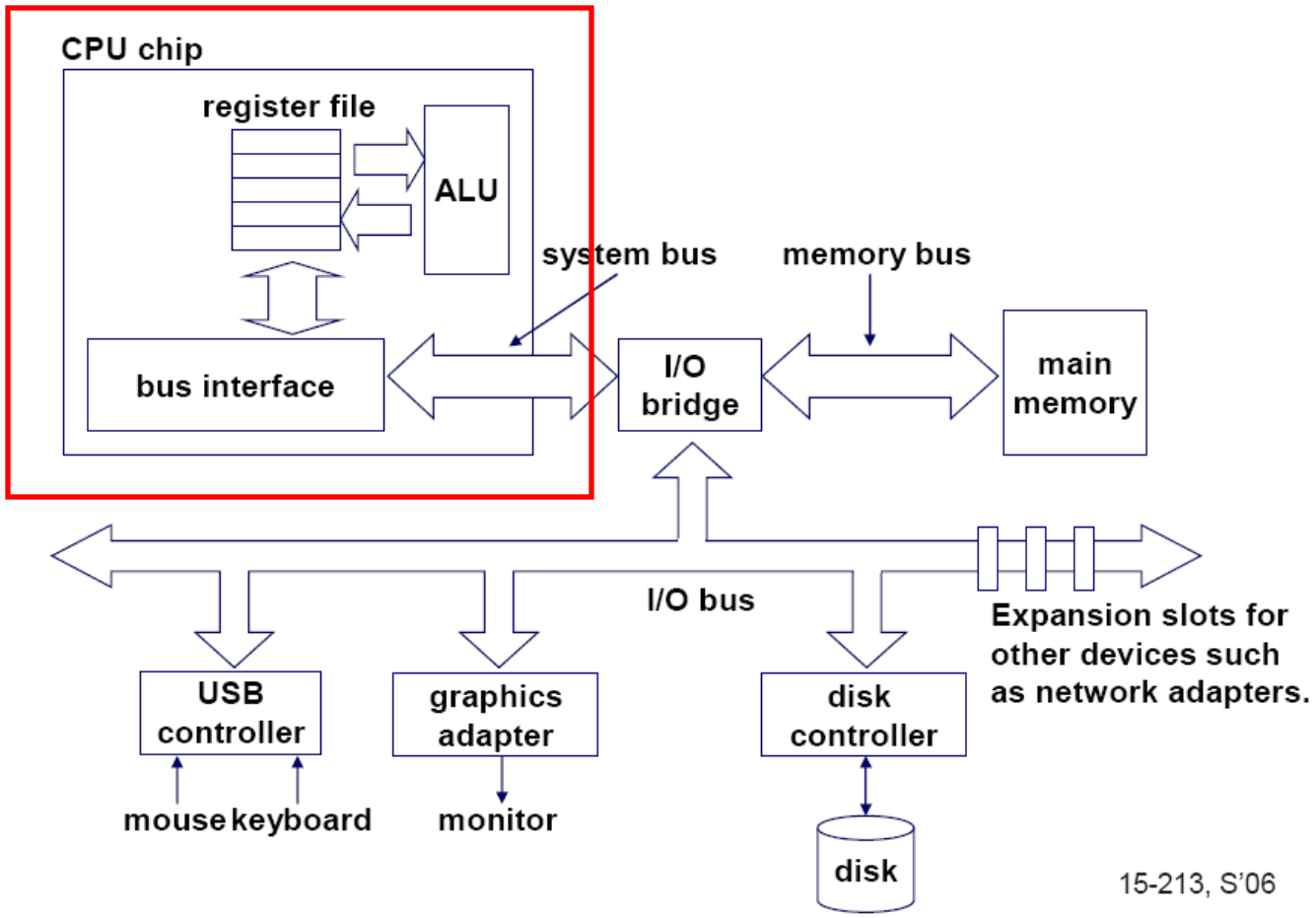
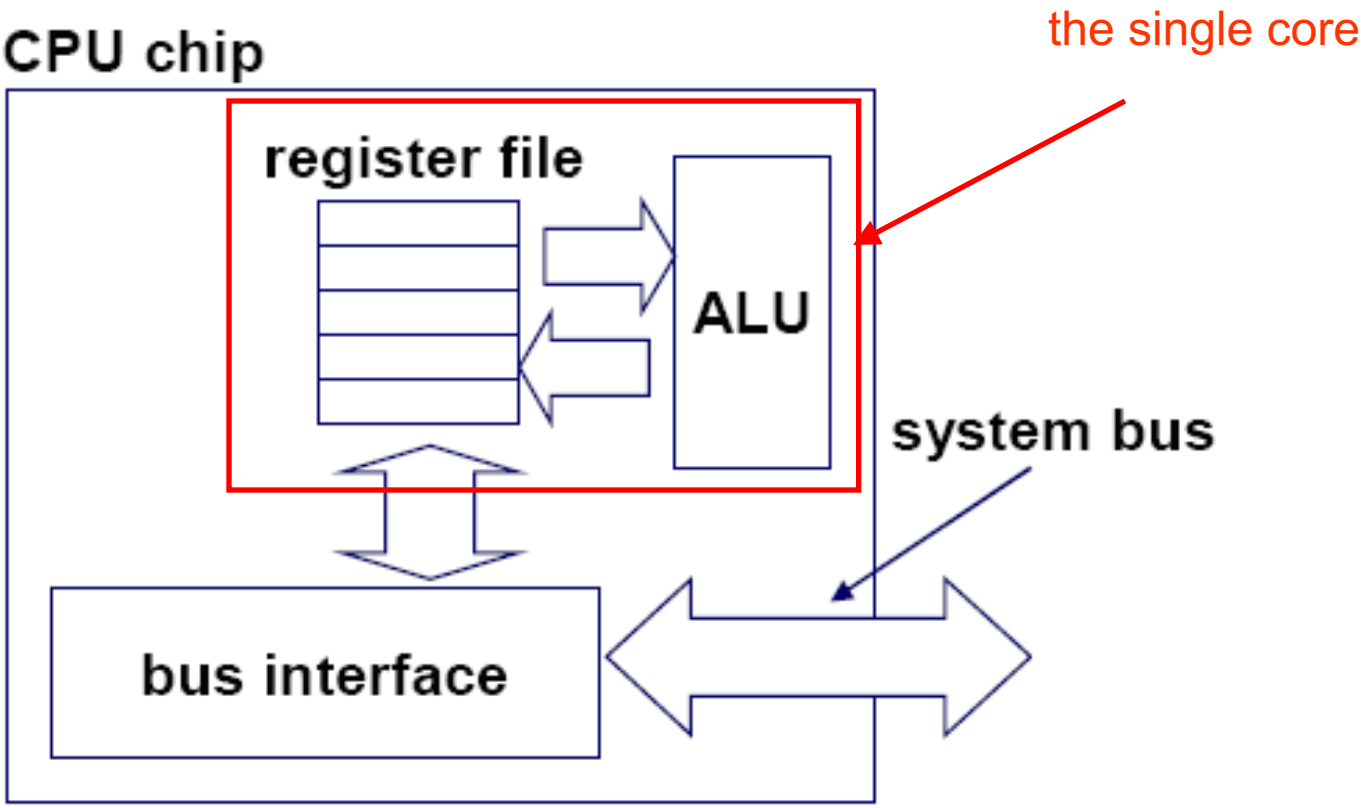
Multi-core/Many-core, GPU Architectures (1)

Single-core computer

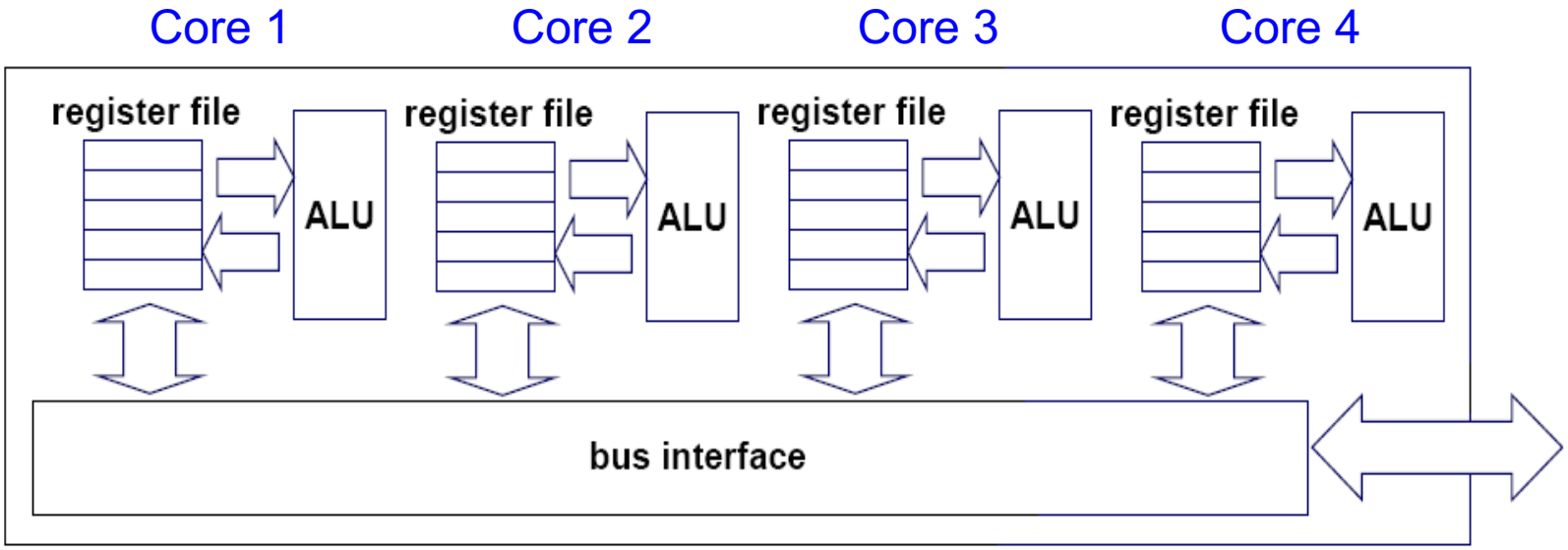


Single-core CPU chip



Multi-core architectures

- This lecture is about a new trend in computer architecture:
  Replicate multiple processor cores on a single die.



Multi-core CPU chip

Why multi-core ?

- Difficult to make single-core clock frequencies even higher
- Deeply pipelined circuits:
  - heat problems
  - speed of light problems
  - difficult design and verification
  - large design teams necessary
  - server farms need expensive air-conditioning

- Many new applications are multithreaded
- General trend in computer architecture (shift towards more parallelism)

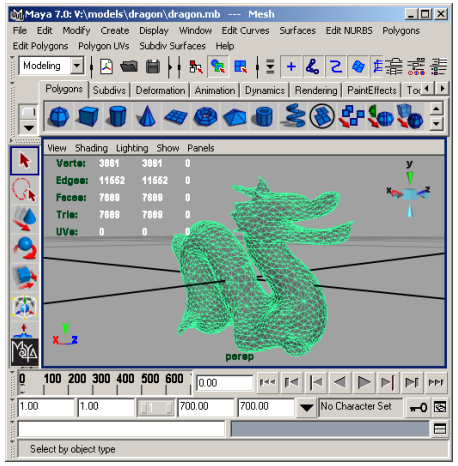
Multi-core processor is a special kind of a multiprocessor:
All processors are on the same chip

- Multi-core processors are MIMD:
  Different cores execute different threads (Multiple Instructions), operating on different parts of memory (Multiple Data).

- Multi-core is a shared memory multiprocessor:
  All cores share the same memory

What applications benefit from multi-core?

- Database servers
- Web servers (Web commerce)
- Compilers
- Multimedia applications
- Scientific applications, CAD/CAM
- In general, applications with *Thread-level parallelism* (as opposed to instruction-level parallelism)



Each can run on its own core

How to control ”Cores”

- Multi-core CPU
  - x86, Cell, SPARC
  - ”Core” is a real CPU that can work independently
  - Effective vector length is 4 (AVX2) – 8(AVX512, HPC-ACE2)
  - Cores are interconnected on a die (chip)
  - Cache / interconnect network for memory sharing
- Many-core type 1 : GPU
  - It’s SIMD. 32 – 64 cores execute same instructions
  - SIMT: Single Instruction stream Multiple Thread
  - Effective vector length is >= 128
- Many-core type 2: GRAPE-DR
  - SIMD : 512 cores execute same instructions
  - Effective vector length is 512 x 4 = 2048
- Many-core type 3 : PEZY-SC
  - MIMD : Each core can executes different instructions

SIMD: Single Instruction, Multiple Data

- Scalar processing
  - traditional mode
  - one operation produces
- SIMD processing
  - with SSE / SSE2
  - one operation produces

```
double *x, *y, *z;
for (i=0;  i<n;  i++)      z[i] = x[i] + y[i];
```

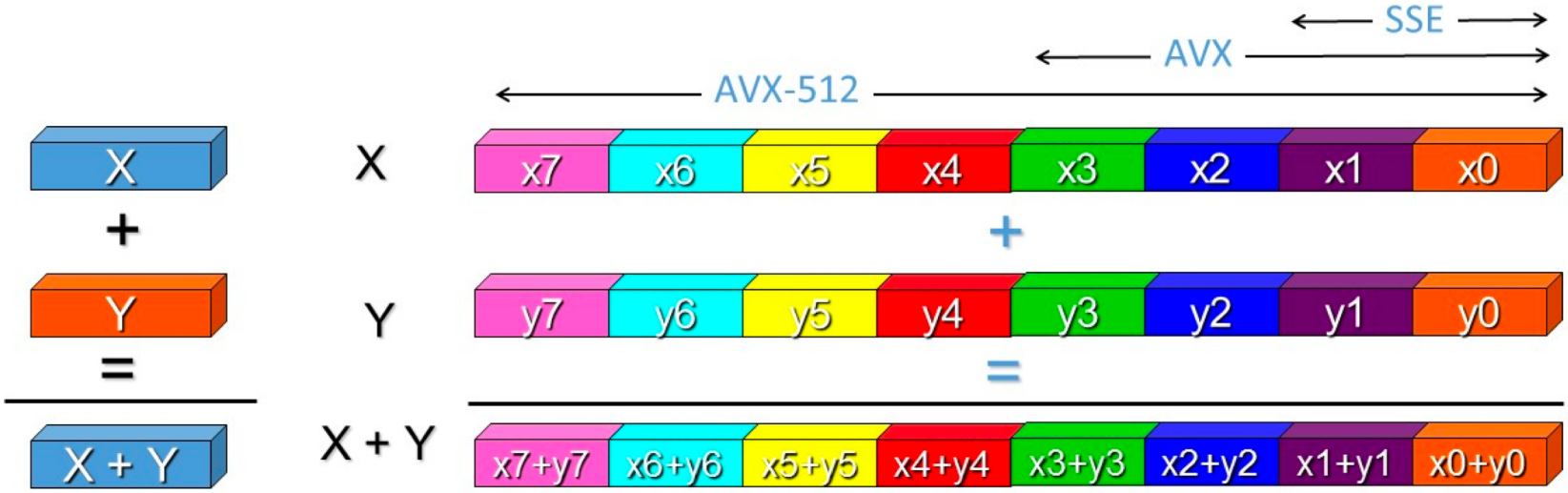


Figure 1 Scalar and vectorized loop versions with Intel® SSE, AVX and AVX-512.

Graphic Processing Units (GPU)

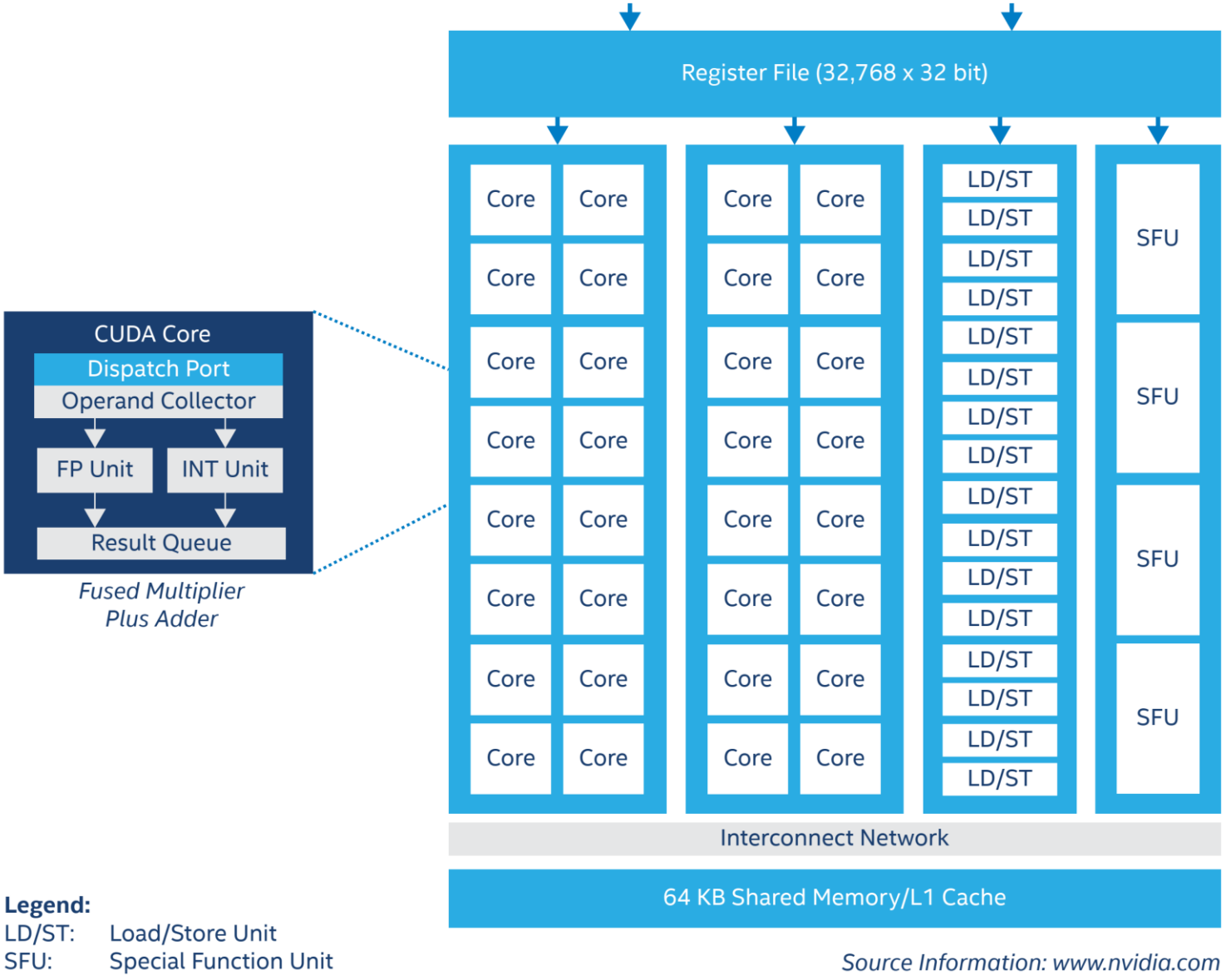


Figure 2. GP-GPU Architecture

Memory / Network on Chip(NoC)

- Multi-core CPU
  - Each core has own register file and 1st/2nd cache
  - 3rd level cache are shared by all cores
  - NoC: Ring, Mesh or Torus
- Many-core type 1 : GPU
  - Each core has own register file and 1st cache
  - Register file is very huge (32K words or more)
  - Shared memory for 32/64 cores
  - Usually have no NoC
- Many-core type 2: GRAPE-DR
  - Each core has own register file and local memory
  - Shared memory
  - Noc: Broadcast and Reduction networks

Structure of ALU and its ISA

- Multi-core CPU
  - Integer ALU & FP units/ Out of Order
  - Legacy ISA are supported in x86 CPU for 40 years
  - FP units are SIMD unit
    - SSE2(2 DP ops), AVX2(4 DP ops), AVX512(8 DP ops)
- Many-core type 1 : GPU
  - Integer ALU & FP units / In-order(?)
  - ISA is designed from scratch
  - FP units is a scalar unit (fused-multiply-add; FMA)
  - NVIDIA: Special function unit (SFU) are shared by 32 cores
- Many-core type 2: GRAPE-DR
  - Integer ALU & FP units / In-order
  - FP add and FP mul
  - No SFU

Performance of processors

- Number of cores x operations per core x clock
  - Definition of “core” is sometimes different in a context
  - FP units are usually FMA (2 operations)
  - Double-precision(DP) and single-precision(SP) performance

- x86 CPU (Haswell): CPU+GPU fused…
  - 2 (core) x 8(AVX2) x 2(FMA) x clock

特徴 [編集]

Ivy Bridgeから引き継ぐ特徴

- トライゲート22nmプロセス[9]
- 14段のパイプライン
- 1つのコアの余剰リソースを2つ目のコアに仕立てるハイパースレッディング・テクノロジー
- ターボブースト・テクノロジー

新規に確認されている特徴

CPU部

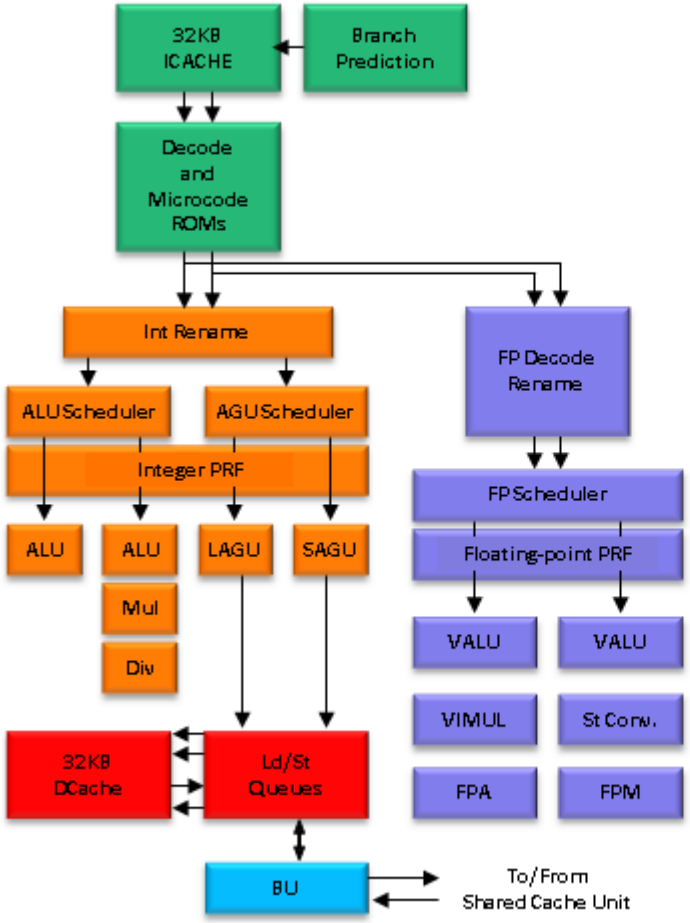
- 演算処理のためのポート数が6個から8個へ拡充[10]
- AVX2 (Advanced Vector Extension 2) のサポートによる数値演算処理性能の向上[11]
- 分岐予測の精度向上による、パイプラインストールの減少[10]
- FMA3(Fused Multiply Add)、BMI(Bit Manipulation Instruction Sets (英語版))などの拡張命令の追加[12]
- トランザクショナルメモリのハードウェアサポート
- L1データキャッシュの帯域倍増（ロード64Bytes/cycle、ストア32Bytes/cycle）
- L2キャッシュの帯域倍増(64Bytes/cycle)
- L2 TLB(Translation Lookaside Buffer)エントリ数の増加およびラージTLBのサポート[13]
- リオーダバッファのエントリ数の増加(168→192)
- 物理レジスタファイルの増加（整数160→168、浮動小数点144→168）
- リザベーションステーションのエントリ数の増加(54→60)

GPU部

Example : AMD Family 16h Processor

Software Optimization Guide for AMD Family 16h Processors

http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2012/10/SOG_16h_52128_PUB_Rev1_1.pdf



Abbreviation	Descriptive Name
Int Rename	Integer Register Rename Unit
Floating-point PRF	Floating-point Physical Register File
Integer PRF	Integer Physical Register File
FP Decode Rename	Floating-point Decode / Rename Unit
ALU	Arithmetic and Logic Unit
LAGU	Load Address Generation Unit
SAGU	Store Address Generation Unit
BU	Bus Unit
VALU	Vector ALU
VIMUL	Vector Integer Multiply Unit
St Conv.	Store / Convert Unit
FPA	Floating-point Addition Unit
FPM	Floating-point Multiplication Unit
Ld/St Queues	Load / Store Queues

Diagram Legend

Example : Integer pipeline

Overall	out-of-order, 2 wide decode/dispatch/retire.			
	6 wide issue (2 alu, 1 ld, 1 st, 2 fp)			
	ISA: AMD64, x87, mmx, SSE1, SSE2, SSE3, SSSE3, SSE4a, SSE4.1, SSE4.2, AES/CLMUL, MOVBE, AVX, F16C, BMI1			
Integer				
	Integer Exec Units:		Pipes:	Pipe0: ALU (also handle FP->INT jam)
	o ALU (x2)			Pipe1: ALU, MUL, DIV
	o AGU (LAGU, SAGU)			Pipe2: LAGU
	o MUL, DIV			Pipe3: SAGU (also handles 3-operand LEA)
	Load latencies			
		normal integer load: 3 cycles		
		normal fp load: 5 cycles		
		misaligned load: +1 cycle		
		non-0 FS,GS: +1 cycle		
		load miss D$, hit L2$: +22 cycles		

Example : Latency of Int. operations

Integer instructions

Instruction	Operands	Ops	Latency	Reciprocal throughput	Execution pipe	Notes
Arithmetic instructions						
ADD, SUB	r,r/i	1	1	0.5	I0/1	
ADD, SUB	r,m	1		1		
ADD, SUB	m,r	1	6	1		
ADC, SBB	r,r/i	1	1	1	I0/1	
ADC, SBB	r,m	1		1		
ADC, SBB	m,r/i	1	8			
CMP	r,r/i	1	1	0.5	I0/1	
CMP	r,m	1		1		
INC, DEC, NEG	r	1	1	0.5	I0/1	
INC, DEC, NEG	m	1	6	1		
MUL, IMUL	r8/m8	1	3	1	I0	
MUL, IMUL	r16/m16	3	3	3	I0	
MUL, IMUL	r32/m32	2	3	2	I0	
MUL, IMUL	r64/m64	2	6	5	I0	
IMUL	r16,r16/m16	1	3	1	I0	
IMUL	r32,r32/m32	1	3	1	I0	
IMUL	r64,r64/m64	1	6	4	I0	
IMUL	r16,(r16),i	2	4	1	I0	
IMUL	r32,(r32),i	1	3	1	I0	
IMUL	r64,(r64),i	1	6	4	I0	
DIV	r8/m8	1	11-14	11-14	I0	
DIV	r16/m16	2	12-19	12-19	I0	
DIV	r32/m32	2	12-27	12-27	I0	
DIV	r64/m64	2	12-43	12-43	I0	
IDIV	r8/m8	1	11-14	11-14	I0	
IDIV	r16/m16	2	12-19	12-19	I0	
IDIV	r32/m32	2	12-27	12-27	I0	
IDIV	r64/m64	2	12-43	12-43	I0	

Example : Latency of FP operations

Floating point XMM instructions

Instruction	Operands	Ops	Latency	Reciprocal throughput	Execution pipe	Notes
Arithmetic						
ADDSS/D SUBSS/D	x,x/m	1	3	1	FP0	
ADDPS/D SUBPS/D	x,x/m	1	3	1	FP0	
VADDPS/D VSUBPS/D	y,y/m	2	3	2	FP0	
ADDSUBPS/D	x,x/m	1	3	1	FP0	SSE3
VADDSUBPS/D	y,y/m	2	3	2	FP0	
HADD/SUBPS/D	x,x/m	1	4	1	FP0	SSE3
VHADD/SUBPS/D	y,y/m	2	4	2	FP0	
MULSS/PS	x,x/m	1	2	1	FP1	
VMULPS	y,y/m	2	2	2	FP1	
Math						
SQRTSS	x,x/m	1	16	16	FP1	
SQRTPS	x,x/m	2	21	21	FP1	
VSQRTPS	y,y/m	2	42	42	FP1	
SQRTSD	x,x/m	1	27	27	FP1	
SQRTPD	x,x/m	2	27	27	FP1	
VSQRTPD	y,y/m	2	54	54	FP1	
RSQRTSS/PS	x,x/m	1	2	1	FP1	
VRSQRTPS	y,y/m	2	2	2	FP1	

RDTSC : ReaD Time Stamp Counter

- On Intel/AMD CPUs, we can count CPU clock cycles by using the following code on Linux

```
unsigned long long t1, t2;
RDTSC(t1);
// Put you code to measure
RDTSC(t2);
printf("CPU Clock=%llu¥n", t2-t1);
```

Macro for C language

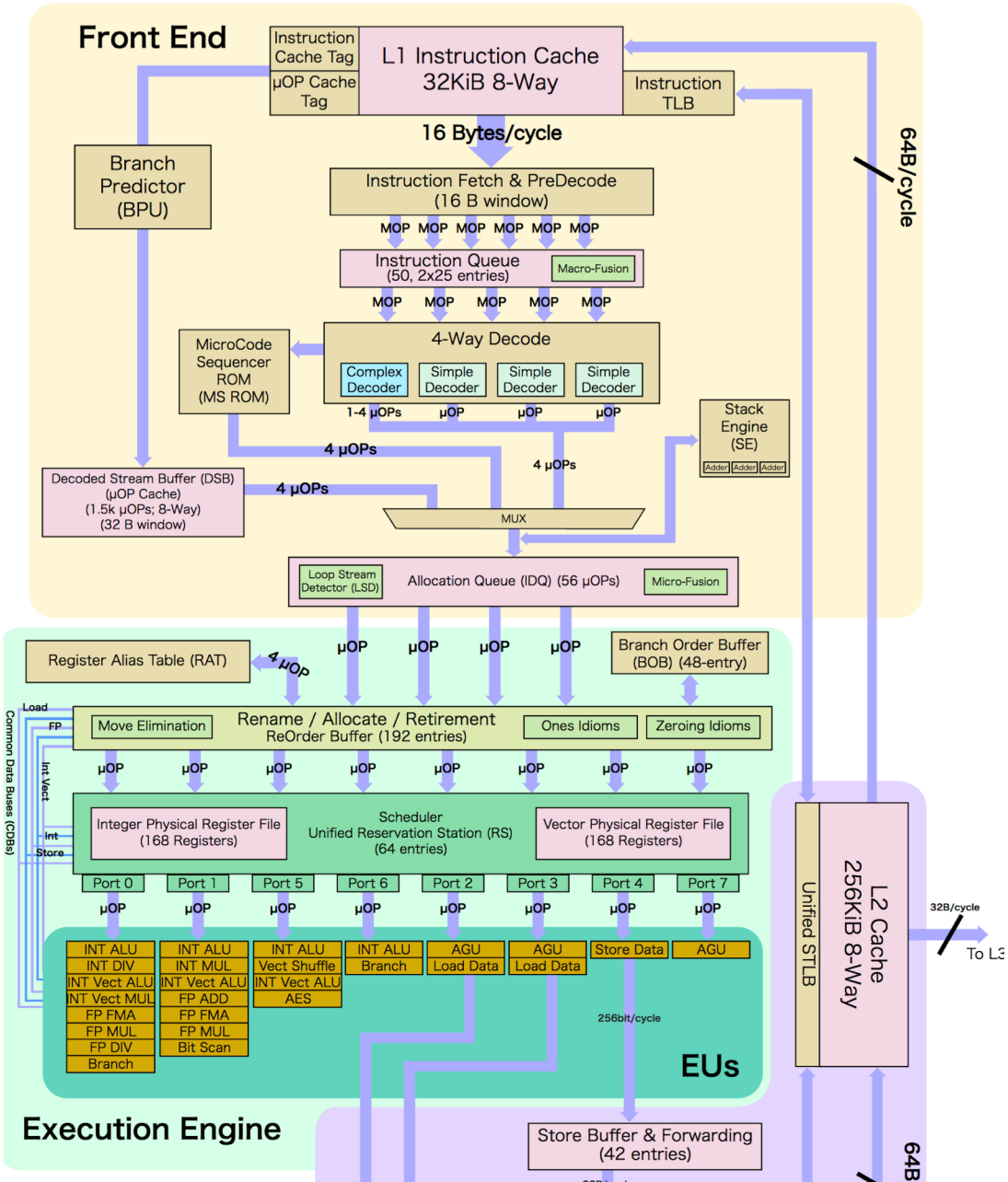
```
#define RDTSC(X)¥
   asm volatile ("rdtsc; shlq $32,%%rdx; orq %%rdx,%%rax" : "=a" (X) :: "%rdx")
```

DP operations : Latency is 4 clock

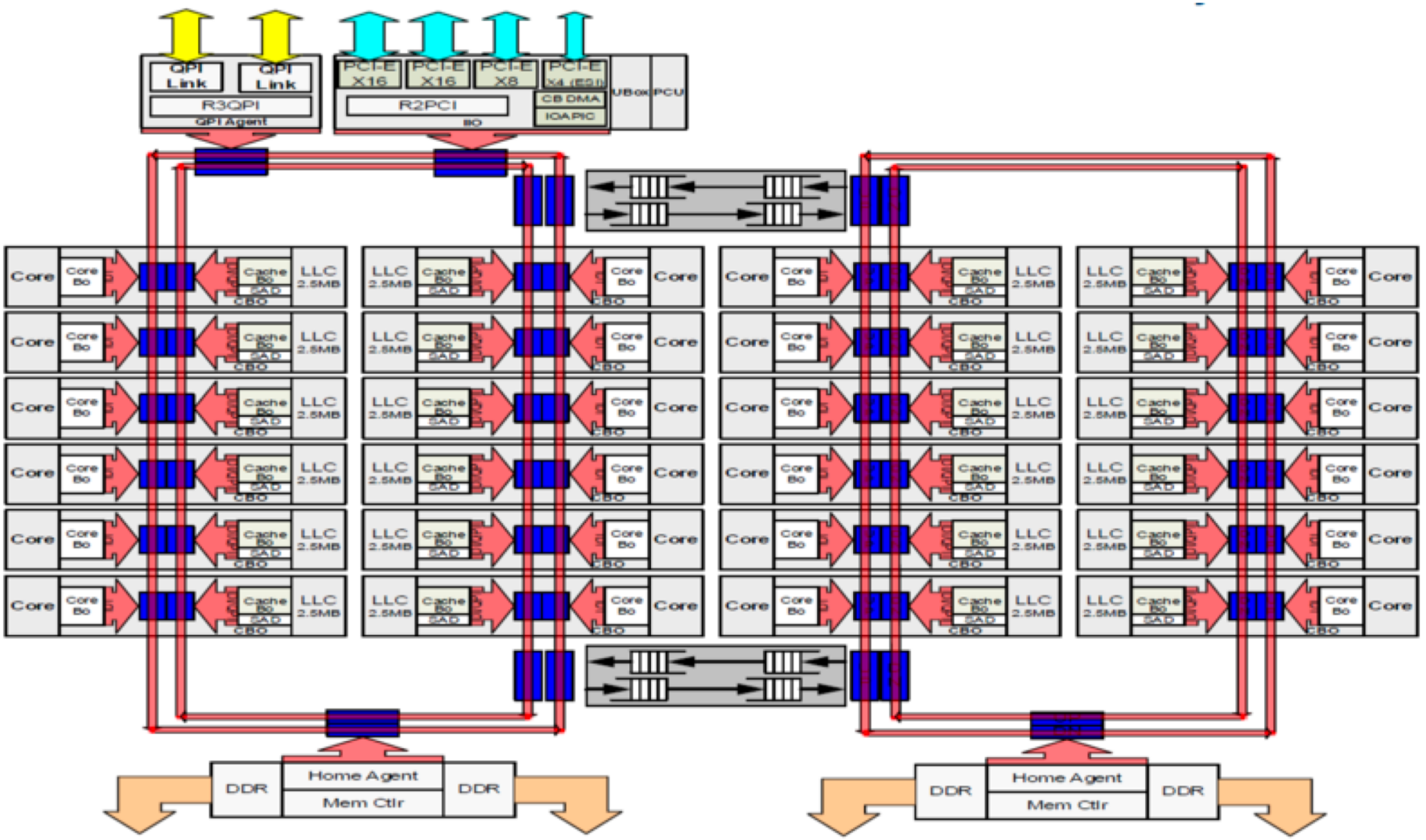
コンパイルの方法	「a=a+a」を 10 回	「a=a*a」を 10 回
gcc -00	129	130
gcc -01 (gcc -0 と同じ)	50	51
gcc -02 / gcc -03	40	40
icc -00	130	145
icc -01	130	129
icc -02 (icc -0 と同じ) / icc -03	40	40

http://www.cc.u-tokyo.ac.jp/support/press/news/VOL10/No5/200809tuning.pdf

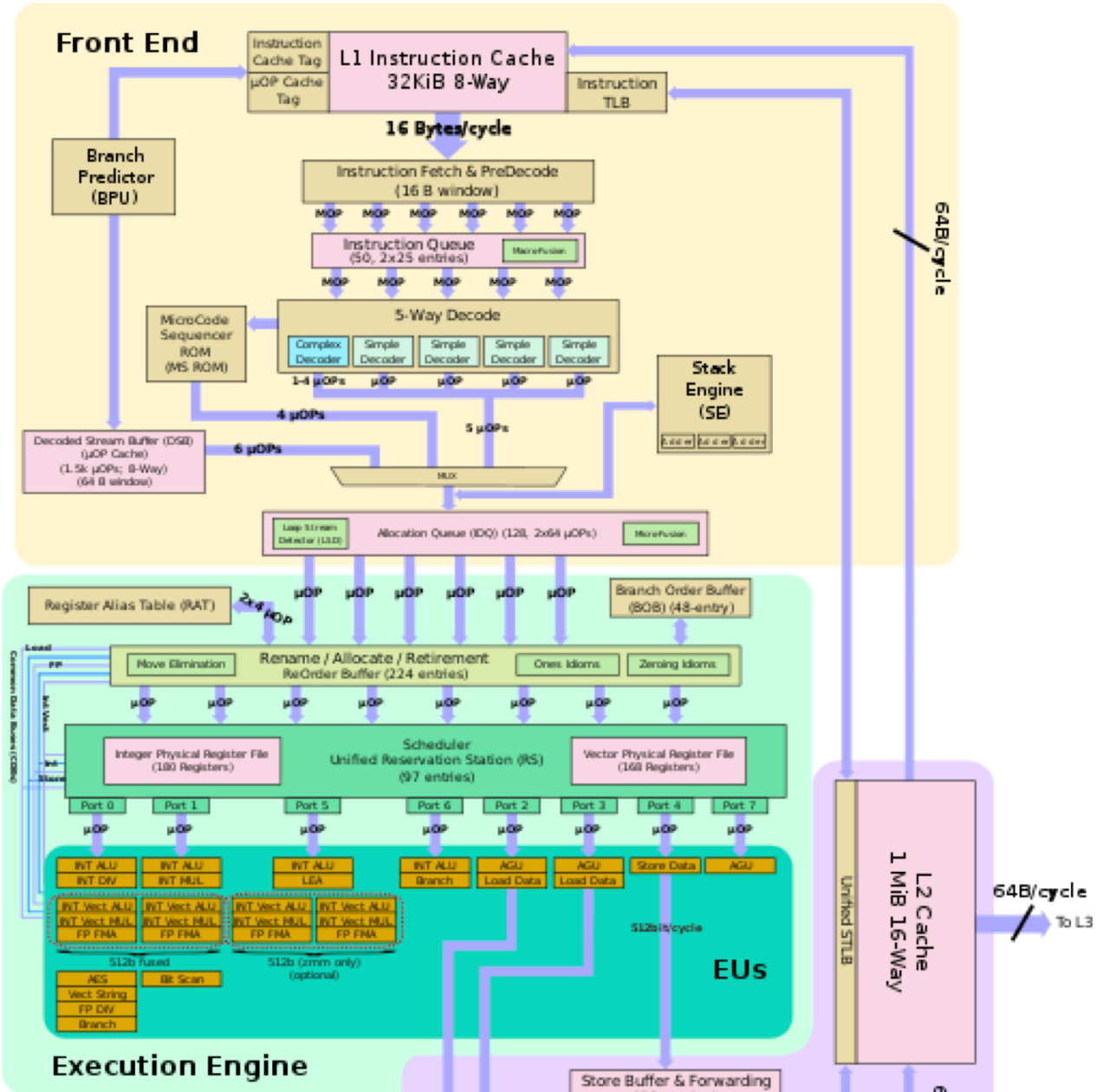
Intel Broadwell architecture



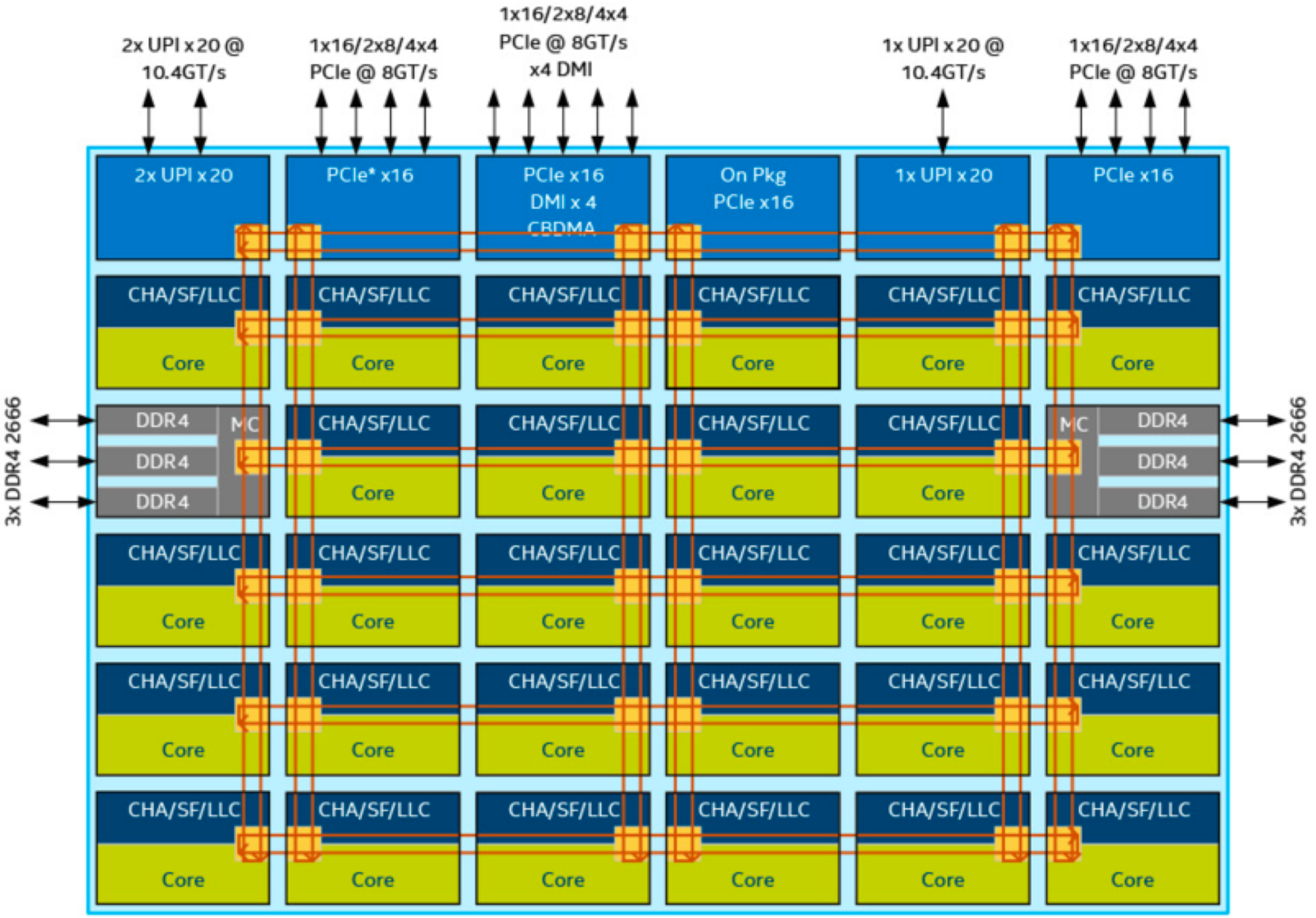
Intel Broadwell NoC



Intel Skylake Architecture

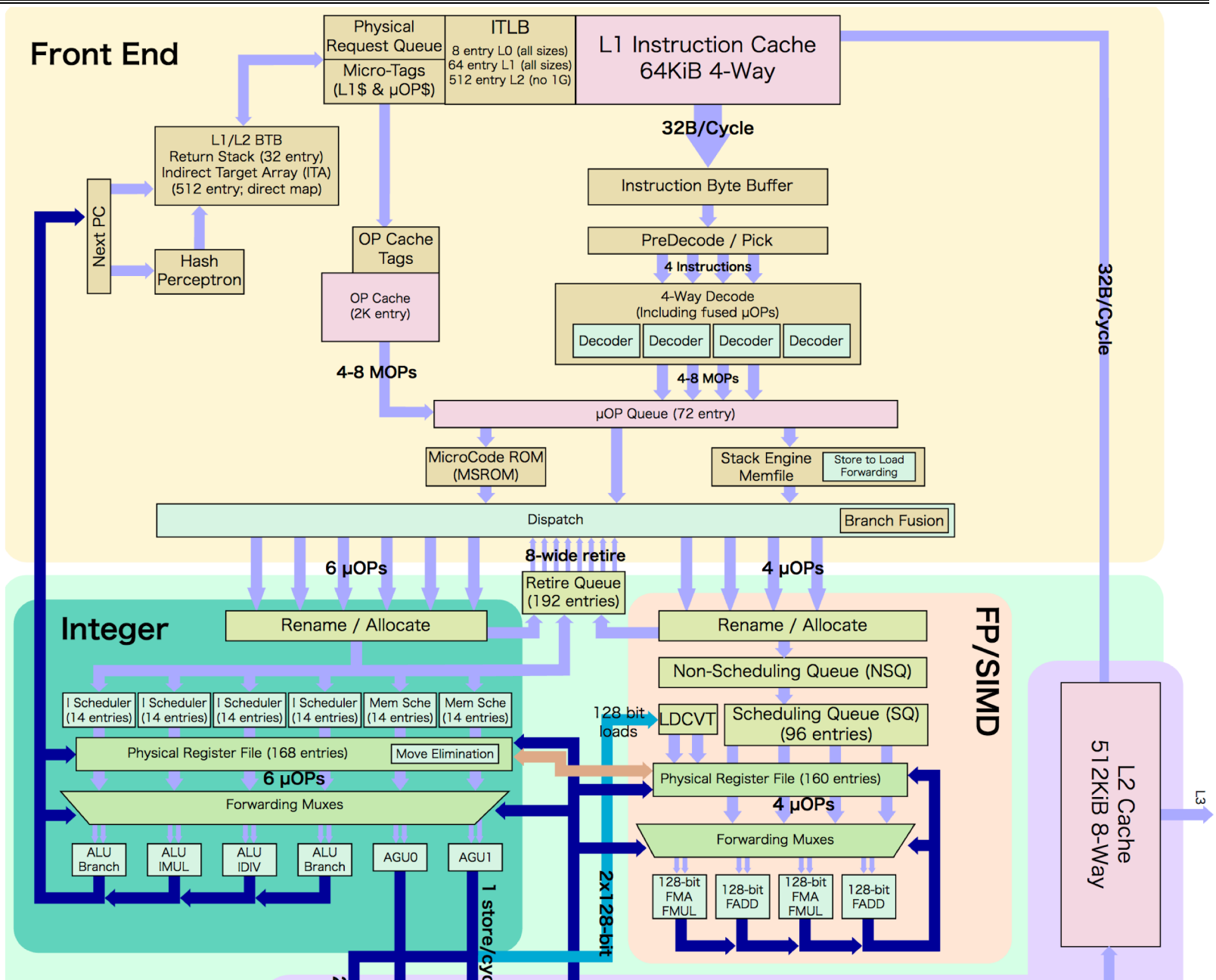


Intel Skylake NoC

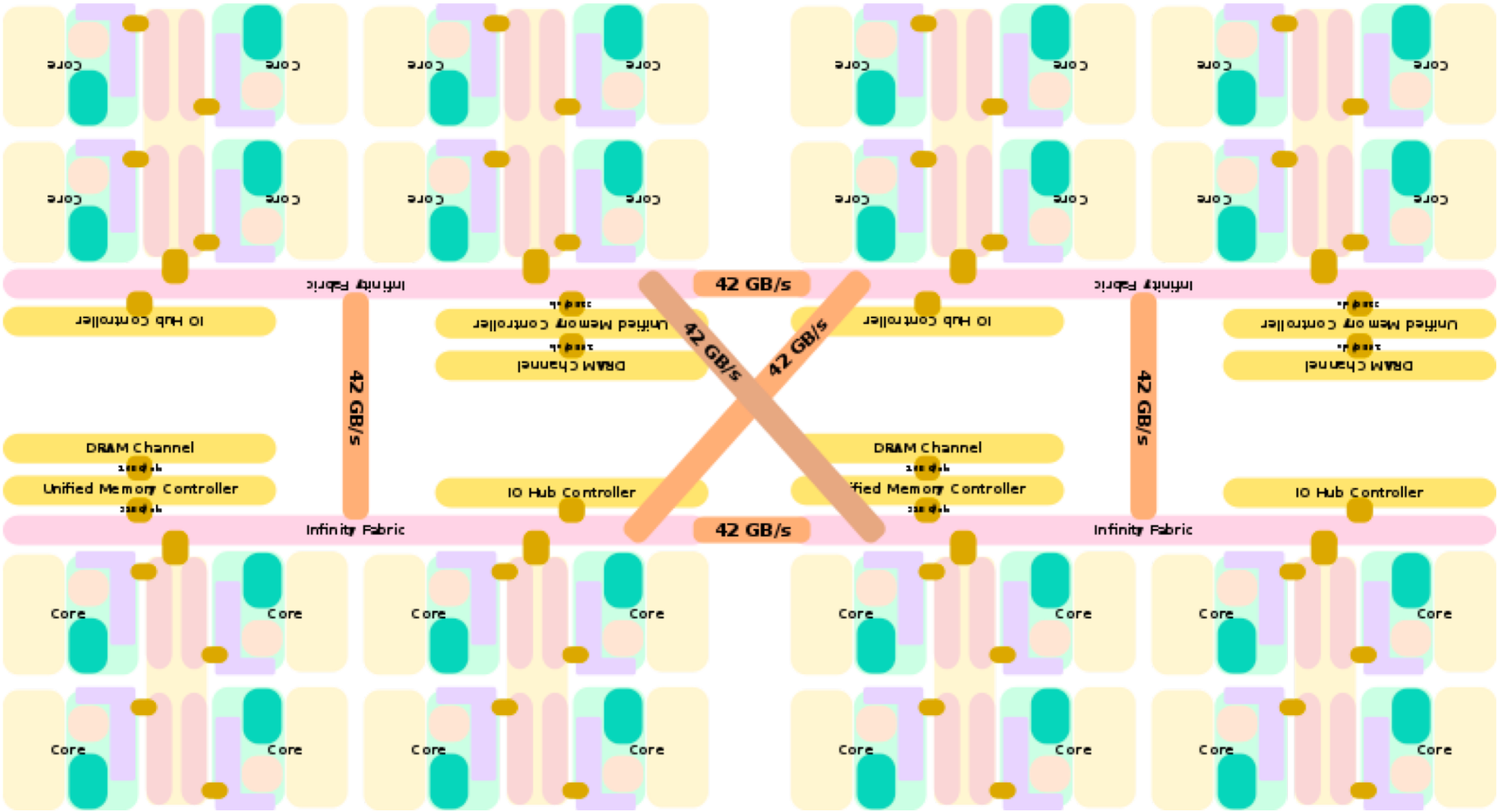


Mesh interconnect

AMD Zen architecture



AMD Zen NoC



8 dies are interconnected; 4 cores x 8 dies = 32 cores

Optimization using SIMD instructions

- Only blocking is not enough to utilize CPUs as you did in the exercise problem
- Need to use SIMD instructions (in this case AVX2)