like musical notes, [it makes] an orchestra of transistors perform a symphony!

Andreas Herten

[An ISA] is like a vocabulary[.]

Moritz Lehmann

[An ISA] is a contract between hardware and software designers that lets us work together

Andrew Richards

For higher-level programmers, [an ISA] is the stable API for the CPU.

Stephen Canon

An ISA is the train-line to which HW designers chain their screaming future colleagues to.

Tom Forsyth

Section 1
Floating point *OR* SIMD

An ISA from 1978 isn't going to fit the needs of today
"I heard you like extensions, so I put extensions extending your extended extensions"

- Much like Fortran 77 Code today, only implementing HW specs from the 70's can be loosely described as a poor life choice

  - The original 8086 and 8088 didn't have HW support for Floating point.

  - Enjoy your ~O(N^2) performance whenever you need to divide A/B

The 8086
Perform DAXPY z[]=(x[]+y[])*z[]

```c
// function to add the elements of two arrays
void DAXPY(int n, double *x, double *y,double *z)
{
    int i = 0;
  do {z[i] = (x[i] + y[i]) * z[i];
        i++;}
            while(i<n);

}

int main(void)
{
    int N = 1<<10; // ~1000 elements

    double *x = new double[N];
    double *y = new double[N];
    double *z = new double[N];

    // Run kernel on 1K elements
    DAXPY(N, x, y, z);

    return 0;
}
```

```asm
DAXPY(int, double*, double*,
double*):
 push    ebp
 push    ebx
 push    edi
 push    esi
 sub     esp,0xc
 call    c <DAXPY(int, double*,
double*, double*)+0xc>
 pop     ebx
 add     ebx,0x3
        R_386_GOTPC
_GLOBAL_OFFSET_TABLE_
 xor     esi,esi
 mov     edi,DWORD PTR [esp+0x2c]
 mov     ebp,DWORD PTR [esp+0x24]
 mov     eax,DWORD PTR [esp+0x28]
 push    DWORD PTR [eax+esi*8+0x4]
 push    DWORD PTR [eax+esi*8]
 push    DWORD PTR [ebp+esi*8+0x4]
 push    DWORD PTR [ebp+esi*8+0x0]
 call    31 <DAXPY(int, double*,
double*, double*)+0x31>
        R_386_PLT32 __adddf3
 add     esp,0x10
 push    DWORD PTR [edi+esi*8+0x4]
 push    DWORD PTR [edi+esi*8]
 push    edx
 push    eax
 call    42 <DAXPY(int, double*,
double*, double*)+0x42>
        R_386_PLT32 __muldf3
 add     esp,0x10
 mov     DWORD PTR [edi+esi*8],eax
 mov     DWORD PTR
[edi+esi*8+0x4],edx
 inc     esi
 cmp     esi,DWORD PTR [esp+0x20]
 jl      1d <DAXPY(int, double*,
double*, double*)+0x1d>
 add     esp,0xc
 pop     esi
 pop     edi
 pop     ebx
 pop     ebp
 ret
main:

 push    ebp
 push    ebx
 push    edi
 push    esi
 sub     esp,0xc
 call    6b <main+0xc>
 pop     ebx
 add     ebx,0x3
        R_386_GOTPC
_GLOBAL_OFFSET_TABLE_
 sub     esp,0xc
 mov     ebp,0x2000
 push    ebp
 call    7c <main+0x1d>
        R_386_PLT32 operator new[]
(unsigned int)
 add     esp,0x10
 mov     esi,eax
 sub     esp,0xc
 push    ebp
 call    8a <main+0x2b>
        R_386_PLT32 operator new[]
(unsigned int)
 add     esp,0x10
 mov     edi,eax
 sub     esp,0xc
 push    ebp
 call    98 <main+0x39>
        R_386_PLT32 operator new[]
(unsigned int)
 add     esp,0x10
 push    eax
 push    edi
 push    esi
 push    0x400
 call    a8 <main+0x49>
        R_386_PC32 DAXPY(int, double*,
double*, double*)
 add     esp,0x10
 xor     eax,eax
 add     esp,0xc
 pop     esi
 pop     edi
 pop     ebx
 pop     ebp
 ret
```
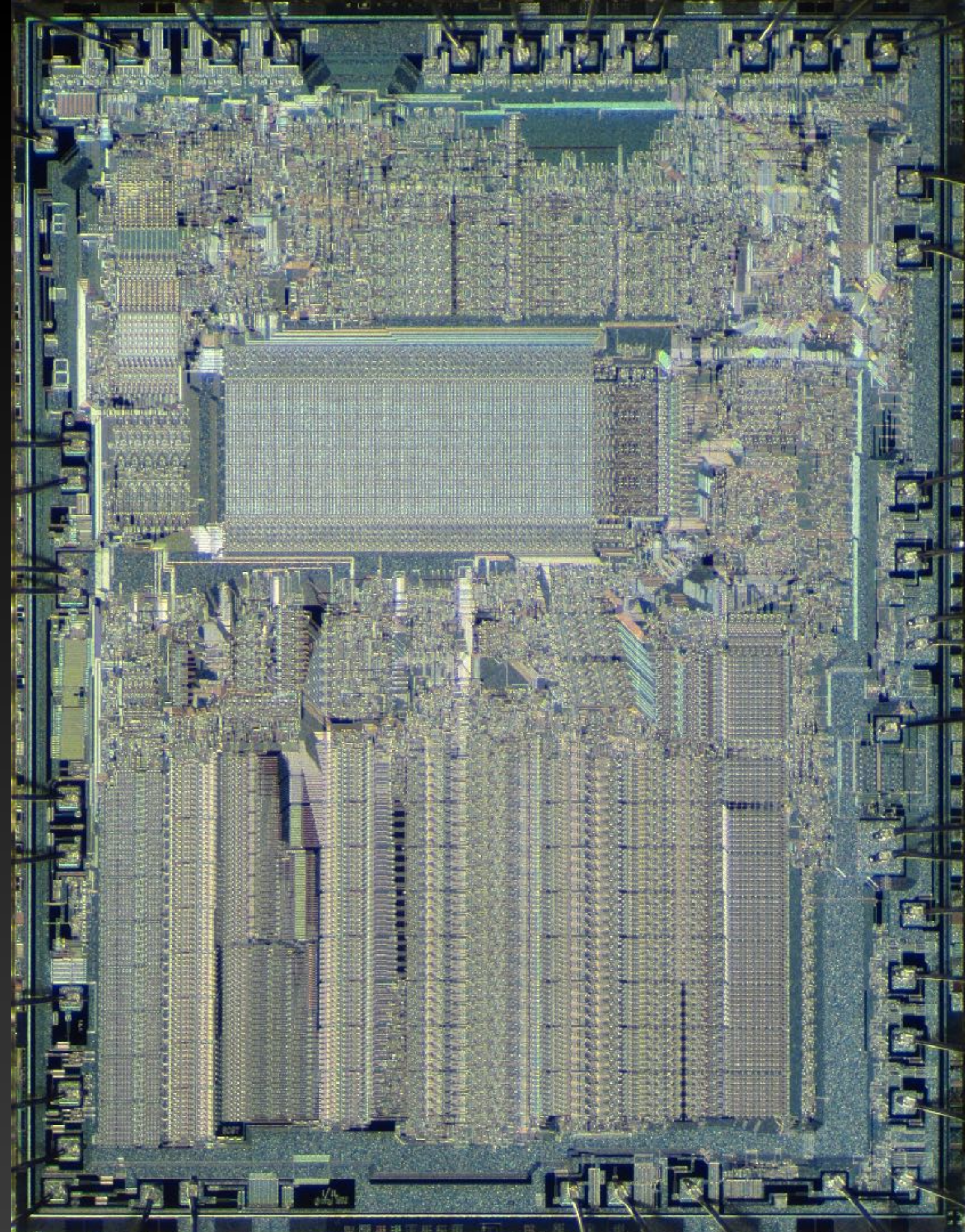
The 8087-1980
Floating point on x86!

- 3 data types:

  - FP32

  - FP64

  - FP80

- Inspired what would become the IEEE-754 (1985) floating spec we know and love

- 80bit registers

The 8087
Perform DAXPY z[]=(x[]+y[])*z[]

```c
// function to add the elements of two arrays
void DAXPY(int n, double *x, double *y,double *z)
{
    int i = 0;
  do {z[i] = (x[i] + y[i]) * z[i];
        i++;}
           while(i<n);

}

int main(void)
{
  int N = 1<<10; // ~1000 elements

  double *x = new double[N];
  double *y = new double[N];
  double *z = new double[N];

  // Run kernel on 1K elements
  DAXPY(N, x, y, z);

  return 0;
}
```
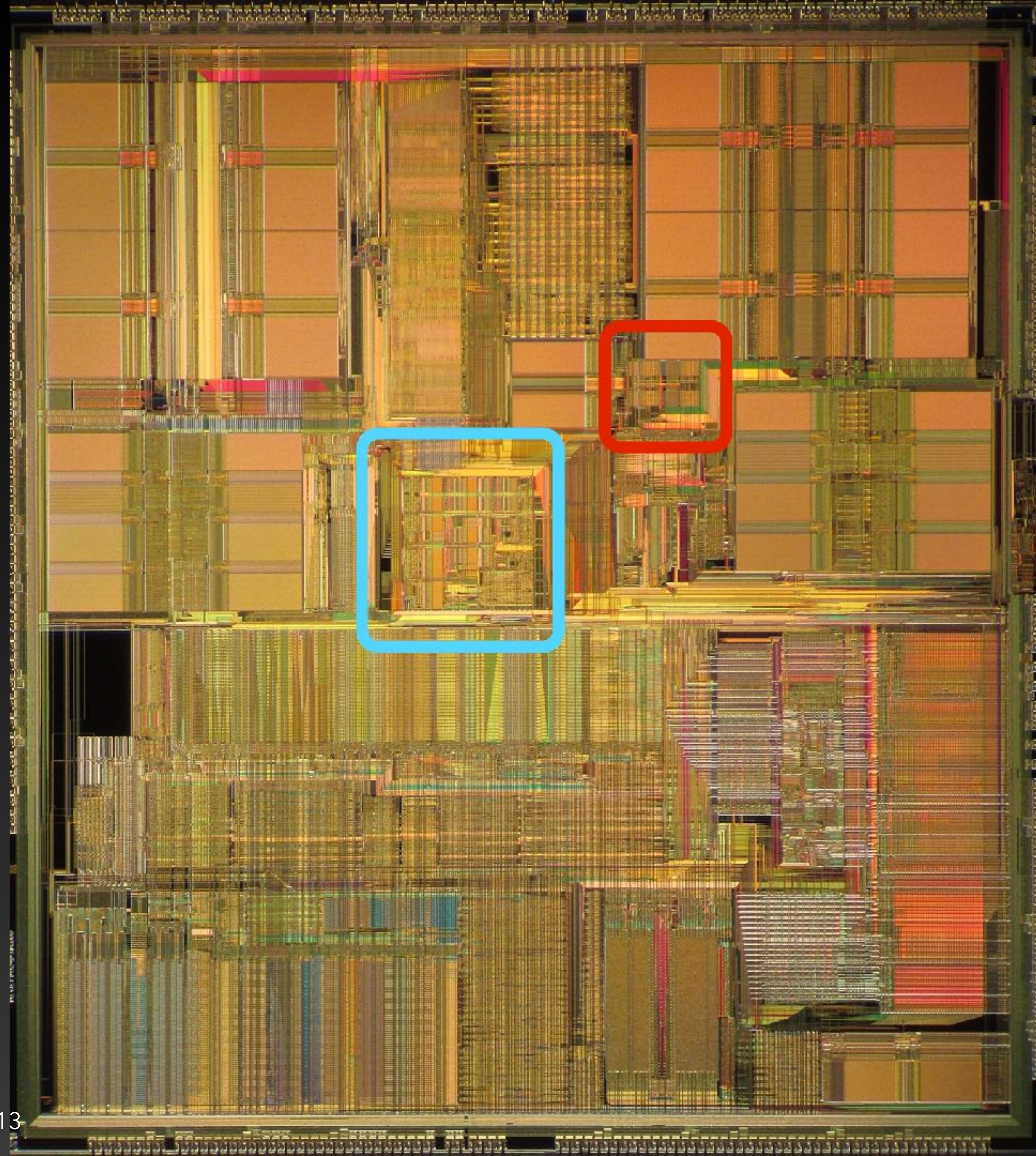
```asm
DAXPY(int, double*, double*,
double*):
 push   edi
 push   esi
 xor    eax,eax
 mov    ecx,DWORD PTR
[esp+0x18]
 mov    edx,DWORD PTR
[esp+0x14]
 mov    esi,DWORD PTR
[esp+0x10]
 mov    edi,DWORD PTR
[esp+0xc]
 fld    QWORD PTR
[esi+eax*8]
 fadd   QWORD PTR
[edx+eax*8]
 fmul   QWORD PTR
[ecx+eax*8]
 fstp   QWORD PTR
[ecx+eax*8]
 inc    eax
 cmp    eax,edi
 jl     14 <DAXPY(int,
double*, double*, double*)
+0x14>
 pop    esi
 pop    edi
 ret
main:
 push   ebp
 push   ebx
 push   edi
 push   esi
 sub    esp,0xc
 call   34 <main+0xc>
 pop    ebx
 add    ebx,0x3
    R_386_GOTPC
_GLOBAL_OFFSET_TABLE_
```

```asm
 sub    esp,0xc
 mov    ebp,0x2000
 push   ebp
 call   45 <main+0x1d>
    R_386_PLT32 operator
new[](unsigned int)
 add    esp,0x10
 mov    esi,eax
 sub    esp,0xc
 push   ebp
 call   53 <main+0x2b>
    R_386_PLT32 operator
new[](unsigned int)
 add    esp,0x10
 mov    edi,eax
 sub    esp,0xc
 push   ebp
 call   61 <main+0x39>
    R_386_PLT32 operator
new[](unsigned int)
 add    esp,0x10
 push   eax
 push   edi
 push   esi
 push   0x400
 call   71 <main+0x49>
    R_386_PC32 DAXPY(int,
double*, double*, double*)
 add    esp,0x10
 xor    eax,eax
 add    esp,0xc
 pop    esi
 pop    edi
 pop    ebx
 pop    ebp
 ret
```

MMX -1997
SIMD on x86

- 4 data types:
  - 1* Int64
  - 2* Int32
  - 4* Int16
  - 8* Int8
- Used the lower 64 bits of the 80 bit x87 register

MMX
Perform DAXPY z[]=(x[]+y[])*z[]

```c
// function to add the elements of two arrays
void DAXPY(int n, double *x, double *y,double *z)
{
    int i = 0;
  do {z[i] = (x[i] + y[i]) * z[i];
        i++;}
            while(i<n);

}

int main(void)
{
  int N = 1<<10; // ~1000 elements

  double *x = new double[N];
  double *y = new double[N];
  double *z = new double[N];

  // Run kernel on 1K elements
  DAXPY(N, x, y, z);

  return 0;
}
```

14

```asm
DAXPY(int, double*, double*,
double*):
 push   edi
 push   esi
 xor    eax,eax
 mov    ecx,DWORD PTR
[esp+0x18]
 mov    edx,DWORD PTR
[esp+0x14]
 mov    esi,DWORD PTR
[esp+0x10]
 mov    edi,DWORD PTR
[esp+0xc]
 fld    QWORD PTR
[esi+eax*8]
 fadd   QWORD PTR
[edx+eax*8]
 fmul   QWORD PTR
[ecx+eax*8]
 fstp   QWORD PTR
[ecx+eax*8]
 inc    eax
 cmp    eax,edi
 jl     14 <DAXPY(int,
double*, double*, double*)
+0x14>
 pop    esi
 pop    edi
 ret
main:
 push   ebp
 push   ebx
 push   edi
 push   esi
 sub    esp,0xc
 call   34 <main+0xc>
 pop    ebx
 add    ebx,0x3
    R_386_GOTPC
_GLOBAL_OFFSET_TABLE_
```

```asm
 sub    esp,0xc
 mov    ebp,0x2000
 push   ebp
 call   45 <main+0x1d>
    R_386_PLT32 operator
new[](unsigned int)
 add    esp,0x10
 mov    esi,eax
 sub    esp,0xc
 push   ebp
 call   53 <main+0x2b>
    R_386_PLT32 operator
new[](unsigned int)
 add    esp,0x10
 mov    edi,eax
 sub    esp,0xc
 push   ebp
 call   61 <main+0x39>
    R_386_PLT32 operator
new[](unsigned int)
 add    esp,0x10
 push   eax
 push   edi
 push   esi
 push   0x400
 call   71 <main+0x49>
    R_386_PC32 DAXPY(int,
double*, double*, double*)
 add    esp,0x10
 xor    eax,eax
 add    esp,0xc
 pop    esi
 pop    edi
 pop    ebx
 pop    ebp
 ret
```

x87
Perform DAXPY z[]=(x[]+y[])*z[]

```
DAXPY(int, double*, double*,              R_386_PLT32 operator new[]
double*):                         (unsigned int)
 push    edi                       add     esp,0x10
 push    esi                       mov     esi,eax
 xor     eax,eax                   sub     esp,0xc
 mov     ecx,DWORD PTR [esp+0x18]  push    ebp
 mov     edx,DWORD PTR [esp+0x14]  call    53 <main+0x2b>
 mov     esi,DWORD PTR [esp+0x10]      R_386_PLT32 operator new[]
 mov     edi,DWORD PTR [esp+0xc]   (unsigned int)
 fld     QWORD PTR [esi+eax*8]     add     esp,0x10
 fadd    QWORD PTR [edx+eax*8]     mov     edi,eax
 fmul    QWORD PTR [ecx+eax*8]     sub     esp,0xc
 fstp    QWORD PTR [ecx+eax*8]     push    ebp
 inc     eax                       call    61 <main+0x39>
 cmp     eax,edi                       R_386_PLT32 operator new[]
 jl      14 <DAXPY(int, double*,  (unsigned int)
double*, double*)+0x14>           add     esp,0x10
 pop     esi                       push    eax
 pop     edi                       push    edi
 ret                               push    esi
main:                              push    0x400
 push    ebp                       call    71 <main+0x49>
 push    ebx                           R_386_PC32 DAXPY(int,
 push    edi                      double*, double*, double*)
 push    esi                       add     esp,0x10
 sub     esp,0xc                   xor     eax,eax
 call    34 <main+0xc>             add     esp,0xc
 pop     ebx                       pop     esi
 add     ebx,0x3                   pop     edi
     R_386_GOTPC                   pop     ebx
_GLOBAL_OFFSET_TABLE_              pop     ebp
 sub     esp,0xc                   ret
 mov     ebp,0x2000
 push    ebp
 call    45 <main+0x1d>
```

MMX
Perform DAXPY z[]=(x[]+y[])*z[]

```
DAXPY(int, double*, double*,              R_386_PLT32 operator new[]
double*):                         (unsigned int)
 push    edi                       add     esp,0x10
 push    esi                       mov     esi,eax
 xor     eax,eax                   sub     esp,0xc
 mov     ecx,DWORD PTR [esp+0x18]  push    ebp
 mov     edx,DWORD PTR [esp+0x14]  call    53 <main+0x2b>
 mov     esi,DWORD PTR [esp+0x10]      R_386_PLT32 operator new[]
 mov     edi,DWORD PTR [esp+0xc]   (unsigned int)
 fld     QWORD PTR [esi+eax*8]     add     esp,0x10
 fadd    QWORD PTR [edx+eax*8]     mov     edi,eax
 fmul    QWORD PTR [ecx+eax*8]     sub     esp,0xc
 fstp    QWORD PTR [ecx+eax*8]     push    ebp
 inc     eax                       call    61 <main+0x39>
 cmp     eax,edi                       R_386_PLT32 operator new[]
 jl      14 <DAXPY(int, double*,  (unsigned int)
double*, double*)+0x14>           add     esp,0x10
 pop     esi                       push    eax
 pop     edi                       push    edi
 ret                               push    esi
main:                              push    0x400
 push    ebp                       call    71 <main+0x49>
 push    ebx                           R_386_PC32 DAXPY(int,
 push    edi                      double*, double*, double*)
 push    esi                       add     esp,0x10
 sub     esp,0xc                   xor     eax,eax
 call    34 <main+0xc>             add     esp,0xc
 pop     ebx                       pop     esi
 add     ebx,0x3                   pop     edi
     R_386_GOTPC                   pop     ebx
_GLOBAL_OFFSET_TABLE_              pop     ebp
 sub     esp,0xc                   ret
 mov     ebp,0x2000
 push    ebp
 call    45 <main+0x1d>
```

8087
Perform DAXPY z[]=(x[]+y[])*z[]

MMX
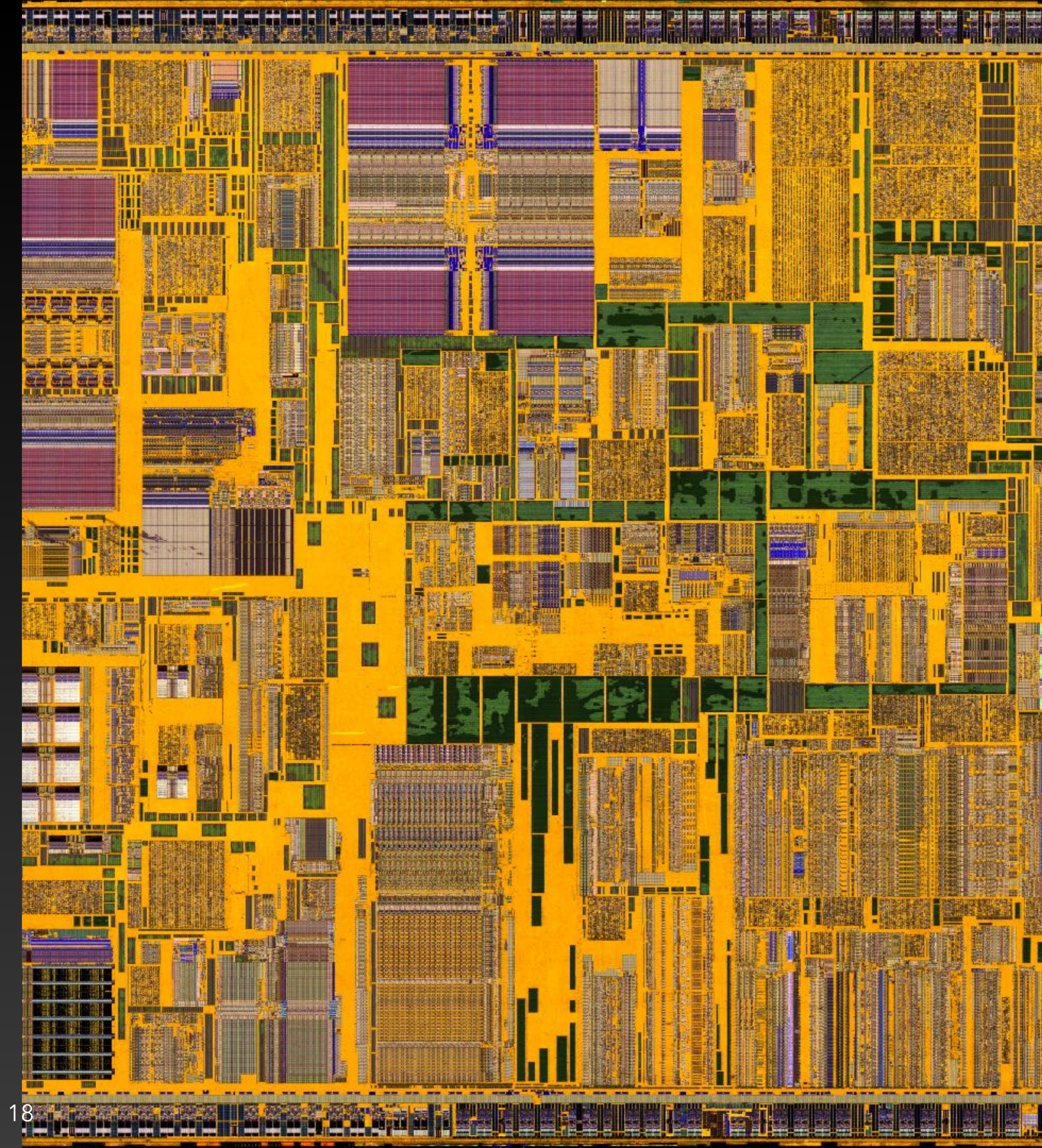## Perform DAXPY z[]=(x[]+y[])*z[]

16

Section 2
Floating point *AND* SIMD

SSE1-1999
Streaming SIMD extensions

- 1 data type:
  - 4* FP32

- 8 new registers
  - xmm0-7
  - 128 bits wide
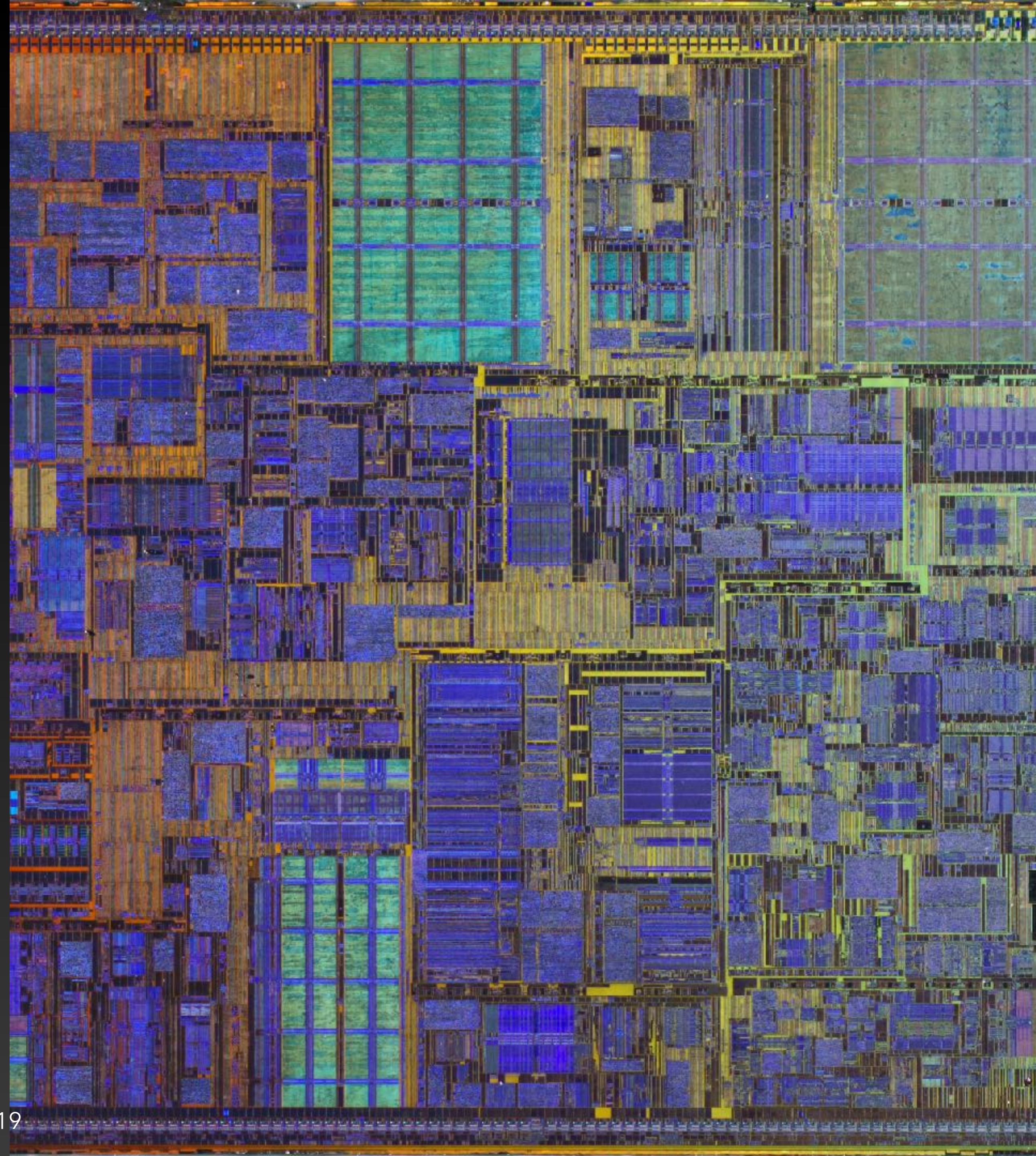
SSE2-2000
"Real" general purpose SIMD

- 6 data types:

  - 2* FP64(!!!)

  - 4* FP32

  - 2* Int64

  - 4* Int32

  - 8* Int16

  - 16* Int8

SSE2

Perform DAXPY z[]=(x[]+y[])*z[]

```c
// function to add the elements of two arrays
void DAXPY(int n, double *x, double *y,double *z)
{
    int i = 0;
  do {z[i] = (x[i] + y[i]) * z[i];
        i++;}
            while(i<n);

}


int main(void)
{
    int N = 1<<10; // ~1000 elements

    double *x = new double[N];
    double *y = new double[N];
    double *z = new double[N];

    // Run kernel on 1K elements
    DAXPY(N, x, y, z);

    return 0;
}
```

```asm
DAXPY(int, double*, double*,
double*):
 cmp     edi,0x2
 mov     eax,0x1
 cmovge eax,edi
 cmp     eax,0xa
 jae    17 <DAXPY(int,
double*, double*, double*)
+0x17>
 xor     edi,edi
 jmp     b6 <DAXPY(int,
double*, double*, double*)
+0xb6>
 lea     rdi,[rcx+rax*8]
 lea     r8,[rsi+rax*8]
 lea     r9,[rdx+rax*8]
 cmp     r8,rcx
 seta    r10b
 cmp     rdi,rsi
 seta    r11b
 cmp     r9,rcx
 seta    r8b
 cmp     rdi,rdx
 seta    r9b
 xor     edi,edi
 test    r10b,r11b
 jne     b6 <DAXPY(int,
double*, double*, double*)
+0xb6>
 and     r8b,r9b
 jne     b6 <DAXPY(int,
double*, double*, double*)
+0xb6>
 mov     edi,eax
 and     edi,0x7ffffffc
 xor     r8d,r8d
 cs nop WORD PTR
[rax+rax*1+0x0]
 movupd xmm0,XMMWORD PTR
[rsi+r8*8]
 movupd xmm1,XMMWORD PTR
[rsi+r8*8+0x10]
 movupd xmm2,XMMWORD PTR
[rdx+r8*8]
 addpd   xmm2,xmm0
 movupd xmm0,XMMWORD PTR
[rdx+r8*8+0x10]
 addpd   xmm0,xmm1
 movupd xmm1,XMMWORD PTR
[rcx+r8*8]
 mulpd   xmm1,xmm2
 movupd xmm2,XMMWORD PTR
[rcx+r8*8+0x10]
 mulpd   xmm2,xmm0
 movupd XMMWORD PTR
[rcx+r8*8],xmm1
 movupd XMMWORD PTR
[rcx+r8*8+0x10],xmm2
 add     r8,0x4
```

```asm
 cmp     rdi,r8
 jne    60 <DAXPY(int,
double*, double*, double*)
+0x60>
 cmp     rdi,rax
 je     165 <DAXPY(int,
double*, double*, double*)
+0x165>
 mov     r8,rdi
 not     r8
 add     r8,rax
 mov     r9,rax
 and     r9,0x3
 je     ec <DAXPY(int,
double*, double*, double*)
+0xec>
 nop    DWORD PTR
[rax+rax*1+0x0]
 movsd  xmm0,QWORD PTR
[rsi+rdi*8]
 addsd  xmm0,QWORD PTR
[rdx+rdi*8]
 mulsd  xmm0,QWORD PTR
[rcx+rdi*8]
 movsd  QWORD PTR
[rcx+rdi*8],xmm0
 inc     rdi
 dec     r9
 jne     d0 <DAXPY(int,
double*, double*, double*)
+0xd0>
 cmp     r8,0x3
 jb     165 <DAXPY(int,
double*, double*, double*)
+0x165>
 data16 data16 data16 data16
 cs nop WORD PTR
[rax+rax*1+0x0]
 movsd  xmm0,QWORD PTR
[rsi+rdi*8]
 addsd  xmm0,QWORD PTR
[rdx+rdi*8]
 mulsd  xmm0,QWORD PTR
[rcx+rdi*8]
 movsd  QWORD PTR
[rcx+rdi*8],xmm0
 movsd  xmm0,QWORD PTR
[rsi+rdi*8+0x8]
 addsd  xmm0,QWORD PTR
[rdx+rdi*8+0x8]
 mulsd  xmm0,QWORD PTR
[rcx+rdi*8+0x8]
 movsd  QWORD PTR
[rcx+rdi*8+0x8],xmm0
 movsd  xmm0,QWORD PTR
[rsi+rdi*8+0x10]
 addsd  xmm0,QWORD PTR
[rdx+rdi*8+0x10]
```

```asm
 mulsd  xmm0,QWORD PTR
[rcx+rdi*8+0x10]
 movsd  QWORD PTR
[rcx+rdi*8+0x10],xmm0
 movsd  xmm0,QWORD PTR
[rsi+rdi*8+0x18]
 addsd  xmm0,QWORD PTR
[rdx+rdi*8+0x18]
 mulsd  xmm0,QWORD PTR
[rcx+rdi*8+0x18]
 movsd  QWORD PTR
[rcx+rdi*8+0x18],xmm0
 add     rdi,0x4
 cmp     rax,rdi
 jne    100 <DAXPY(int,
double*, double*, double*)
+0x100>
 ret
 cs nop  WORD PTR
[rax+rax*1+0x0]
main:
 push    r14
 push    rbx
 push    rax
 mov     edi,0x2000
 call   17e <main+0xe>
        R_X86_64_PLT32 operator
new[](unsigned long)-0x4
 mov     rbx,rax
 mov     edi,0x2000
 call   18b <main+0x1b>
        R_X86_64_PLT32 operator
new[](unsigned long)-0x4
 mov     r14,rax
 mov     edi,0x2000
 call   198 <main+0x28>
        R_X86_64_PLT32 operator
new[](unsigned long)-0x4
 mov     edi,0x400
 mov     rsi,rbx
 mov     rdx,r14
 mov     rcx,rax
 call   1ab <main+0x3b>
        R_X86_64_PLT32 DAXPY(int,
double*, double*, double*)-0x4
 xor     eax,eax
 add     rsp,0x8
 pop     rbx
 pop     r14
 ret
```

SSE2 IS

STILL

The default ISA target

When you, your users, your packages or your libraries don't provide `-march={foo}` you are locking your code generation to an ISA from the year 2000
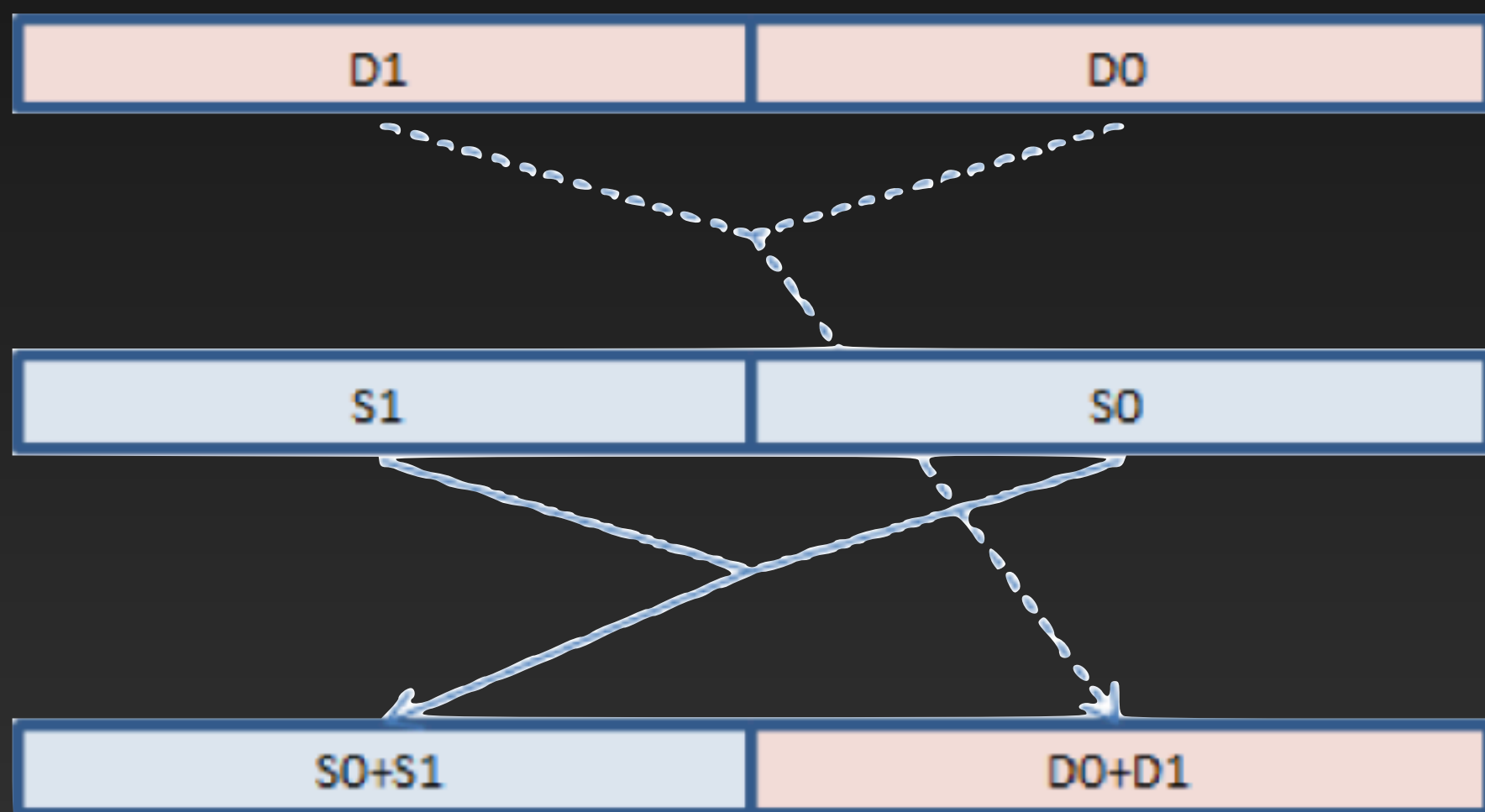
It's hard to believe, but that's over 2 decades ago, and technology has come a long way

21

SSE3-2004
Horizontal sums!

- `HADDPD xmm1, xmm2`

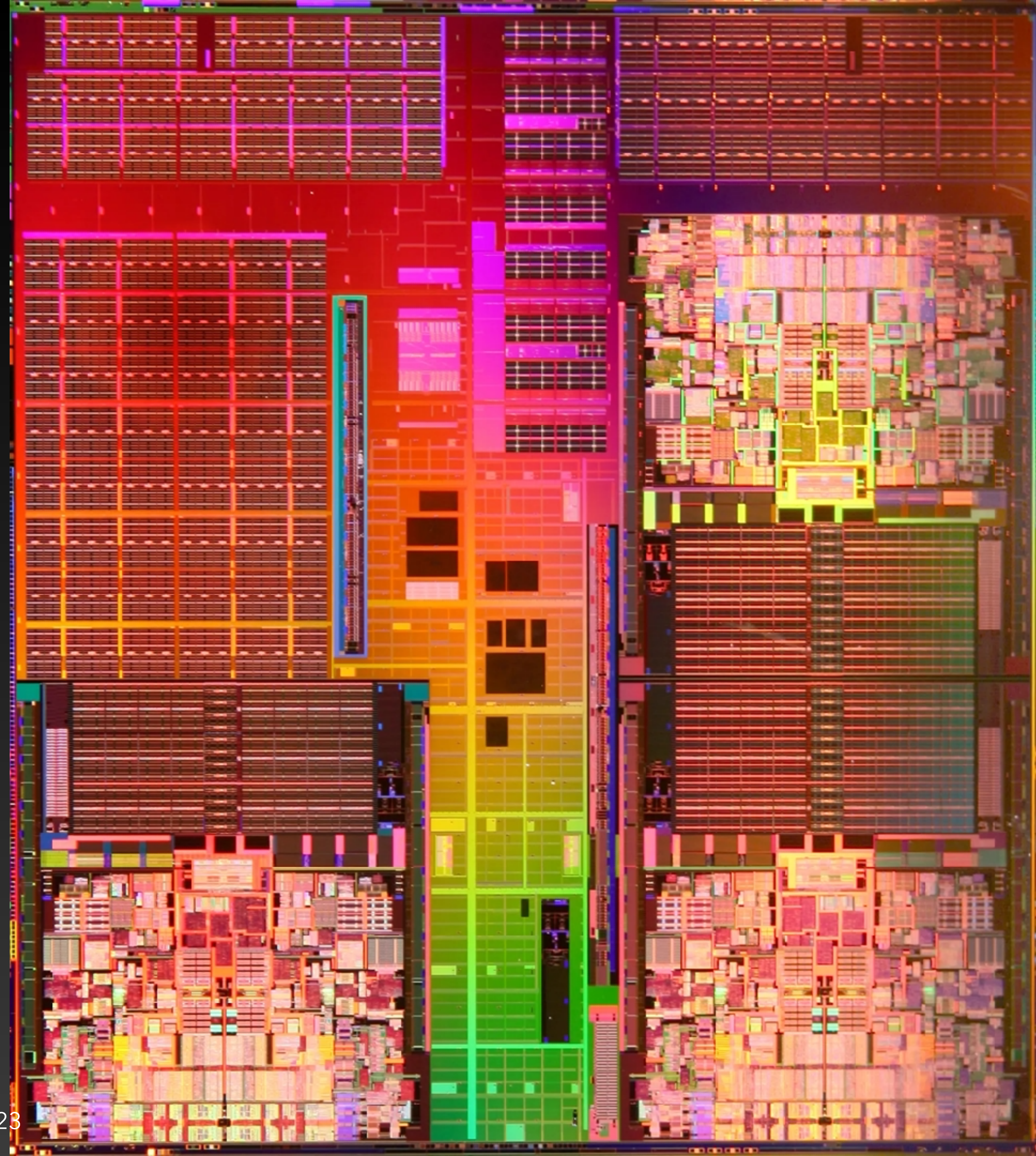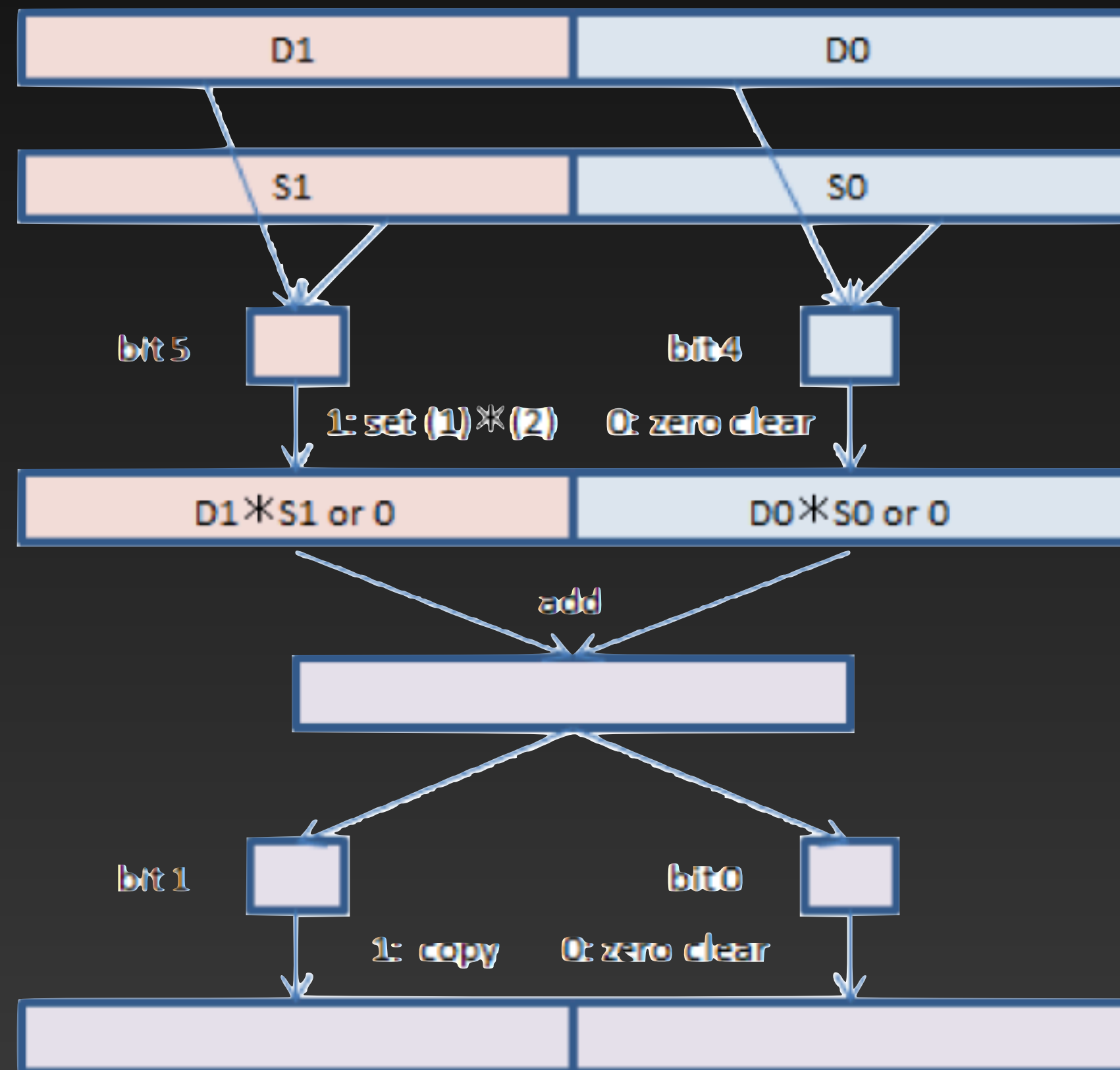| D1	D0

S1	S0

S0+S1	D0+D1

22

SSE4.1 and SSE4.2-2004
Dot Products!

- `DPPD xmm1, xmm2, imm8`

Targets that implement all the way to SSE4.2 are considered x86-64-V2

A "generic" CPU target that includes every CPU, including embed, since ~2013

Provides the compiler with a generic target for ISA's, doesn't include any tuning/implementation data

MISC
"Um actually"- a close friend

- There were also some weird in between extensions

  - AMD 3DNOW!

  - SSSE3

  - SSE5*

  - etc.

- None of them survived to become anything particularly relevant

- With the move to 64 bit operating systems, we double from 8x128 bit SSE registers to 16x128 bit registers registers. You can thank the fine folks at AMD for that! (AMD64)

Section 1+2
Quick clarifications?

Section 3
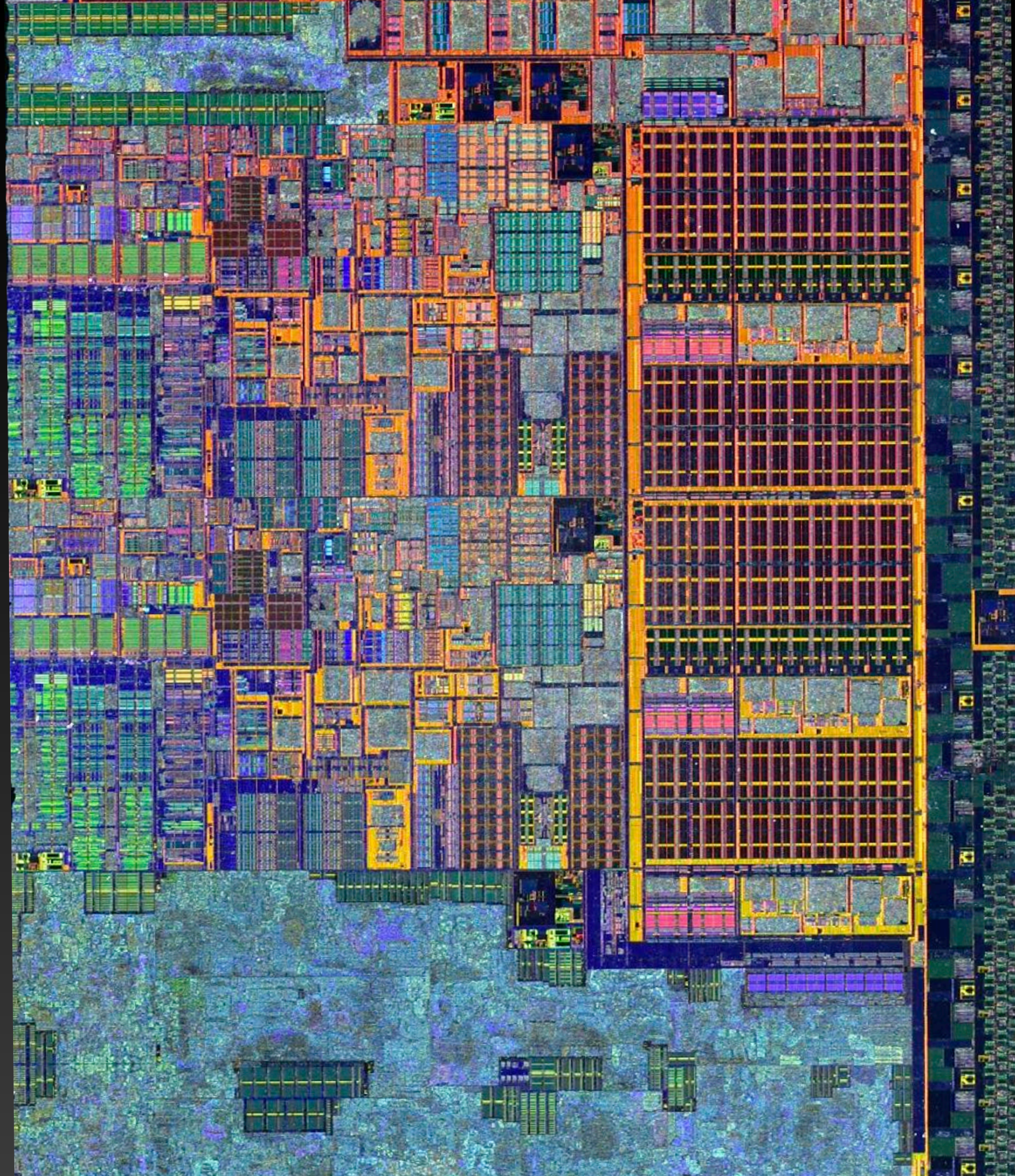# *Advanced* Floating point SIMD

AVX(1) ~2010
"Advanced" SIMD

- 16 x 256 bit vector registers

- 2 data types:

  - 4* FP64

  - 8* FP32

- "Expands" the encoding space with VEX encoding; no more register clobbering!

- addpd -> vaddpd

AVX
Perform DAXPY z[]=(x[]+y[])*z[]

```cpp
// function to add the elements of two arrays
void DAXPY(int n, double *x, double *y,double *z)
{

    int i = 0;
  do {z[i] = (x[i] + y[i]) * z[i];
        i++;}
            while(i<n);

}


int main(void)
{

   int N = 1<<10; // ~1000 elements

   double *x = new double[N];
   double *y = new double[N];
   double *z = new double[N];


   // Run kernel on 1K elements
   DAXPY(N, x, y, z);


   return 0;
}
```

```asm
DAXPY(int, double*, double*, double*):
  cmp     edi,0x2
  mov     eax,0x1
  cmovge  eax,edi
  cmp     eax,0x10
  jae     17 <DAXPY(int, double*, double*, double*)
+0x17>
  xor     edi,edi
  jmp     de <DAXPY(int, double*, double*, double*)
+0xde>
  lea     rdi,[rcx+rax*8]
  lea     r8,[rsi+rax*8]
  cmp     r8,rcx
  seta    r10b
  lea     r8,[rdx+rax*8]
  cmp     rdi,rsi
  seta    r11b
  cmp     r8,rcx
  seta    r8b
  cmp     rdi,rdx
  seta    r9b
  xor     edi,edi
  test    r10b,r11b
  jne     de <DAXPY(int, double*, double*, double*)
+0xde>
  and     r8b,r9b
  jne     de <DAXPY(int, double*, double*, double*)
+0xde>
  mov     edi,eax
  and     edi,0x7ffffff0
  xor     r8d,r8d
  xchg    ax,ax
  vmovupd ymm0,YMMWORD PTR [rdx+r8*8]
  vmovupd ymm1,YMMWORD PTR [rdx+r8*8+0x20]
  vmovupd ymm2,YMMWORD PTR [rdx+r8*8+0x40]
  vmovupd ymm3,YMMWORD PTR [rdx+r8*8+0x60]
  vmulpd  ymm0,ymm0,YMMWORD PTR [rcx+r8*8]
  vaddpd  ymm0,ymm0,YMMWORD PTR [rsi+r8*8]
  vmulpd  ymm1,ymm1,YMMWORD PTR [rcx+r8*8+0x20]
  vaddpd  ymm1,ymm1,YMMWORD PTR [rsi+r8*8+0x20]
  vmulpd  ymm2,ymm2,YMMWORD PTR [rcx+r8*8+0x40]
  vaddpd  ymm2,ymm2,YMMWORD PTR [rsi+r8*8+0x40]
  vmulpd  ymm3,ymm3,YMMWORD PTR [rcx+r8*8+0x60]
  vaddpd  ymm3,ymm3,YMMWORD PTR [rsi+r8*8+0x60]
  vmovupd YMMWORD PTR [rcx+r8*8],ymm0
  vmovupd YMMWORD PTR [rcx+r8*8+0x20],ymm1
  vmovupd YMMWORD PTR [rcx+r8*8+0x40],ymm2
  vmovupd YMMWORD PTR [rcx+r8*8+0x60],ymm3
  add     r8,0x10
  cmp     rdi,r8
  jne     60 <DAXPY(int, double*, double*, double*)
+0x60>
  cmp     rdi,rax
  je      185 <DAXPY(int, double*, double*, double*)
+0x185>
  mov     r8,rdi
  not     r8
  add     r8,rax
  mov     r9,rax
  and     r9,0x3
  je      10c <DAXPY(int, double*, double*, double*)
+0x10c>
  vmovsd  xmm0,QWORD PTR [rdx+rdi*8]
  vmulsd  xmm0,xmm0,QWORD PTR [rcx+rdi*8]
  vaddsd  xmm0,xmm0,QWORD PTR [rsi+rdi*8]
  vmovsd  QWORD PTR [rcx+rdi*8],xmm0
  inc     rdi
```

```asm
  dec     r9
  jne     f0 <DAXPY(int, double*, double*, double*)
+0xf0>
  cmp     r8,0x3
  jb      185 <DAXPY(int, double*, double*, double*)
+0x185>
  data16 data16 data16 data16 cs nop WORD PTR
[rax+rax*1+0x0]
  vmovsd  xmm0,QWORD PTR [rdx+rdi*8]
  vmulsd  xmm0,xmm0,QWORD PTR [rcx+rdi*8]
  vaddsd  xmm0,xmm0,QWORD PTR [rsi+rdi*8]
  vmovsd  QWORD PTR [rcx+rdi*8],xmm0
  vmovsd  xmm0,QWORD PTR [rdx+rdi*8+0x8]
  vmulsd  xmm0,xmm0,QWORD PTR [rcx+rdi*8+0x8]
  vaddsd  xmm0,xmm0,QWORD PTR [rsi+rdi*8+0x8]
  vmovsd  QWORD PTR [rcx+rdi*8+0x8],xmm0
  vmovsd  xmm0,QWORD PTR [rdx+rdi*8+0x10]
  vmulsd  xmm0,xmm0,QWORD PTR [rcx+rdi*8+0x10]
  vaddsd  xmm0,xmm0,QWORD PTR [rsi+rdi*8+0x10]
  vmovsd  QWORD PTR [rcx+rdi*8+0x10],xmm0
  vmovsd  xmm0,QWORD PTR [rdx+rdi*8+0x18]
  vmulsd  xmm0,xmm0,QWORD PTR [rcx+rdi*8+0x18]
  vaddsd  xmm0,xmm0,QWORD PTR [rsi+rdi*8+0x18]
  vmovsd  QWORD PTR [rcx+rdi*8+0x18],xmm0
  add     rdi,0x4
  cmp     rax,rdi
  jne     120 <DAXPY(int, double*, double*, double*)
+0x120>
  vzeroupper
  ret
  nop     DWORD PTR [rax+0x0]
main:
  push    r14
  push    rbx
  push    rax
  mov     edi,0x2000
  call    19e <main+0xe>
      R_X86_64_PLT32 operator new[](unsigned
long)-0x4
  mov     rbx,rax
  mov     edi,0x2000
  call    1ab <main+0x1b>
      R_X86_64_PLT32 operator new[](unsigned
long)-0x4
  mov     r14,rax
  mov     edi,0x2000
  call    1b8 <main+0x28>
      R_X86_64_PLT32 operator new[](unsigned
long)-0x4
  mov     edi,0x400
  mov     rsi,rbx
  mov     rdx,r14
  mov     rcx,rax
  call    1cb <main+0x3b>
      R_X86_64_PLT32 DAXPY(int, double*, double*,
double*)-0x4
  xor     eax,eax
  add     rsp,0x8
  pop     rbx
  pop     r14
  ret
```
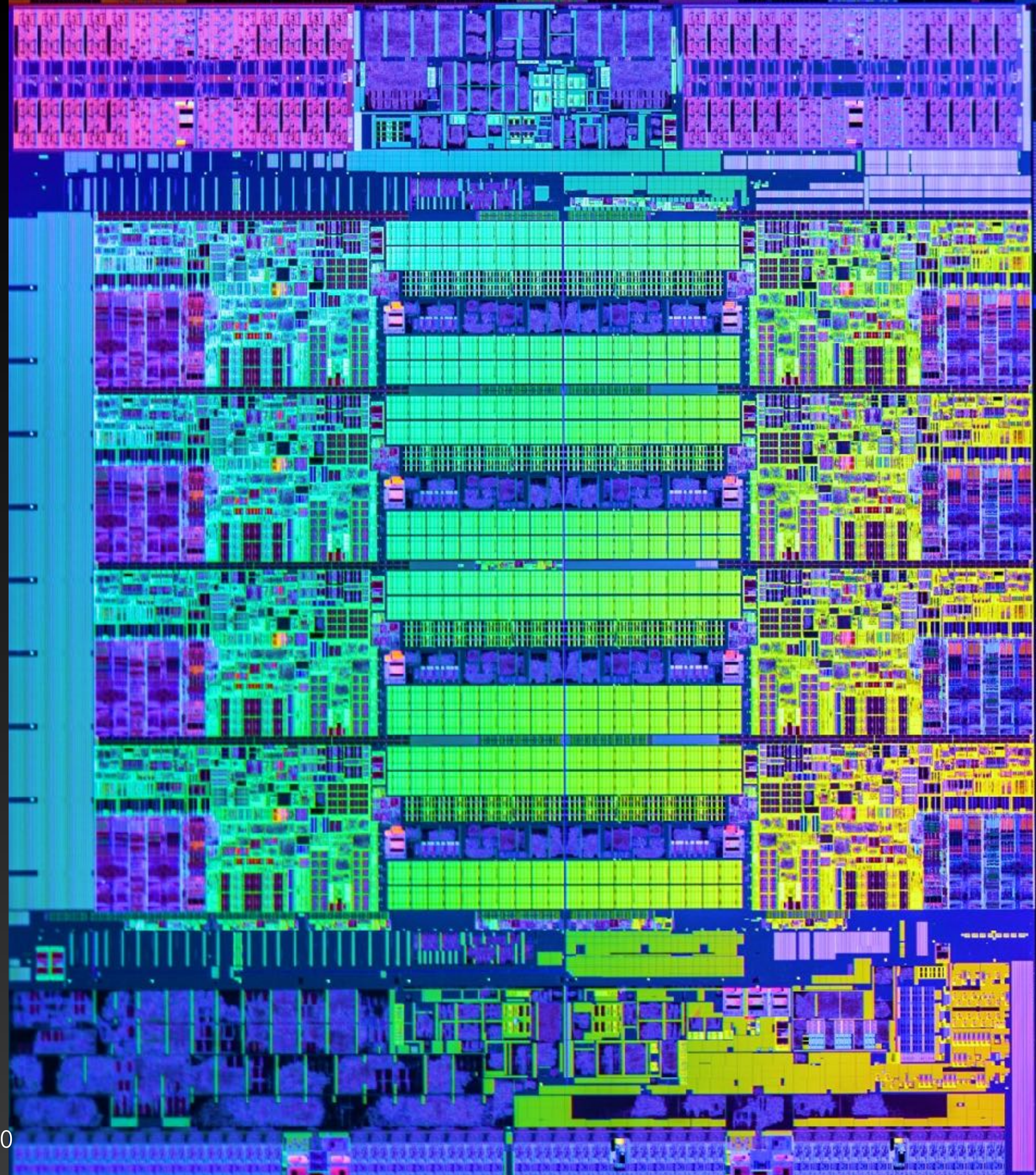
AVX(2) ~2010
"Advanced" SIMD 2 (electric Boogaloo)

- 256 bit vectors

- 6.5 data types:

  - 4* FP64

  - 8* FP32

  - 4* 64 bit ints

  - 8* 32 bit ints

  - 16* 16 bit ints

  - 32* 8 bit ints

  - Storage and conversion to IEEE FP16

- FMA instructions released separately, allow for single instruction AXPY

AVX2
Perform DAXPY z[]=(x[]+y[])*z[]

```c
// function to add the elements of two arrays
void DAXPY(int n, double *x, double *y,double *z)
{

    int i = 0;
  do {z[i] = (x[i] + y[i]) * z[i];
        i++;}
            while(i<n);

}


int main(void)
{

   int N = 1<<10; // ~1000 elements


   double *x = new double[N];
   double *y = new double[N];
   double *z = new double[N];


   // Run kernel on 1K elements
   DAXPY(N, x, y, z);


   return 0;
}
```

```asm
DAXPY(int, double*, double*, double*):
  cmp     edi,0x2
  mov     eax,0x1
  cmovge  eax,edi
  cmp     eax,0x10
  jae     17 <DAXPY(int, double*, double*, double*)
+0x17>
  xor     edi,edi
  jmp     de <DAXPY(int, double*, double*, double*)
+0xde>
  lea     rdi,[rcx+rax*8]
  lea     r8,[rsi+rax*8]
  cmp     r8,rcx
  seta    r10b
  lea     r8,[rdx+rax*8]
  cmp     rdi,rsi
  seta    r11b
  cmp     r8,rcx
  seta    r8b
  cmp     rdi,rdx
  seta    r9b
  xor     edi,edi
  test    r10b,r11b
  jne     de <DAXPY(int, double*, double*, double*)
+0xde>
  and     r8b,r9b
  jne     de <DAXPY(int, double*, double*, double*)
+0xde>
  mov     edi,eax
  and     edi,0x7ffffff0
  xor     r8d,r8d
  xchg    ax,ax
  vmovupd ymm0,YMMWORD PTR [rdx+r8*8]
  vmovupd ymm1,YMMWORD PTR [rdx+r8*8+0x20]
  vmovupd ymm2,YMMWORD PTR [rdx+r8*8+0x40]
  vmovupd ymm3,YMMWORD PTR [rdx+r8*8+0x60]
  vmovupd ymm4,YMMWORD PTR [rcx+r8*8]
  vmovupd ymm5,YMMWORD PTR [rcx+r8*8+0x20]
  vmovupd ymm6,YMMWORD PTR [rcx+r8*8+0x40]
  vmovupd ymm7,YMMWORD PTR [rcx+r8*8+0x60]
  vfmadd213pd ymm4,ymm0,YMMWORD PTR [rsi+r8*8]
  vfmadd213pd ymm5,ymm1,YMMWORD PTR [rsi+r8*8+0x20]
  vfmadd213pd ymm6,ymm2,YMMWORD PTR [rsi+r8*8+0x40]
  vfmadd213pd ymm7,ymm3,YMMWORD PTR [rsi+r8*8+0x60]
  vmovupd YMMWORD PTR [rcx+r8*8],ymm4
  vmovupd YMMWORD PTR [rcx+r8*8+0x20],ymm5
  vmovupd YMMWORD PTR [rcx+r8*8+0x40],ymm6
  vmovupd YMMWORD PTR [rcx+r8*8+0x60],ymm7
  add     r8,0x10
  cmp     rdi,r8
  jne     60 <DAXPY(int, double*, double*, double*)
+0x60>
  cmp     rdi,rax
  je      189 <DAXPY(int, double*, double*, double*)
+0x189>
  mov     r8,rdi
  not     r8
  add     r8,rax
  mov     r9,rax
  and     r9,0x3
  je      10d <DAXPY(int, double*, double*, double*)
+0x10d>
  vmovsd xmm0,QWORD PTR [rdx+rdi*8]
  vmovsd xmm1,QWORD PTR [rcx+rdi*8]
  vfmadd213sd xmm1,xmm0,QWORD PTR [rsi+rdi*8]
  vmovsd QWORD PTR [rcx+rdi*8],xmm1
  inc     rdi
  dec     r9
  jne     f0 <DAXPY(int, double*, double*, double*)
+0xf0>
  cmp     r8,0x3
  jb      189 <DAXPY(int, double*, double*, double*)
+0x189>
  data16 data16 data16 cs nop WORD PTR
[rax+rax*1+0x0]
  vmovsd xmm0,QWORD PTR [rdx+rdi*8]
  vmovsd xmm1,QWORD PTR [rcx+rdi*8]
  vmovsd xmm2,QWORD PTR [rcx+rdi*8+0x8]
  vfmadd213sd xmm1,xmm0,QWORD PTR [rsi+rdi*8]
  vmovsd QWORD PTR [rcx+rdi*8],xmm1
  vmovsd xmm0,QWORD PTR [rdx+rdi*8+0x8]
  vfmadd213sd xmm0,xmm2,QWORD PTR [rsi+rdi*8+0x8]
  vmovsd QWORD PTR [rcx+rdi*8+0x8],xmm0
  vmovsd xmm0,QWORD PTR [rdx+rdi*8+0x10]
  vmovsd xmm1,QWORD PTR [rcx+rdi*8+0x10]
  vfmadd213sd xmm1,xmm0,QWORD PTR [rsi+rdi*8+0x10]
  vmovsd QWORD PTR [rcx+rdi*8+0x10],xmm1
  vmovsd xmm0,QWORD PTR [rdx+rdi*8+0x18]
  vmovsd xmm1,QWORD PTR [rcx+rdi*8+0x18]
  vfmadd213sd xmm1,xmm0,QWORD PTR [rsi+rdi*8+0x18]
  vmovsd QWORD PTR [rcx+rdi*8+0x18],xmm1
  add     rdi,0x4
  cmp     rax,rdi
  jne     120 <DAXPY(int, double*, double*, double*)
+0x120>
  vzeroupper
  ret
  nop     DWORD PTR [rax]
main:
  push    r14
  push    rbx
  push    rax
  mov     edi,0x2000
  call    19e <main+0xe>
      R_X86_64_PLT32 operator new[](unsigned
long)-0x4
  mov     rbx,rax
  mov     edi,0x2000
  call    1ab <main+0x1b>
      R_X86_64_PLT32 operator new[](unsigned
long)-0x4
  mov     r14,rax
  mov     edi,0x2000
  call    1b8 <main+0x28>
      R_X86_64_PLT32 operator new[](unsigned
long)-0x4
  mov     edi,0x400
  mov     rsi,rbx
  mov     rdx,r14
  mov     rcx,rax
  call    1cb <main+0x3b>
      R_X86_64_PLT32 DAXPY(int, double*, double*,
double*)-0x4
  xor     eax,eax
  add     rsp,0x8
  pop     rbx
  pop     r14
  ret
```

AVX

Perform DAXPY z[]=(x[]+y[])*z[]

```asm
DAXPY(int, double*, double*, double*):
  cmp    edi,0x2
  mov    eax,0x1
  cmovge eax,edi
  cmp    eax,0x10
  jae    17 <DAXPY(int, double*, double*, double*)
+0x17>
  xor    edi,edi
  jmp    de <DAXPY(int, double*, double*, double*)
+0xde>
  lea    rdi,[rcx+rax*8]
  lea    r8,[rsi+rax*8]
  cmp    r8,rcx
  seta   r10b
  lea    r8,[rdx+rax*8]
  cmp    rdi,rsi
  seta   r11b
  cmp    r8,rcx
  seta   r8b
  cmp    rdi,rdx
  seta   r9b
  xor    edi,edi
  test   r10b,r11b
  jne    de <DAXPY(int, double*, double*, double*)
+0xde>
  and    r8b,r9b
  jne    de <DAXPY(int, double*, double*, double*)
+0xde>
  mov    edi,eax
  and    edi,0x7ffffff0
  xor    r8d,r8d
  xchg   ax,ax
  vmovupd ymm0,YMMWORD PTR [rdx+r8*8]
  vmovupd ymm1,YMMWORD PTR [rdx+r8*8+0x20]
  vmovupd ymm2,YMMWORD PTR [rdx+r8*8+0x40]
  vmovupd ymm3,YMMWORD PTR [rdx+r8*8+0x60]
  vmulpd ymm0,ymm0,YMMWORD PTR [rcx+r8*8]
  vaddpd ymm0,ymm0,YMMWORD PTR [rsi+r8*8]
  vmulpd ymm1,ymm1,YMMWORD PTR [rcx+r8*8+0x20]
  vaddpd ymm1,ymm1,YMMWORD PTR [rsi+r8*8+0x20]
  vmulpd ymm2,ymm2,YMMWORD PTR [rcx+r8*8+0x40]
  vaddpd ymm2,ymm2,YMMWORD PTR [rsi+r8*8+0x40]
  vmulpd ymm3,ymm3,YMMWORD PTR [rcx+r8*8+0x60]
  vaddpd ymm3,ymm3,YMMWORD PTR [rsi+r8*8+0x60]
  vmovupd YMMWORD PTR [rcx+r8*8],ymm0
  vmovupd YMMWORD PTR [rcx+r8*8+0x20],ymm1
  vmovupd YMMWORD PTR [rcx+r8*8+0x40],ymm2
  vmovupd YMMWORD PTR [rcx+r8*8+0x60],ymm3
  add    r8,0x10
  cmp    rdi,r8
  jne    60 <DAXPY(int, double*, double*, double*)
+0x60>
  cmp    rdi,rax
  je     185 <DAXPY(int, double*, double*, double*)
+0x185>
  mov    r8,rdi
  not    r8
  add    r8,rax
  mov    r9,rax
  and    r9,0x3
  je     10c <DAXPY(int, double*, double*, double*)
+0x10c>
  vmovsd xmm0,QWORD PTR [rdx+rdi*8]
  vmulsd xmm0,xmm0,QWORD PTR [rcx+rdi*8]
  vaddsd xmm0,xmm0,QWORD PTR [rsi+rdi*8]

  vmovsd QWORD PTR [rcx+rdi*8],xmm0
  inc    rdi
  dec    r9
  jne    f0 <DAXPY(int, double*, double*, double*)
+0xf0>
  cmp    r8,0x3
  jb     185 <DAXPY(int, double*, double*, double*)
+0x185>
  data16 data16 data16 data16 cs nop WORD PTR
[rax+rax*1+0x0]
  vmovsd xmm0,QWORD PTR [rdx+rdi*8]
  vmulsd xmm0,xmm0,QWORD PTR [rcx+rdi*8]
  vaddsd xmm0,xmm0,QWORD PTR [rsi+rdi*8]
  vmovsd QWORD PTR [rcx+rdi*8],xmm0
  vmovsd xmm0,QWORD PTR [rdx+rdi*8+0x8]
  vmulsd xmm0,xmm0,QWORD PTR [rcx+rdi*8+0x8]
  vaddsd xmm0,xmm0,QWORD PTR [rsi+rdi*8+0x8]
  vmovsd QWORD PTR [rcx+rdi*8+0x8],xmm0
  vmovsd xmm0,QWORD PTR [rdx+rdi*8+0x10]
  vmulsd xmm0,xmm0,QWORD PTR [rcx+rdi*8+0x10]
  vaddsd xmm0,xmm0,QWORD PTR [rsi+rdi*8+0x10]
  vmovsd QWORD PTR [rcx+rdi*8+0x10],xmm0
  vmovsd xmm0,QWORD PTR [rdx+rdi*8+0x18]
  vmulsd xmm0,xmm0,QWORD PTR [rcx+rdi*8+0x18]
  vaddsd xmm0,xmm0,QWORD PTR [rsi+rdi*8+0x18]
  vmovsd QWORD PTR [rcx+rdi*8+0x18],xmm0
  add    rdi,0x4
  cmp    rax,rdi
  jne    120 <DAXPY(int, double*, double*, double*)
+0x120>
  vzeroupper
  ret
  nop    DWORD PTR [rax+0x0]
main:
  push   r14
  push   rbx
  push   rax
  mov    edi,0x2000
  call   19e <main+0xe>
    R_X86_64_PLT32 operator new[](unsigned long)-0x4
  mov    rbx,rax
  mov    edi,0x2000
  call   1ab <main+0x1b>
    R_X86_64_PLT32 operator new[](unsigned long)-0x4
  mov    r14,rax
  mov    edi,0x2000
  call   1b8 <main+0x28>
    R_X86_64_PLT32 operator new[](unsigned long)-0x4
  mov    edi,0x400
  mov    rsi,rbx
  mov    rdx,r14
  mov    rcx,rax
  call   1cb <main+0x3b>
    R_X86_64_PLT32 DAXPY(int, double*, double*,
double*)-0x4
  xor    eax,eax
  add    rsp,0x8
  pop    rbx
  pop    r14
  ret
```

AVX2+FMA

Perform DAXPY z[]=(x[]+y[])*z[]

```asm
DAXPY(int, double*, double*, double*):
  cmp    edi,0x2
  mov    eax,0x1
  cmovge eax,edi
  cmp    eax,0x10
  jae    17 <DAXPY(int, double*, double*, double*)
+0x17>
  xor    edi,edi
  jmp    de <DAXPY(int, double*, double*, double*)
+0xde>
  lea    rdi,[rcx+rax*8]
  lea    r8,[rsi+rax*8]
  cmp    r8,rcx
  seta   r10b
  lea    r8,[rdx+rax*8]
  cmp    rdi,rsi
  seta   r11b
  cmp    r8,rcx
  seta   r8b
  cmp    rdi,rdx
  seta   r9b
  xor    edi,edi
  test   r10b,r11b
  jne    de <DAXPY(int, double*, double*, double*)
+0xde>
  and    r8b,r9b
  jne    de <DAXPY(int, double*, double*, double*)
+0xde>
  mov    edi,eax
  and    edi,0x7ffffff0
  xor    r8d,r8d
  xchg   ax,ax
  vmovupd ymm0,YMMWORD PTR [rdx+r8*8]
  vmovupd ymm1,YMMWORD PTR [rdx+r8*8+0x20]
  vmovupd ymm2,YMMWORD PTR [rdx+r8*8+0x40]
  vmovupd ymm3,YMMWORD PTR [rdx+r8*8+0x60]
  vmovupd ymm4,YMMWORD PTR [rcx+r8*8]
  vmovupd ymm5,YMMWORD PTR [rcx+r8*8+0x20]
  vmovupd ymm6,YMMWORD PTR [rcx+r8*8+0x40]
  vmovupd ymm7,YMMWORD PTR [rcx+r8*8+0x60]
  vfmadd213pd ymm4,ymm0,YMMWORD PTR [rsi+r8*8]
  vfmadd213pd ymm5,ymm1,YMMWORD PTR [rsi+r8*8+0x20]
  vfmadd213pd ymm6,ymm2,YMMWORD PTR [rsi+r8*8+0x40]
  vfmadd213pd ymm7,ymm3,YMMWORD PTR [rsi+r8*8+0x60]
  vmovupd YMMWORD PTR [rcx+r8*8],ymm4
  vmovupd YMMWORD PTR [rcx+r8*8+0x20],ymm5
  vmovupd YMMWORD PTR [rcx+r8*8+0x40],ymm6
  vmovupd YMMWORD PTR [rcx+r8*8+0x60],ymm7
  add    r8,0x10
  cmp    rdi,r8
  jne    60 <DAXPY(int, double*, double*, double*)
+0x60>
  cmp    rdi,rax
  je     189 <DAXPY(int, double*, double*, double*)
+0x189>
  mov    r8,rdi
  not    r8
  add    r8,rax
  mov    r9,rax
  and    r9,0x3
  je     10d <DAXPY(int, double*, double*, double*)
+0x10d>
  vmovsd xmm0,QWORD PTR [rdx+rdi*8]
  vmovsd xmm1,QWORD PTR [rcx+rdi*8]
  vfmadd213sd xmm1,xmm0,QWORD PTR [rsi+rdi*8]

  vmovsd QWORD PTR [rcx+rdi*8],xmm1
  inc    rdi
  dec    r9
  jne    f0 <DAXPY(int, double*, double*, double*)
+0xf0>
  cmp    r8,0x3
  jb     189 <DAXPY(int, double*, double*, double*)
+0x189>
  data16 data16 data16 cs nop WORD PTR [rax+rax*1+0x0]
  vmovsd xmm0,QWORD PTR [rdx+rdi*8]
  vmovsd xmm1,QWORD PTR [rcx+rdi*8]
  vmovsd xmm2,QWORD PTR [rcx+rdi*8+0x8]
  vfmadd213sd xmm1,xmm0,QWORD PTR [rsi+rdi*8]
  vmovsd QWORD PTR [rcx+rdi*8],xmm1
  vmovsd xmm0,QWORD PTR [rdx+rdi*8+0x8]
  vfmadd213sd xmm0,xmm2,QWORD PTR [rsi+rdi*8+0x8]
  vmovsd QWORD PTR [rcx+rdi*8+0x8],xmm0
  vmovsd xmm0,QWORD PTR [rdx+rdi*8+0x10]
  vmovsd xmm1,QWORD PTR [rcx+rdi*8+0x10]
  vfmadd213sd xmm1,xmm0,QWORD PTR [rsi+rdi*8+0x10]
  vmovsd QWORD PTR [rcx+rdi*8+0x10],xmm1
  vmovsd xmm0,QWORD PTR [rdx+rdi*8+0x18]
  vmovsd xmm1,QWORD PTR [rcx+rdi*8+0x18]
  vfmadd213sd xmm1,xmm0,QWORD PTR [rsi+rdi*8+0x18]
  vmovsd QWORD PTR [rcx+rdi*8+0x18],xmm1
  add    rdi,0x4
  cmp    rax,rdi
  jne    120 <DAXPY(int, double*, double*, double*)
+0x120>
  vzeroupper
  ret
  nop    DWORD PTR [rax]
main:
  push   r14
  push   rbx
  push   rax
  mov    edi,0x2000
  call   19e <main+0xe>
    R_X86_64_PLT32 operator new[](unsigned long)-0x4
  mov    rbx,rax
  mov    edi,0x2000
  call   1ab <main+0x1b>
    R_X86_64_PLT32 operator new[](unsigned long)-0x4
  mov    r14,rax
  mov    edi,0x2000
  call   1b8 <main+0x28>
    R_X86_64_PLT32 operator new[](unsigned long)-0x4
  mov    edi,0x400
  mov    rsi,rbx
  mov    rdx,r14
  mov    rcx,rax
  call   1cb <main+0x3b>
    R_X86_64_PLT32 DAXPY(int, double*, double*,
double*)-0x4
  xor    eax,eax
  add    rsp,0x8
  pop    rbx
  pop    r14
  ret
```

AVX

Perform DAXPY z[]=(x[]+y[])*z[]

```
Main Loop:
 vmovupd ymm0,YMMWORD PTR [rdx+r8*8]
 vmovupd ymm1,YMMWORD PTR [rdx+r8*8+0x20]
 vmovupd ymm2,YMMWORD PTR [rdx+r8*8+0x40]
 vmovupd ymm3,YMMWORD PTR [rdx+r8*8+0x60]
 vmulpd ymm0,ymm0,YMMWORD PTR [rcx+r8*8]
 vaddpd ymm0,ymm0,YMMWORD PTR [rsi+r8*8]
 vmulpd ymm1,ymm1,YMMWORD PTR [rcx+r8*8+0x20]
 vaddpd ymm1,ymm1,YMMWORD PTR [rsi+r8*8+0x20]
 vmulpd ymm2,ymm2,YMMWORD PTR [rcx+r8*8+0x40]
 vaddpd ymm2,ymm2,YMMWORD PTR [rsi+r8*8+0x40]
 vmulpd ymm3,ymm3,YMMWORD PTR [rcx+r8*8+0x60]
 vaddpd ymm3,ymm3,YMMWORD PTR [rsi+r8*8+0x60]
 vmovupd YMMWORD PTR [rcx+r8*8],ymm0
 vmovupd YMMWORD PTR [rcx+r8*8+0x20],ymm1
 vmovupd YMMWORD PTR [rcx+r8*8+0x40],ymm2
 vmovupd YMMWORD PTR [rcx+r8*8+0x60],ymm3
 add    r8,0x10
Residuals:
 vmovsd xmm0,QWORD PTR [rdx+rdi*8]
 vmulsd xmm0,xmm0,QWORD PTR [rcx+rdi*8]
 vaddsd xmm0,xmm0,QWORD PTR [rsi+rdi*8]
 vmovsd QWORD PTR [rcx+rdi*8],xmm0
 vmovsd xmm0,QWORD PTR [rdx+rdi*8+0x8]
 vmulsd xmm0,xmm0,QWORD PTR [rcx+rdi*8+0x8]
 vaddsd xmm0,xmm0,QWORD PTR [rsi+rdi*8+0x8]
 vmovsd QWORD PTR [rcx+rdi*8+0x8],xmm0
 vmovsd xmm0,QWORD PTR [rdx+rdi*8+0x10]
 vmulsd xmm0,xmm0,QWORD PTR [rcx+rdi*8+0x10]
 vaddsd xmm0,xmm0,QWORD PTR [rsi+rdi*8+0x10]
 vmovsd QWORD PTR [rcx+rdi*8+0x10],xmm0
 vmovsd xmm0,QWORD PTR [rdx+rdi*8+0x18]
 vmulsd xmm0,xmm0,QWORD PTR [rcx+rdi*8+0x18]
 vaddsd xmm0,xmm0,QWORD PTR [rsi+rdi*8+0x18]
 vmovsd QWORD PTR [rcx+rdi*8+0x18],xmm0
 add    rdi,0x4
 cmp    rax,rdi
 jne    120 <DAXPY(int, double*, double*, double*)+0x120>
 vzeroupper
 ret
```

AVX2+FMA

Perform DAXPY z[]=(x[]+y[])*z[]

```
Main Loop:
 vmovupd ymm0,YMMWORD PTR [rdx+r8*8]
 vmovupd ymm1,YMMWORD PTR [rdx+r8*8+0x20]
 vmovupd ymm2,YMMWORD PTR [rdx+r8*8+0x40]
 vmovupd ymm3,YMMWORD PTR [rdx+r8*8+0x60]
 vmovupd ymm4,YMMWORD PTR [rcx+r8*8]
 vmovupd ymm5,YMMWORD PTR [rcx+r8*8+0x20]
 vmovupd ymm6,YMMWORD PTR [rcx+r8*8+0x40]
 vmovupd ymm7,YMMWORD PTR [rcx+r8*8+0x60]
 vfmadd213pd ymm4,ymm0,YMMWORD PTR [rsi+r8*8]
 vfmadd213pd ymm5,ymm1,YMMWORD PTR [rsi+r8*8+0x20]
 vfmadd213pd ymm6,ymm2,YMMWORD PTR [rsi+r8*8+0x40]
 vfmadd213pd ymm7,ymm3,YMMWORD PTR [rsi+r8*8+0x60]
 vmovupd YMMWORD PTR [rcx+r8*8],ymm4
 vmovupd YMMWORD PTR [rcx+r8*8+0x20],ymm5
 vmovupd YMMWORD PTR [rcx+r8*8+0x40],ymm6
 vmovupd YMMWORD PTR [rcx+r8*8+0x60],ymm7
Residuals:
vmovsd xmm0,QWORD PTR [rdx+rdi*8]
 vmovsd xmm1,QWORD PTR [rcx+rdi*8]
 vmovsd xmm2,QWORD PTR [rcx+rdi*8+0x8]
 vfmadd213sd xmm1,xmm0,QWORD PTR [rsi+rdi*8]
 vmovsd QWORD PTR [rcx+rdi*8],xmm1
 vmovsd xmm0,QWORD PTR [rdx+rdi*8+0x8]
 vfmadd213sd xmm0,xmm2,QWORD PTR [rsi+rdi*8+0x8]
 vmovsd QWORD PTR [rcx+rdi*8+0x8],xmm0
 vmovsd xmm0,QWORD PTR [rdx+rdi*8+0x10]
 vmovsd xmm1,QWORD PTR [rcx+rdi*8+0x10]
 vfmadd213sd xmm1,xmm0,QWORD PTR [rsi+rdi*8+0x10]
 vmovsd QWORD PTR [rcx+rdi*8+0x10],xmm1
 vmovsd xmm0,QWORD PTR [rdx+rdi*8+0x18]
 vmovsd xmm1,QWORD PTR [rcx+rdi*8+0x18]
 vfmadd213sd xmm1,xmm0,QWORD PTR [rsi+rdi*8+0x18]
 vmovsd QWORD PTR [rcx+rdi*8+0x18],xmm1
 add    rdi,0x4
 cmp    rax,rdi
 jne    120 <DAXPY(int, double*, double*, double*)+0x120>
 vzeroupper
 ret
```

Targets that implement all the way to AVX2 including FMA3 are considered x86-64-V3 compliant

A "generic" CPU target that includes every major CPU since ~2014

Provides the compiler with a generic target for ISA's, doesn't include any tuning/implementation data

Section 3
Quick clarifications?

Section 4
Larger Advanced FP SIMD

Section 4
AVX "3" to "AVX9"

Granite Rapids
AVX10/512

Golden Cove WS/Server

AVX512FP16

Willow Cove

VP2INTERSECT

Sunny Cove

AVX512VAES
AVX512GFNI
AVX512VBMI2
AVX512BITALG
AVX512VPCLMULQDQ

Palm Cove
AVX512IFMA
AVX512VBMI

Cascade Lake

AVX512VNNI

Cooper Lake

AVX512BF16

Skylake-SP

AVX512F
AVX512CD

AVX512BW
AVX512DQ
AVX512VL

KNL

AVX512ER
AVX512PF

KNM
AVX5124FMAPS
AVX5124VNNIW

AVX512VPOPCNTDQ

I'm ignoring Phi

For the sake of this talk, Phi and Larrabee aren't super relevant

If you're interested in what became Phi, I highly recommend Tom Forsyths talk
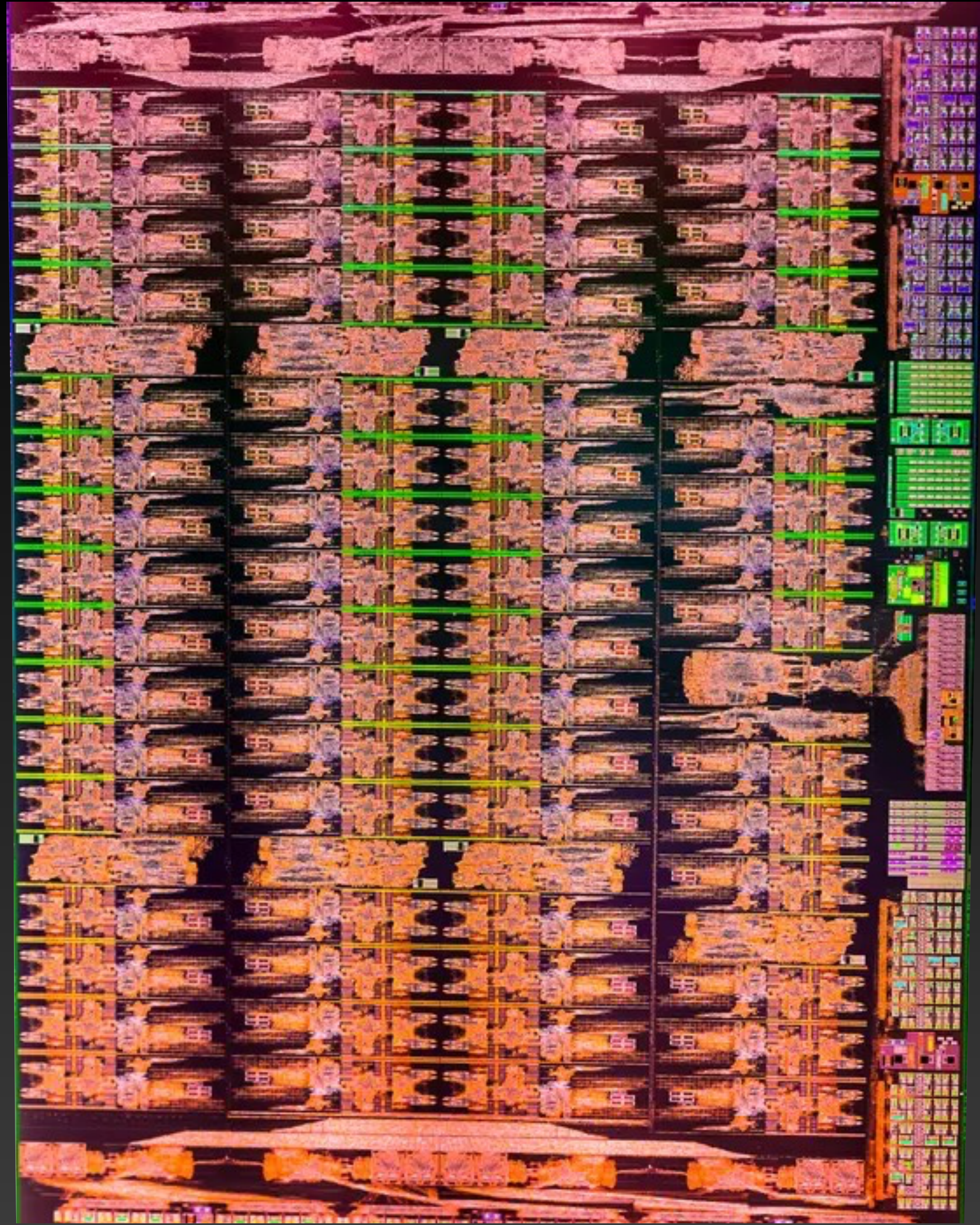"SMACNI to AVX512: The Life cycle of an instruction set".
Links are one the EasyBuild page!

AVX512 AKA AVX"3"
"Advanced" SIMD 3 (Big vectors, look at me!)

- 32x512 bit "ZMM" vector registers

- 4 data types:

  - 8* FP64

  - 16* FP32

  - 8* 64 bit ints

  - 16* 32 bit ints
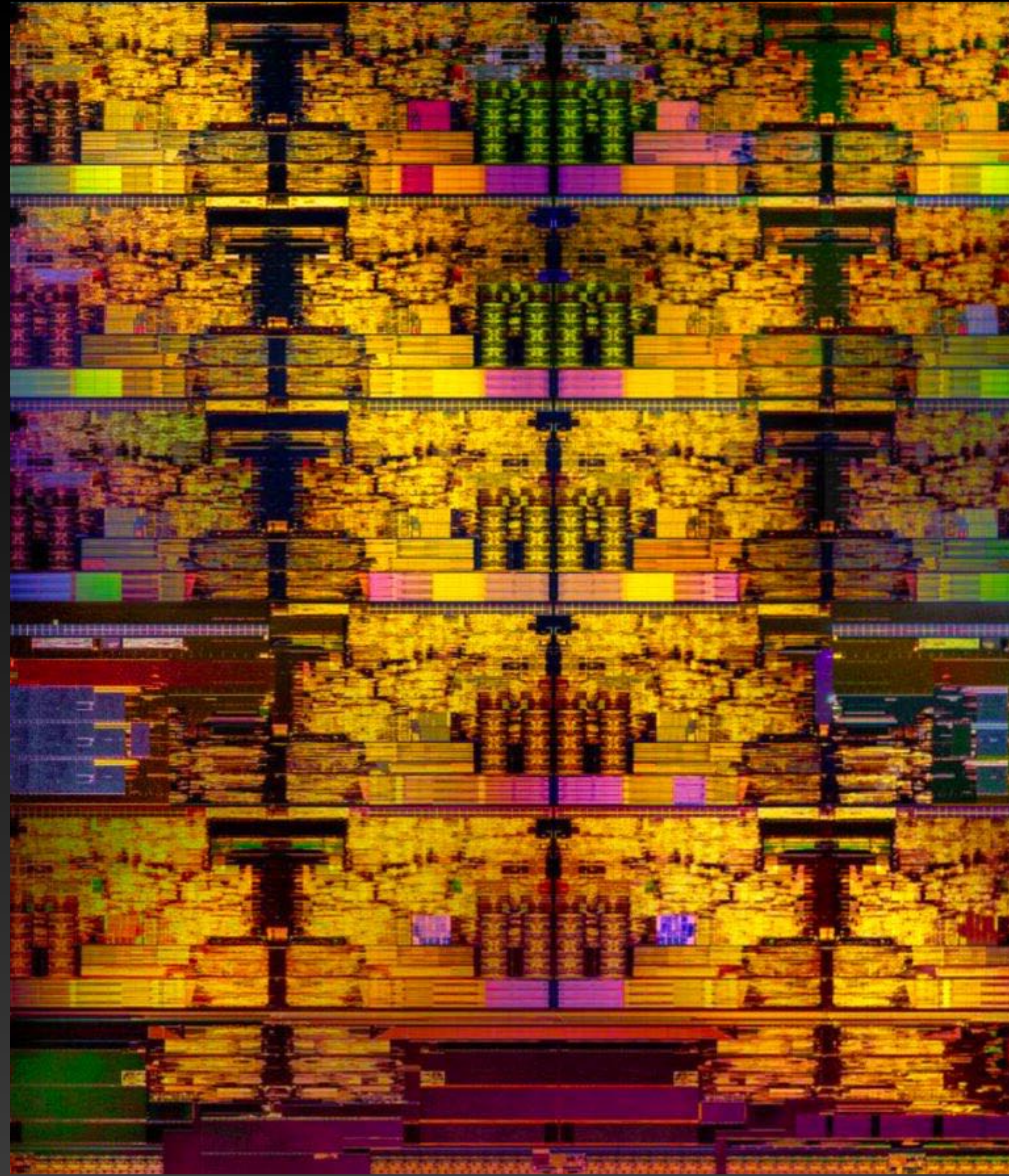
"Real" AVX512 (V4)

Variable length, Bytes/words/DWords/QWords

- 5 data types:

  - 16* 32 bit uints

  - 32* 16 bit (u)ints

  - 64 * 8 bit (u)ints

- HPC: more of the same from AVX1

40

AVX "4"
Perform DAXPY z[]=(x[]+y[])*z[]

```c
// function to add the elements of two arrays
void DAXPY(int n, double *x, double *y,double *z)
{

    int i = 0;
  do {z[i] = (x[i] + y[i]) * z[i];
        i++;}
            while(i<n);

}


int main(void)
{

   int N = 1<<10; // ~1000 elements


   double *x = new double[N];
   double *y = new double[N];
   double *z = new double[N];


   // Run kernel on 1K elements
   DAXPY(N, x, y, z);


   return 0;
}
```

```asm
DAXPY(int, double*, double*, double*):
  cmp     edi,0x2
  mov     eax,0x1
  cmovge  eax,edi
  cmp     eax,0x20
  jae     17 <DAXPY(int, double*, double*, double*)
+0x17>
  xor     edi,edi
  jmp     f2 <DAXPY(int, double*, double*, double*)
+0xf2>
  lea     rdi,[rcx+rax*8]
  lea     r8,[rsi+rax*8]
  lea     r9,[rdx+rax*8]
  cmp     r8,rcx
  seta    r10b
  cmp     rdi,rsi
  seta    r11b
  cmp     r9,rcx
  seta    r8b
  cmp     rdi,rdx
  seta    r9b
  xor     edi,edi
  test    r10b,r11b
  jne     f2 <DAXPY(int, double*, double*, double*)
+0xf2>
  and     r8b,r9b
  jne     f2 <DAXPY(int, double*, double*, double*)
+0xf2>
  mov     edi,eax
  and     edi,0x7ffffe0
  xor     r8d,r8d
  xchg    ax,ax
  vmovupd zmm0,ZMMWORD PTR [rdx+r8*8]
  vmovupd zmm1,ZMMWORD PTR [rdx+r8*8+0x40]
  vmovupd zmm2,ZMMWORD PTR [rdx+r8*8+0x80]
  vmovupd zmm3,ZMMWORD PTR [rdx+r8*8+0xc0]
  vmovupd zmm4,ZMMWORD PTR [rcx+r8*8]
  vmovupd zmm5,ZMMWORD PTR [rcx+r8*8+0x40]
  vmovupd zmm6,ZMMWORD PTR [rcx+r8*8+0x80]
  vmovupd zmm7,ZMMWORD PTR [rcx+r8*8+0xc0]
  vfmadd213pd zmm4,zmm0,ZMMWORD PTR [rsi+r8*8]
  vfmadd213pd zmm5,zmm1,ZMMWORD PTR [rsi+r8*8+0x40]
  vfmadd213pd zmm6,zmm2,ZMMWORD PTR [rsi+r8*8+0x80]
  vfmadd213pd zmm7,zmm3,ZMMWORD PTR [rsi+r8*8+0xc0]
  vmovupd ZMMWORD PTR [rcx+r8*8],zmm4
  vmovupd ZMMWORD PTR [rcx+r8*8+0x40],zmm5
  vmovupd ZMMWORD PTR [rcx+r8*8+0x80],zmm6
  vmovupd ZMMWORD PTR [rcx+r8*8+0xc0],zmm7
  add     r8,0x20
  cmp     rdi,r8
  jne     60 <DAXPY(int, double*, double*, double*)
+0x60>
  cmp     rdi,rax
  je      1a9 <DAXPY(int, double*, double*, double*)
+0x1a9>
  mov     r8,rdi
  not     r8
  add     r8,rax
  mov     r9,rax
  and     r9,0x3
  je      12d <DAXPY(int, double*, double*, double*)
+0x12d>
  data16 data16 cs nop WORD PTR [rax+rax*1+0x0]
  vmovsd  xmm0,QWORD PTR [rdx+rdi*8]
  vmovsd  xmm1,QWORD PTR [rcx+rdi*8]
  vfmadd213sd xmm1,xmm0,QWORD PTR [rsi+rdi*8]
  vmovsd  QWORD PTR [rcx+rdi*8],xmm1

  inc     rdi
  dec     r9
  jne     110 <DAXPY(int, double*, double*, double*)
+0x110>
  cmp     r8,0x3
  jb      1a9 <DAXPY(int, double*, double*, double*)
+0x1a9>
  data16 data16 data16 cs nop WORD PTR
[rax+rax*1+0x0]
  vmovsd  xmm0,QWORD PTR [rdx+rdi*8]
  vmovsd  xmm1,QWORD PTR [rcx+rdi*8]
  vmovsd  xmm2,QWORD PTR [rcx+rdi*8+0x8]
  vfmadd213sd xmm1,xmm0,QWORD PTR [rsi+rdi*8]
  vmovsd  QWORD PTR [rcx+rdi*8],xmm1
  vmovsd  xmm0,QWORD PTR [rdx+rdi*8+0x8]
  vfmadd213sd xmm0,xmm2,QWORD PTR [rsi+rdi*8+0x8]
  vmovsd  QWORD PTR [rcx+rdi*8+0x8],xmm0
  vmovsd  xmm0,QWORD PTR [rdx+rdi*8+0x10]
  vmovsd  xmm1,QWORD PTR [rcx+rdi*8+0x10]
  vfmadd213sd xmm1,xmm0,QWORD PTR [rsi+rdi*8+0x10]
  vmovsd  QWORD PTR [rcx+rdi*8+0x10],xmm1
  vmovsd  xmm0,QWORD PTR [rdx+rdi*8+0x18]
  vmovsd  xmm1,QWORD PTR [rcx+rdi*8+0x18]
  vfmadd213sd xmm1,xmm0,QWORD PTR [rsi+rdi*8+0x18]
  vmovsd  QWORD PTR [rcx+rdi*8+0x18],xmm1
  add     rdi,0x4
  cmp     rax,rdi
  jne     140 <DAXPY(int, double*, double*, double*)
+0x140>
  vzeroupper
  ret
  nop     DWORD PTR [rax]
main:
  push    r14
  push    rbx
  push    rax
  mov     edi,0x800000
  call    1be <main+0xe>
      R_X86_64_PLT32 operator new[](unsigned
long)-0x4
  mov     rbx,rax
  mov     edi,0x800000
  call    1cb <main+0x1b>
      R_X86_64_PLT32 operator new[](unsigned
long)-0x4
  mov     r14,rax
  mov     edi,0x800000
  call    1d8 <main+0x28>
      R_X86_64_PLT32 operator new[](unsigned
long)-0x4
  mov     edi,0x100000
  mov     rsi,rbx
  mov     rdx,r14
  mov     rcx,rax
  call    1eb <main+0x3b>
      R_X86_64_PLT32 DAXPY(int, double*, double*,
double*)-0x4
  xor     eax,eax
  add     rsp,0x8
  pop     rbx
  pop     r14
  ret
```

AVX512 "V5"
Integer FMA+VBMI

- 1 data type:

  - 8*52 bit integers with 104 bit internal rounding

  - Think of an integer that uses the FP64 unit; codes that can use integer maths love this one

No fun pictures, Canon lake never "really" made it to market

AVX512 "V6"
"Brain Float"

- 1 data type:

    - 32* 16 bit truncated FP32s, AKA "BF16"

- AI firms using HPC loved this one for training workloads

No fun pictures, Cooper lake had a limited installation market
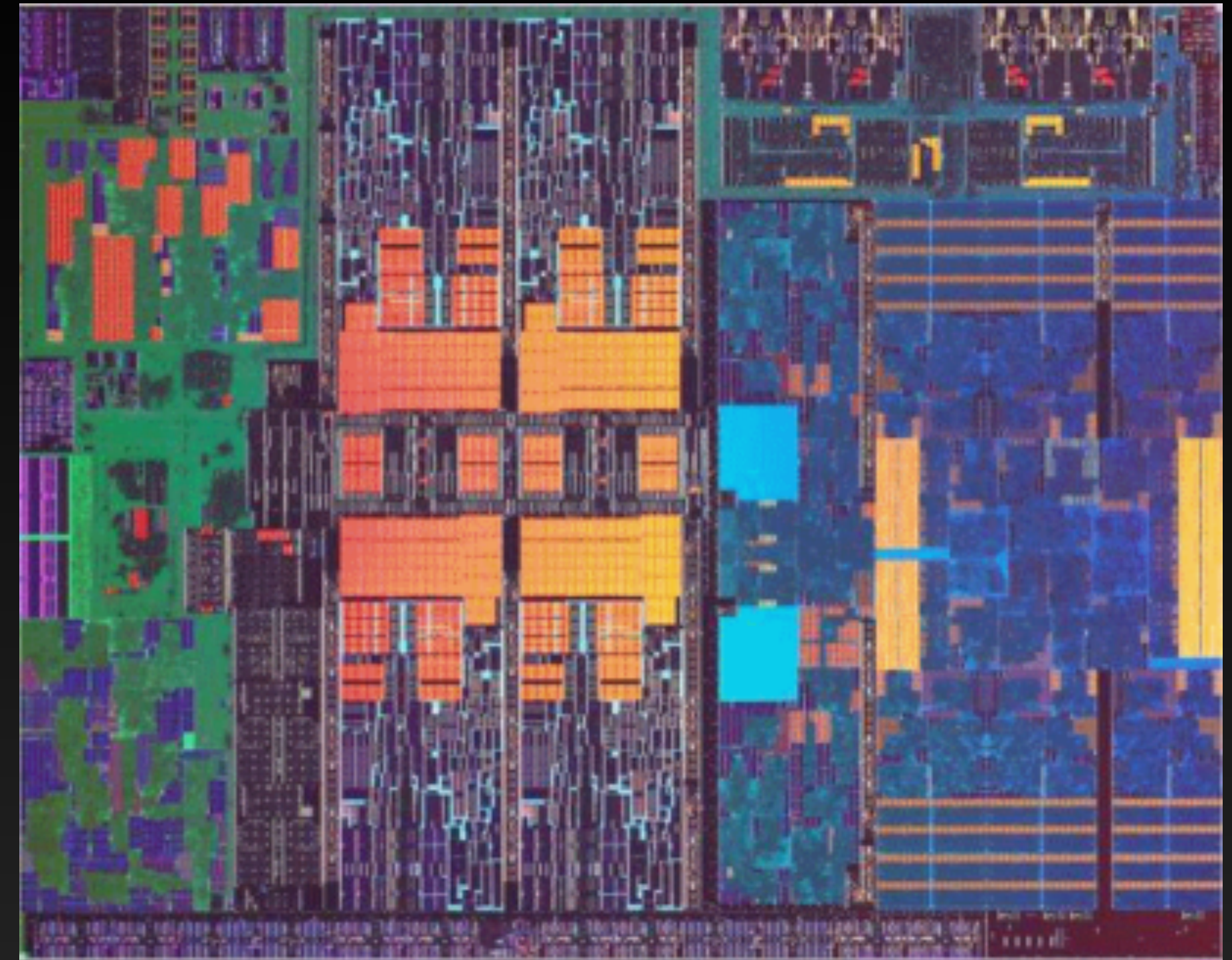
AVX512 "V7"
"VNNI, VBMI2, BITALG"

- 0 data types

- Basically integer FMA for (u)int8 and (u)int16

- AI using HPC loved this one for CPU inference

44

AVX512 "V8"
VP2INSERSECT

- 0 data types

- Micro coded Intructions to compute the intersection of 2 vectors and return a mask

- Devs proved post release it could be done faster with "basic" AVX512 instructions (bitalg, VBMI etc.)

- Has since been deprecated

@fclc@mast.hpc.social 🤖 @FelixCLC_ · Jul 19
Hey @IntelDevTools and @IntelDevTools!

Trying to confirm if we should consider the #AVX512 VP2intersect extensions as deprecated in current and future chips for SW roadmaps. Any chance I can get a confirmation?

It's been removed from future chip features in docs 😊😅

💬 3   🔁 1   ♡ 7   � 1,911   ⬆

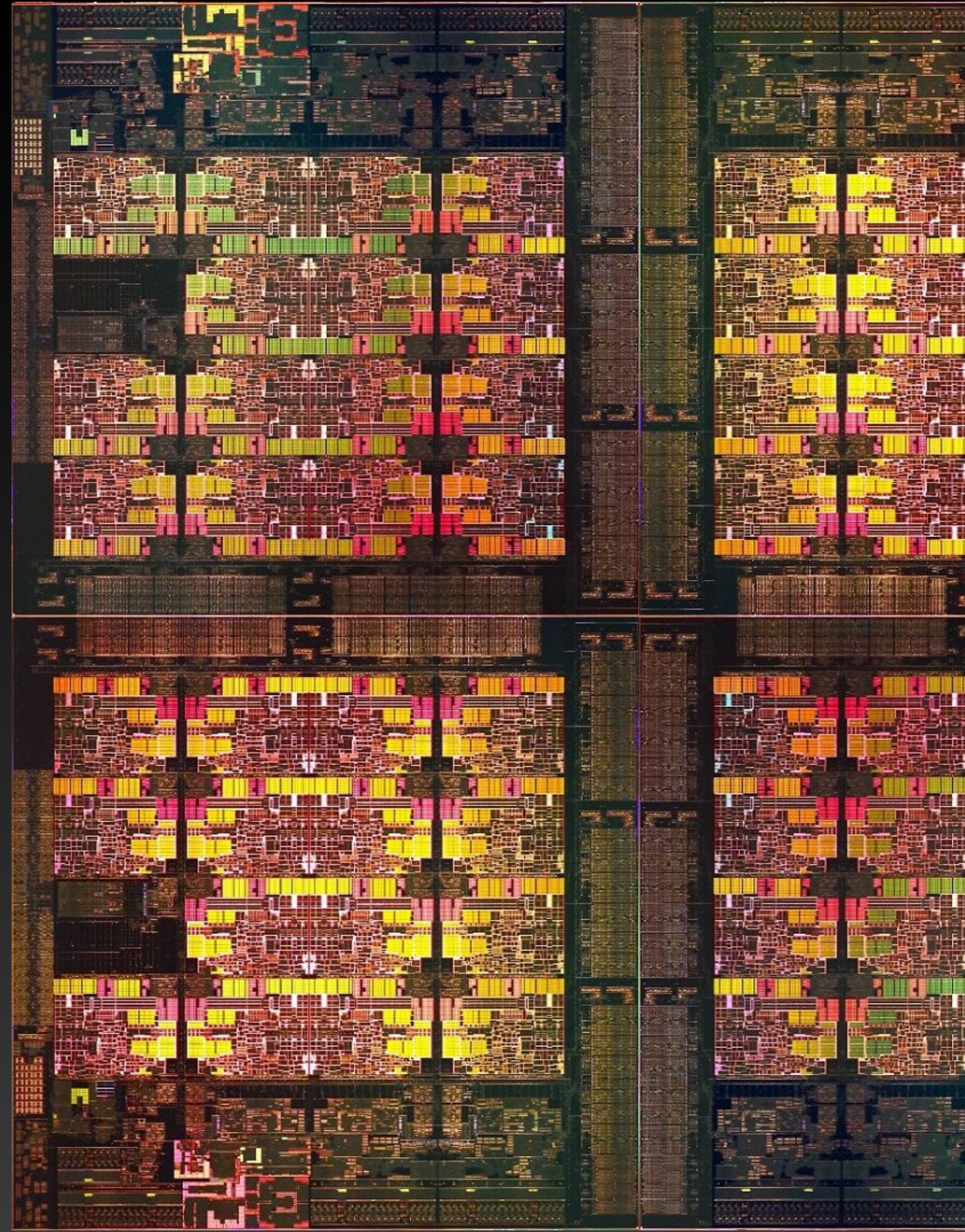Ronak Singhal
@rsinghal1

Yes. Deprecated.

AVX512 "V9"
FP16

- 1 + i data types

- IEEE754-2008 compliant Binary16

- Provides everything that FP32 and FP64 have

- Even gets its own series of complex number Intructions

- HPC: Good/great for mixed precision algorithms, includes FP16 compute on top of storage

- Expands F16C to also be able to move from FP16<->FP64 directly, instead of store f16C:
Load FP16->FP32->FP64-> math -> FP32->FP16 store

Targets that implement all the way to AVX512{F,BW,CD,DQ,VL} & are primary processors are considered x86-64-V4 compliant

All Intel HPC CPU since ~2015 (SKX)
AMD HPC Processors since ~2022 (Zen4)
For the 2 of you in the room, Centaur CNS (doesn't matter)

Really powerful data processing, lots of massive vectors

So AVX10?

Lots of talk about what AVX512 was, what in the world is AVX10?

Section 5
No more *subversioning* expectation

The goal of AVX10 is to put an end to this:

"I hope AVX512 dies a painful death, and that Intel starts fixing real problems instead of trying to create magic instructions to then create benchmarks that they can look good on.
[...]
Stop with the special-case garbage, and make all the core common stuff that *everybody* cares about run as well as you humanly can. Then do a FPU that is barely good enough on the side, and people will be happy. AVX2 is much more than enough.

Yeah, I'm grumpy. "

Linus Torvalds

And this:

```
SIMD: 512-bit
  * AVX512-F                 = Yes
    AVX512-CD                = Yes
    AVX512-PF                = No
    AVX512-ER                = No
  * AVX512-VL                = Yes
  * AVX512-BW                = Yes
  * AVX512-DQ                = Yes
    AVX512-VPOPCNTDQ         = Yes
    AVX512-4VNNIW            = No
    AVX512-4FMAPS            = No
  * AVX512-IFMA              = Yes
  * AVX512-VBMI              = Yes
    AVX512-VNNI              = Yes
  * AVX512-VBMI2             = Yes
    AVX512-BITALG            = Yes
    AVX512-VPCLMULQDQ        = Yes
    AVX512-VAES              = Yes
  * AVX512-GFNI              = Yes
    AVX512-BF16              = Yes
    AVX512-VP2INTERSECT      = Yes
    AVX512-FP16              = Yes
```

And this:

```
global GetAVX512Capability
global GetAVXCapability
global GetSSECapability
global GetAVX512fp16
global Getfp16c

section .text


; Checks CPUID for AVX capability
GetAVXCapability:
        push rbx

        mov eax, 1
        cpuid

        mov eax, ecx
        shr eax, 28              ; Read bit 28
        and eax, 1

        pop rbx
        ret


; Checks CPUID for SSE capability
GetSSECapability:
        push rbx

        mov eax, 1
        cpuid

        mov eax, edx
        shr eax, 25              ; Read bit 25
        and eax, 1

        pop rbx
        ret

; Checks CPUID for AVX512 capability
GetAVX512Capability:
        push rbx
```

```
        mov eax, 7
    xor ecx, ecx
        cpuid

        mov eax, ebx
        shr eax, 16              ; Read bit 16
        and eax, 1

        pop rbx
        ret

; Checks CPUID for AVX512-fp16 capability
GetAVX512fp16:
        push rbx

        mov eax, 7
        xor ecx, ecx
        cpuid

        mov eax, edx
        shr eax, 23              ; Read bit 23 of register edx
        and eax, 1

        pop rbx
        ret

Getfp16c:
        push rbx

        mov eax, 1
        xor ecx, ecx
        cpuid

        mov eax, ecx
        shr eax, 29              ; Read bit 29 of register ecx
        and eax, 1

        pop rbx
        ret
```

And this:

```
case 5:
    // Skylake X
    if(support_avx512_bf16())
        return CPUTYPE_COOPERLAKE;
    if(support_avx512())
        return CPUTYPE_SKYLAKEX;
    if(support_avx2())
        return CPUTYPE_HASWELL;
    if(support_avx())
    return CPUTYPE_SANDYBRIDGE;
    else
    return CPUTYPE_NEHALEM;
    case 14:
    // Skylake
    if(support_avx2())
        return CPUTYPE_HASWELL;
    if(support_avx())
    return CPUTYPE_SANDYBRIDGE;
    else
    return CPUTYPE_NEHALEM;
case 7:
    // Xeon Phi Knights Landing
    if(support_avx2())
        return CPUTYPE_HASWELL;
    if(support_avx())
    return CPUTYPE_SANDYBRIDGE;
    else
    return CPUTYPE_NEHALEM;
case 12:
    // Apollo Lake
case 15:
    // Denverton
    return CPUTYPE_NEHALEM;
}
break;
    case 6:   // family 6 exmodel 6
    switch (model) {
    case 6: // Cannon Lake
    if(support_avx512())
        return CPUTYPE_SKYLAKEX;
    if(support_avx2())
        return CPUTYPE_HASWELL;
```

```
        if(support_avx())
        return CPUTYPE_SANDYBRIDGE;
        else
        return CPUTYPE_NEHALEM;
    case 10: // Ice Lake SP
    if(support_avx512_bf16())
        return CPUTYPE_COOPERLAKE;
        if(support_avx512())
        return CPUTYPE_SKYLAKEX;
        if(support_avx2())
        return CPUTYPE_HASWELL;
        if(support_avx())
        return CPUTYPE_SANDYBRIDGE;
        else
        return CPUTYPE_NEHALEM;
        }
    break;
    case 7: // family 6 exmodel 7
    switch (model) {
    case 10: // Goldmont Plus
        return CPUTYPE_NEHALEM;
        case 14: // Ice Lake
        if(support_avx512())
        return CPUTYPE_SKYLAKEX;
        if(support_avx2())
        return CPUTYPE_HASWELL;
        if(support_avx())
        return CPUTYPE_SANDYBRIDGE;
        else
        return CPUTYPE_NEHALEM;
        }
    break;
    case 8:
    switch (model) {
    case 12: // Tiger Lake
    case 13: // Tiger Lake (11th Gen
Intel(R) Core(TM) i7-11800H @ 2.30GHz)
        if(support_avx512())
        return CPUTYPE_SKYLAKEX;
        if(support_avx2())
        return CPUTYPE_HASWELL;
        if(support_avx())
```

```
        return CPUTYPE_SANDYBRIDGE;
        else
        return CPUTYPE_NEHALEM;
    case 14: // Kaby Lake and refreshes
        if(support_avx2())
        return CPUTYPE_HASWELL;
        if(support_avx())
        return CPUTYPE_SANDYBRIDGE;
        else
        return CPUTYPE_NEHALEM;
    case 15: // Sapphire Rapids
    if(support_avx512_bf16())
        return CPUTYPE_COOPERLAKE;
        if(support_avx512())
        return CPUTYPE_SKYLAKEX;
        if(support_avx2())
        return CPUTYPE_HASWELL;
        if(support_avx())
        return CPUTYPE_SANDYBRIDGE;
        else
        return CPUTYPE_NEHALEM;
        }
    break;
    case 9:
    switch (model) {
    case 7: // Alder Lake desktop
    case 10: // Alder Lake mobile
    if(support_avx512_bf16())
        return CPUTYPE_COOPERLAKE;
        if(support_avx512())
        return CPUTYPE_SKYLAKEX;
        if(support_avx2())
        return CPUTYPE_HASWELL;
        if(support_avx())
        return CPUTYPE_SANDYBRIDGE;
        else
        return CPUTYPE_NEHALEM;
    case 13: // Ice Lake NNPI
    if(support_avx512())
        return CPUTYPE_SKYLAKEX;
        if(support_avx2())
        return CPUTYPE_HASWELL;
```

```
        if(support_avx())
        return CPUTYPE_SANDYBRIDGE;
        else
        return CPUTYPE_NEHALEM;
    case 14: // Kaby Lake and refreshes
        if(support_avx2())
        return CPUTYPE_HASWELL;
        if(support_avx())
        return CPUTYPE_SANDYBRIDGE;
        else
        return CPUTYPE_NEHALEM;
    }
    break;
    case 10: //family 6 exmodel 10
    switch (model) {
    case 5: // Comet Lake H and S
    case 6: // Comet Lake U
    if(support_avx2())
        return CPUTYPE_HASWELL;
    if(support_avx())
        return
CPUTYPE_SANDYBRIDGE;
        else
        return CPUTYPE_NEHALEM;
    case 7: // Rocket Lake
    if(support_avx512())
        return CPUTYPE_SKYLAKEX;
    if(support_avx2())
        return CPUTYPE_HASWELL;
    if(support_avx())
        return CPUTYPE_SANDYBRIDGE;
        else
        return CPUTYPE_NEHALEM;
    }
    break;
    }
    break;
```

And this:

54

I think everybody gets the point

I think everybody gets the point

If not, don't know what to tell you

So AVX10?

Answer the darn question!

AVX10 is actually AVX10.N/M

What does the .N in AVX10.N/M mean?

Simple: N is version number. Something get added? N increments

Importantly: if you have N+4, you *must* support N+3, N+2 ... N

What does the /M in AVX10.N/M mean?

Simple: M is *implementation size*
Valid Options are 512, 256, 128*
Building a small consumer CPU? Make AVX10.N/256. Smaller registers, less state, less power.
Importantly: must support all smaller sizes

What does AVX10.N/M include?

Version 1, AKA AVX10.1/M supports:
All of AVX2,AVX1, FMA, SSE1-4.2
and
All of the *instructions* from AVX512 {F, CD, VL, DQ, BW,
IFMA, VBMI, VBMI2, VPOPCNTDQ, BITALG, VNNI,
VPCLMULQDQ, GFNI, VAES, BF16, FP16}

The italics and asterix on that last slide are very 👻spooky🎃, what's up with that? Because AVX10 is meant to allow instruction support on smaller chips, it keeps the instructions from AVX512, but not the register size

Isn't that just going to make things worse?

Yes and no

It means you can run a single check and know what is and isn't supported

Does mean you need to check local implementation size

Say again?

Most of the Intructions from {AVX512} are great for commercial data centre as well as local client data processing; they don't do much for classical HPC FP64 apps

So AVX10 for HPC?

Beyond the spec, WHAT IS AVX10?!?!?

For admins and users:
A single compiler flag to enable all of the goodies in the history of AVX512

For programmers/developers/RSE's:
A pair of CPUID flags to check for kernel version targeting and implementation size

Is it out?

Yes* 😉

Section 6
AVX10 for HPC in 2023

AVX10.1/512 is coming next year in silicon
with Intels Granite Rapids Xeons

GOOD NEWS
EVERYONE!

AVX10.1/512 is the same as Sapphire Rapids and upcoming Emerald Rapids

For those with intel dev cloud or ANL/ALCF Sunspot access, your codes for SPR are the same ISA as AVX10.1/512

Section 7
That's all folks!

Brief note on AVX10/256:

AVX10/256 devices are coming soon. Personally I expect Intel in '24, AMD in '25

Brief note on AVX10/128

AVX10/128 is a silly, silly idea that should never have been allowed to exist, and that I've asked intel to remove from the spec.

Brief note on AVX10/128

I wrote an entire companion article on this topic for the publication Chips and Cheese

Link: https://chipsandcheese.com/
2023/10/11/avx10-128-is-a-silly-idea-
and-should-be-completely-removed-
from-the-specification/

Information:

Mostly active on mastodon
fclc@mast.hpc.social
Twitter (for now) @felixclc_
Email:
felix.leclair123@hotmail.com

Acknowledgements:
Everyone who provided quotes
Various CPU architects who provided guidance
Various "spec lawyers" who provided insights
Chips and Cheese for hosting the accompanying article

Acknowledgements:
Easy Build

Now the fun Part: Closed session Q and A