Processors and Memory

System Processors

CPU Avrchitecture

Memory Systems

Memory Addressing

Memory Management in Software

System processors

- Two main families of CPU for desktop computers
  - Intel (Core line)
  - AMD (Athlon)

- Both families use x86 instruction set
  - CPU hardware very different between Intel & AMD
  - software compatible

- Intel
  - Core i3 (low end), i5 (mid range), i7 (high end)
- AMD
  - first to bring 64 bit CPU to desktop
  - first to attach memory directly to CPU
  - first to offer desktop multicore CPUs

Sample x86 assembler

```
.model small
.stack 100h

.data
 msg db 'Hello world!$’

.code
start:
      mov ah, 09h ; Display the message
      lea dx, msg
      int 21h
      mov ax, 4C00h ; Terminate the executable
      int 21h
end start
```

- From Wikipedia

Assembly Language

Machine code is the language the CPU uses.

- It can be written in a more human-readable format called ***assembly language***
- ***assembly language*** Such languages are unique to every different CPU make or model.
- Code written in this language can be ***assembled*** into machine code.
- The instruction stream at bottom is actually what is executed and is saved in an ***object file***

Syntax is:
Action, Dest, Src

Machine code bytes	Assembly language statements
B8 22 11 00 FF	foo:
01 CA	movl $0xFF001122, %eax
31 F6	addl %ecx, %edx
53	xorl %esi, %esi
8B 5C 24 04	pushl %ebx
8D 34 48	movl 4(%esp), %ebx
39 C3	leal (%eax,%ecx,2), %esi
72 EB	cmpl %eax, %ebx
C3	jnae foo
	retl

Intel Core (64bit)

Instruction stream

```
B8 22 11 00 FF 01 CA 31 F6 53 8B 5C 24
04 8D 34 48 39 C3 72 EB C3
```

Assembly Language

Machine code is the language the CPU uses.

- It can be written in a more human-readable format called ***assembly language***
- ***`assembly language`***
- The same code stream, but now interpreted by an old 32-bit intel chip.

Machine code bytes

```
0000 B82211
0003 00FF
0005 01CA
0007 31F6
0009 53
000A 8B5C24
000D 018D3448
0011 39C3
0013 72EB
0015 C3
```

Assembly language statements

```
MOV     AX,1122
ADD     BH,BH
ADD     DX,CX
XOR     SI,SI
PUSH    BX
MOV     BX,[SI+24]
ADD     [DI+4834],CX
CMP     BX,AX
JB      0000
RET
```

Intel 8086 (32bit)

- Note how the text is different to that of the previous slide !
- Note how the binary is not (!!!)

Instruction stream

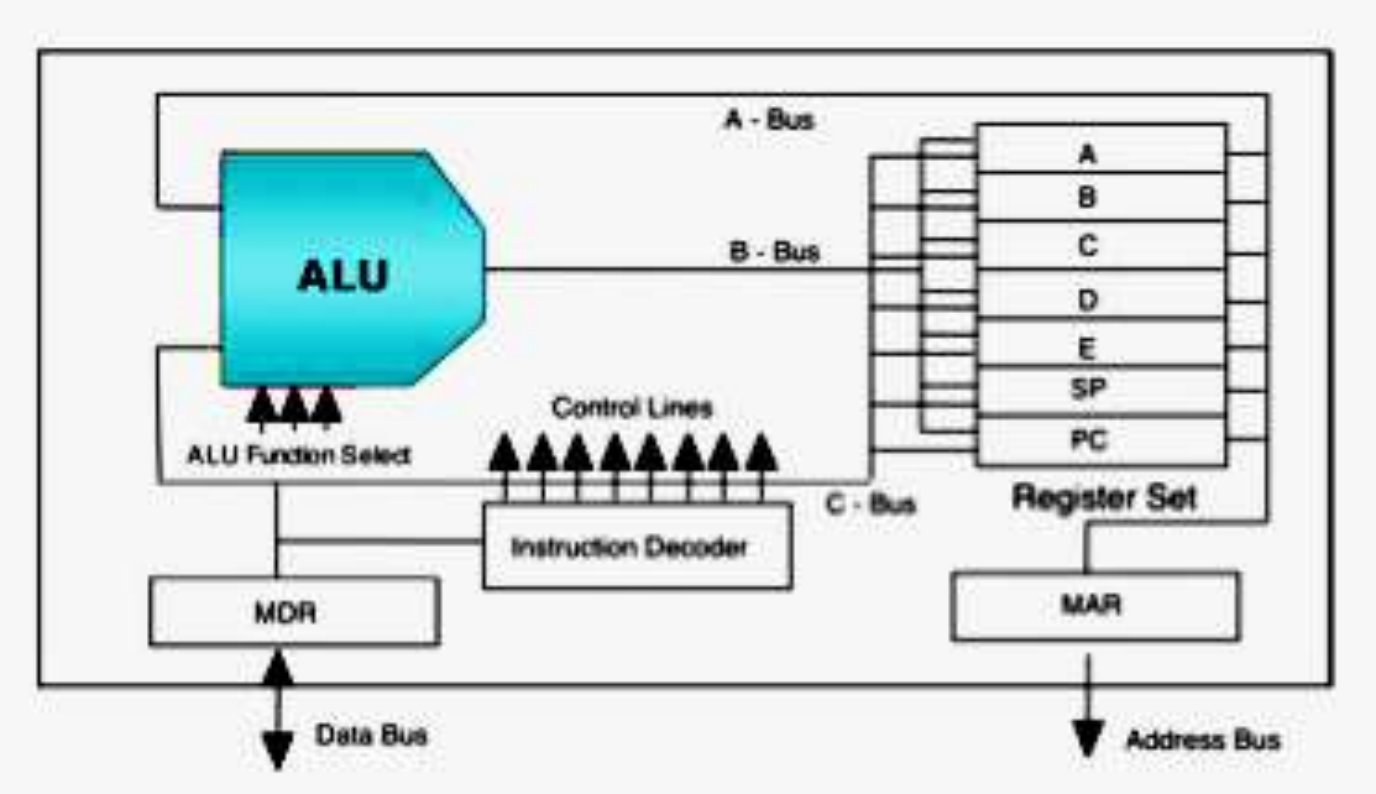
```
B8 22 11 00 FF 01 CA 31 F6 53 8B 5C 24
04 8D 34 48 39 C3 72 EB C3
```

- In fact, backward compatibility of its binary op-code was one of Intel's biggest features.

CPU Processor sockets

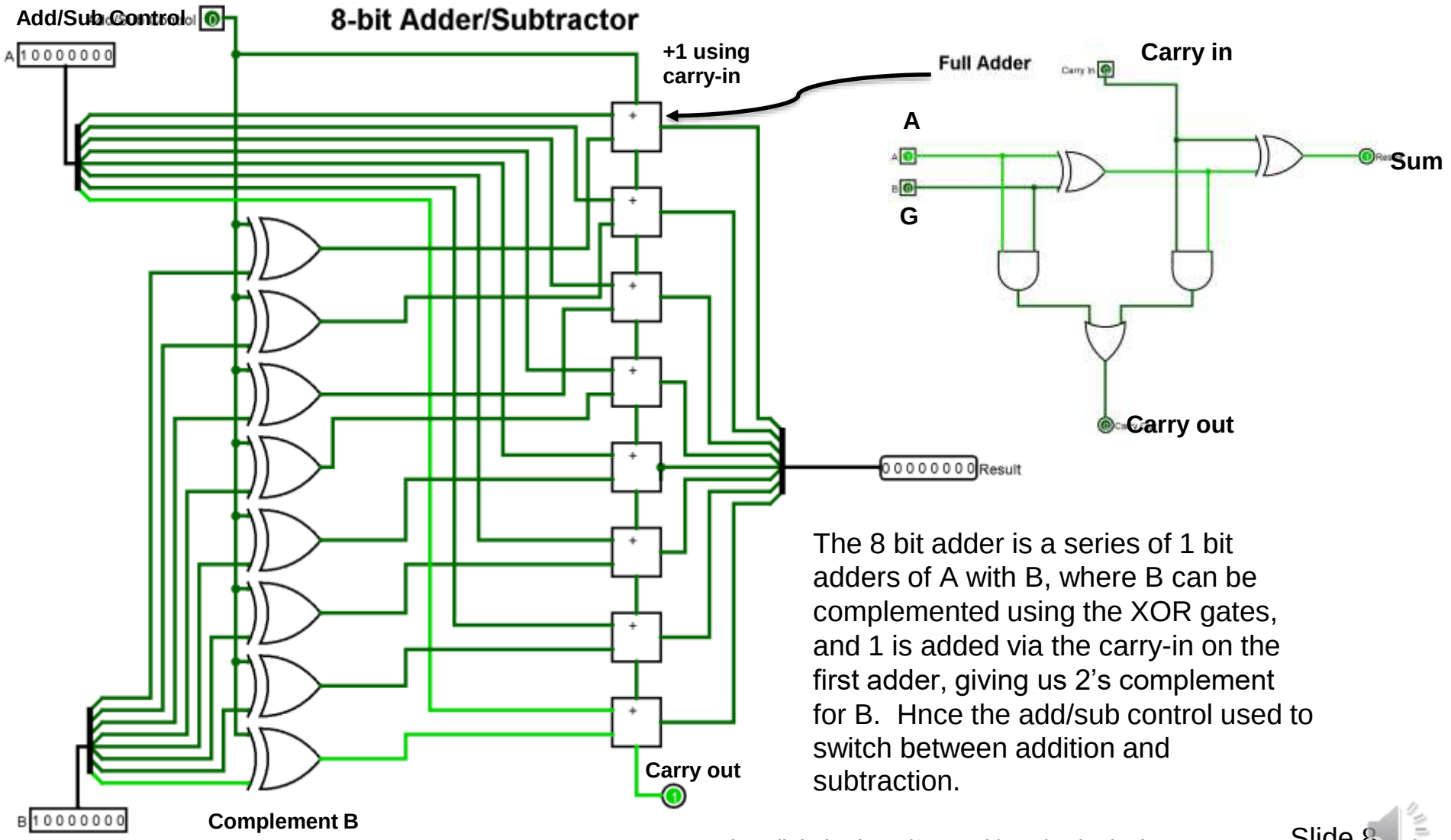
- CPUs sit in a motherboard socket
  - motherboard socket has pins, CPU connects with the pins
  - since Core and Athlon CPUs have different designs, they have different socket designs
  - newer CPUs might not be supported by older motherboards

CPU Architexture



- ***Arithmetic Logic Unit (ALU)*** – performs arithmetic and logic functions
  - ***B-Bus = A-Bus operation C***-bus
- ***Register stack*** – high speed‘scratch pad’to store data currently being processed
- ***Memory Data Register (MDR)*** – stores data just received from, or to be written to memory
- ***Memory Address Register (MAR)*** – stores address of memory to be accessed next
- ***Program Counter (PC)*** – stores address of next instruction
- ***Stack Pointyer (SP)*** – stores call return address and temporary local values

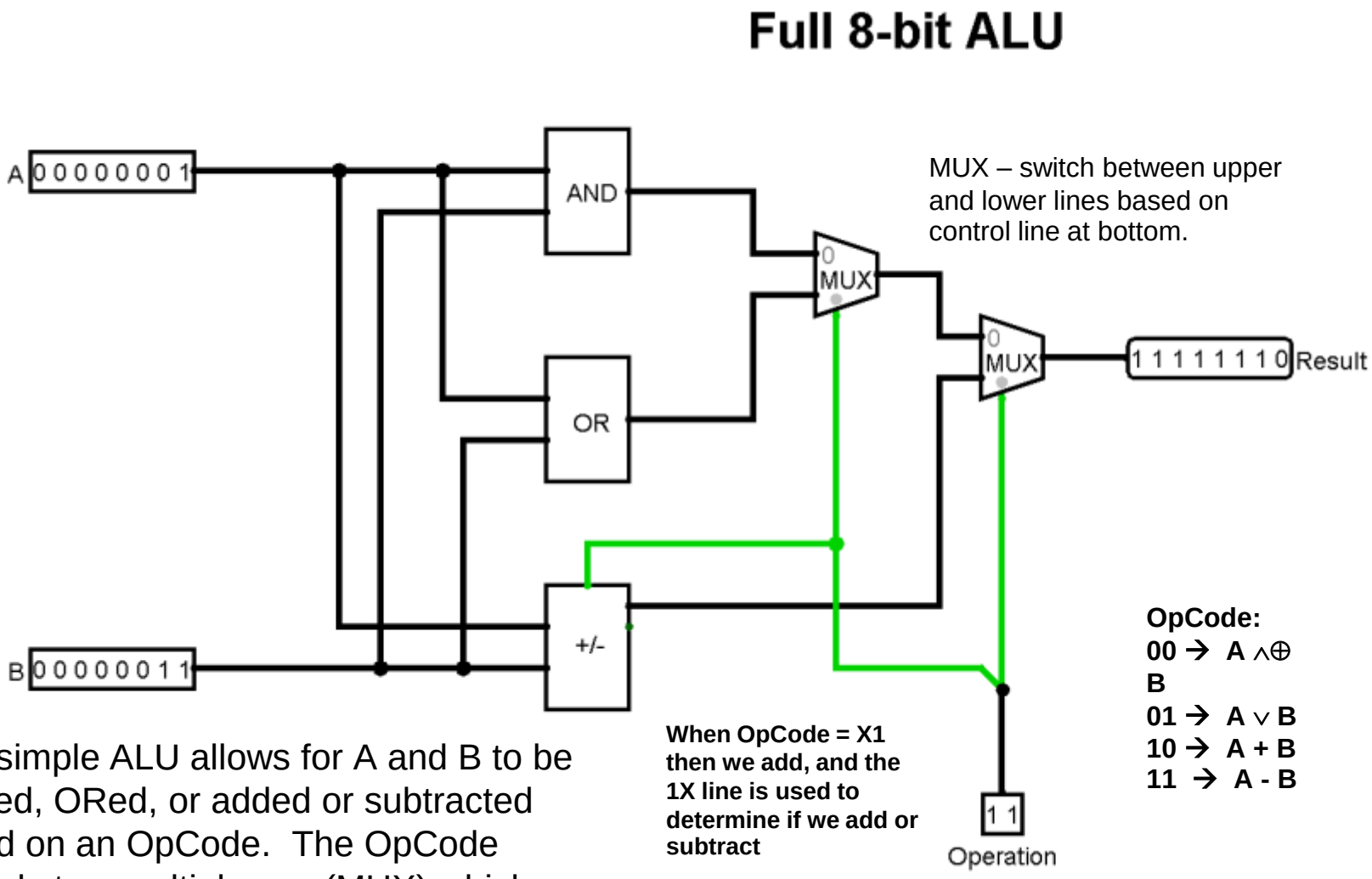
Add / SUB circuit



The 8 bit adder is a series of 1 bit adders of A with B, where B can be complemented using the XOR gates, and 1 is added via the carry-in on the first adder, giving us 2's complement for B. Hnce the add/sub control used to switch between addition and subtraction.

More: https://minnie.tuhs.org/CompArch/Tutes/week03.html

Simple ALU (using Logisym)



This simple ALU allows for A and B to be ANDed, ORed, or added or subtracted based on an OpCode. The OpCode controls two multiplexers (MUX) which determine which circuit's output is actually wired to the ALU output.

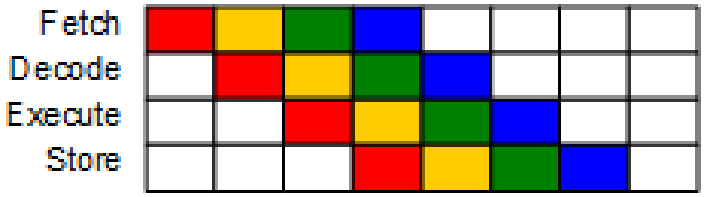
CPU architecture

- Instructions per clock (IPC)
  - amount of work performed per clock tick
  - some designs have fast clock ticks but lower IPC
  - faster clock speeds consume more energy

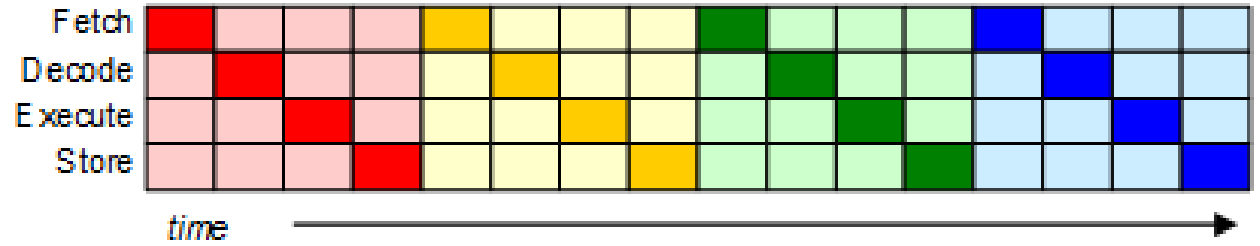
- Instruction pipelining (superscalar architectures)
  - analogous to an assembly line, where each worker on the line does a small amount of work building an item (say, a car), which moves down the line from one worker to another
  - basic instruction cycle (fetch, decode, execute, store) can be pipelined (e.g. while one instruction is being executed, another is being decoded)
  - steps in the instruction cycle can be broken down into micro-ops that can be pipelined
  - length of pipeline is called its *depth*

CPU architecture

pipelined



not pipelined



- Pipelining enhances throughput
  - individual instruction doesn't run any faster

Fetch Execute Cycle

- Fetch
  - copy pc to MAR
  - Get instruction into MDR
- Decode
  - Identify operation and operands
- Execute
  - Carry out instruction
  - May require additional memory fetches
- Store
  - Put data to be stored in MDR
  - Put location into MAR

CPU architecture

- Pipelines may *stall*
  - e.g. an instruction need to wait for memory
  - causes a ‘bubble’ in the pipeline where no work is being done

- Branch prediction
  - CPU estimates what future instructions will be, and performs them inside a pipeline stall or other ‘spare time’
  - quite good predictions possible, but not perfect

- Out of order execution
  - instructions re-arranged in pipeline to reduce overall processing time and minimise stalls

CPU Multiprocessing

- Symmetric multiprocessing
  - CPU has more than one (identical) core (i5 core shown below)
    - e.g. Intel Core i7 series has 2, 4, 6, 8 (8 is most common) or 6 cores
  - cores work in parallel
  - thus more than one instruction executed at once

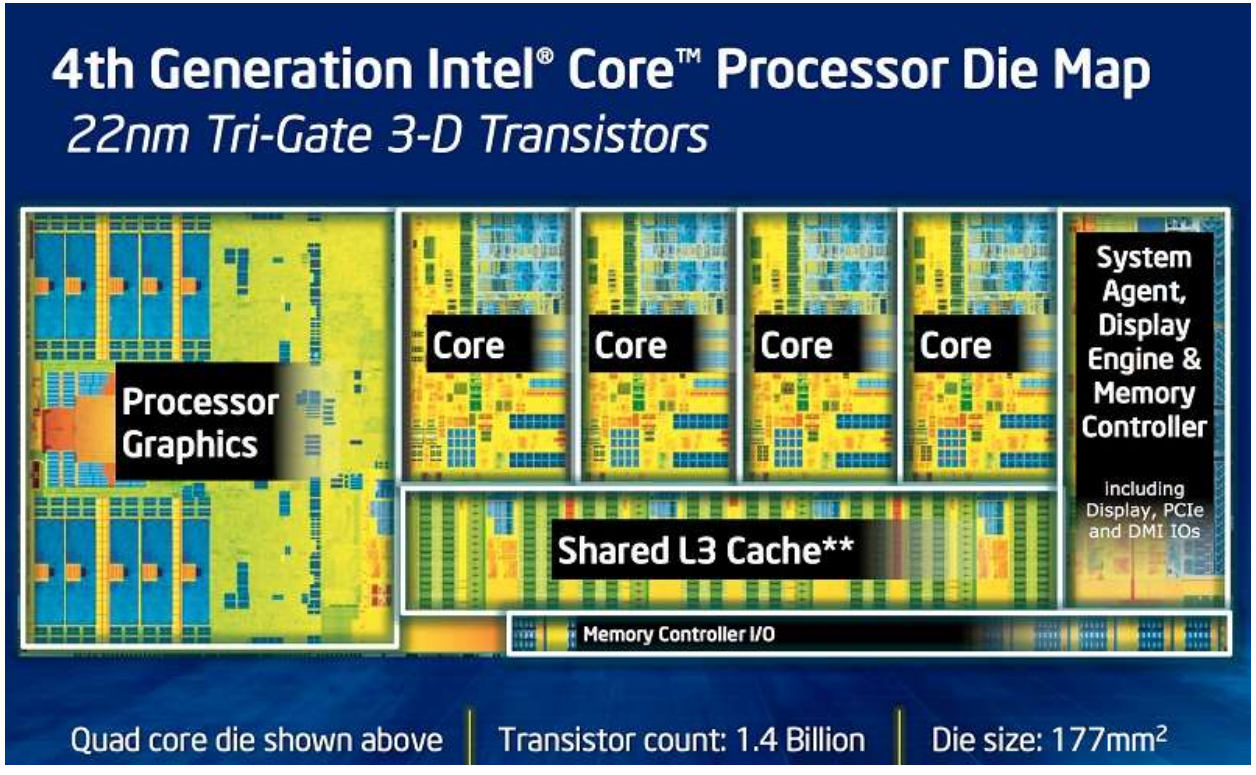


diagram: Intel Corporat

RISC and CISC Architecture

- CPU architecture can be broadly classified into two different types:
- CISC – Complex instruction Set
- RISC – Reduced instruction Set

- Another distinction that can be made is the Von Neumann model memory model vs the Harvard memory model – we will discuss this later

CISC – Complex Instruction Set

- When modern CPU's were first developed, memory was very expensive
- Programs could be smaller if instructions were more powerful and could perform multiple operations

- Example: Multiply two numbers together store the result.
- HLL: c = a * b;
- CISC MULT A,B,C

CISC

- Multiple operations lead to many different kinds of instructions that access memory
- This results in length variable and fetch-decode-execute time unpredictable – making it more complex
- Thus hardware handles the complexity
- X86 hardware is like this

RISC

- Attempt to make architecture simpler
- Reduced number of instructions in the instruction set
- Need more instructions to do the same thing compared to CISC
- Try to educe the number of memory accesses required by increasing the number of registers

- The multiply example would become:

LOAD 1,A

LOAD 2,B

MULT 1,2

STORE 1,C

RISC

- A RISC processor will generally have:

- A large number of general purpose registers
- A small number of instructions with the same format
- Optimization of instruction pipeline

- Uniform instruction size and format should make pipelining easier

System processors

- Advanced RISC Machines
  - Some designers of RISC architecture license their designs to hardware manufactures, and have lots of different options
  - used in smart phones, tablets, embedded systems
  - not x86 compatible
  - some chip designs have heterogeneous cores
    - low-power low-performance cores teamed with more powerful but more power-hungry cores.
    - cores activated or deactivated according to processor workload
    - very efficient use of electricity

ARM Architecture

Architecture	Core bit-width	Cores	Profile	References
ARM Holdings	Third-party			
ARMv1	32[a 1]	ARM1		Classic
ARMv2	32[a 1]	ARM2, ARM250, ARM3	Amber, STORM Open Soft Core[39]	Classic
ARMv3	32[a 2]	ARM6, ARM7		Classic
ARMv4	32[a 2]	ARM8	StrongARM, FA526, ZAP Open Source Processor Core[40]	Classic
ARMv4T	32[a 2]	ARM7TDMI, ARM9TDMI, SecurCore S C100		Classic
ARMv5TE	32	ARM7EJ, ARM9E, ARM10E	XScale, FA626TE, Feroceon, PJ1/Mohawk	Classic
ARMv6	32	ARM11		Classic
ARMv6-M	32	ARM Cortex-M0, ARM Cortex-M0+, ARM Cortex-M1, SecurCore SC000		Microcontroller
ARMv7-M	32	ARM Cortex-M3, SecurCore SC300		Microcontroller
ARMv7E-M	32	ARM Cortex-M4, ARM Cortex-M7		Microcontroller
ARMv8-M	32	ARM Cortex-M23,[41] ARM Cortex-M33[42]		Microcontroller [43]
ARMv7-R	32	ARM Cortex-R4, ARM Cortex-R5, ARM Cortex-R7, ARM Cortex-R8		Real-time
ARMv8-R	32	ARM Cortex-R52		Real-time [44][45][46]
ARMv7-A	32	ARM Cortex-A5, ARM Cortex-A7, ARM Cortex-A8, ARM Cortex-A9, ARM Cortex-A12, ARM Cortex-A15, ARM Cortex-A17	Qualcomm Krait, Scorpion, PJ4/Sheeva, Apple Swift	Application
ARMv8-A	32	ARM Cortex-A32		Application
ARMv8-A	64/32	ARM Cortex-A35,[47] ARM Cortex-A53, ARM Cortex-A57,[48] ARM Cortex-A72,[49] ARM Cortex-A73[50]	X-Gene, Nvidia Project Denver, Cavium Thunder X,[51][52][53] AMD K12, Apple Cyclone/Typhoon/Twister/Hurricane/Zephyr, Qualcomm Kryo, Samsung M1 and M2 ("Mongoose")[54]	Application [55][56]
ARMv8.1-A	64/32	TBA	ThunderX2[57]	Application
ARMv8.2-A	64/32	ARM Cortex-A55,[58] ARM Cortex-A75,[59] ARM Cortex-A76[60]		Application [61]
ARMv8.3-A	64/32	TBA	Apple A12 Bionic	Application
ARMv8.4-A	64/32	TBA		Application

RISC

	CISC processor	RISC processor
	Intel 80486	Sun SPARC
Year developed	1989	1987
Num. instructions	235	69
Instruction Size (bytes)	1-11	4
Addressing modes	11	1
GP Registers	8	40-520

Which is faster?

$$\frac{\text{time}}{\text{program}} = \frac{\text{time}}{\text{cycle}} \times \frac{\text{cycles}}{\text{instruction}} \times \frac{\text{instructions}}{\text{program}}$$

Program time = (cycle time / inst per cycle) *
 # of instructions

Answer: It depends

CISC vs RISC

CISC	RISC
Emphasis on hardware	Emphasis on software
Multiple instruction sizes and formats	Instructions of same set with few formats
Less registers	Uses more registers
More addressing modes	Fewer addressing modes
Extensive use of microprogramming	Complexity in compiler
Instructions take a varying amount of cycle time	Instructions take one cycle time
Pipelining is difficult	Pipelining is easy

Risc – each instruction takes the same time,
Simpler architechure -> smaller, faster, more cache

CPU architecture

- Computation speed-up of a task not linear
  - many problems are not 100% parallelisable
  - overhead involved in coordinating sharing of work between cores
  - access to memory must be coordinated
  - keeping each cores cache consistent is difficult

- Often each core is performing a *different* task

Hyper Threading

Best described using a Kitchen metaphor

- Multi-processing
  - 3 Kitchens, 1 chef in each
  - Can work independently
  - Only share a delivery bench for waiter delivery to customers
  - Comparatively little congestion
- Threads
  - 5 Chefs sharing one kitchen
  - Can make much better use of kitchen facilities
  - Unless organised, can easily get in each other's way
  - Sacrifice indepdence for utility

Hyper Threading

- Simultaneous multithreading (SMT)
  - pipeline stalls are inevitable – so use that wasted CPU time
  - use stalls in running one thread of execution to run another thread of execution
  - operating system thus sees a single core as two logical cores, and schedules different tasks to each
  - example: Intel's Hyper-threading

Hyper Threading

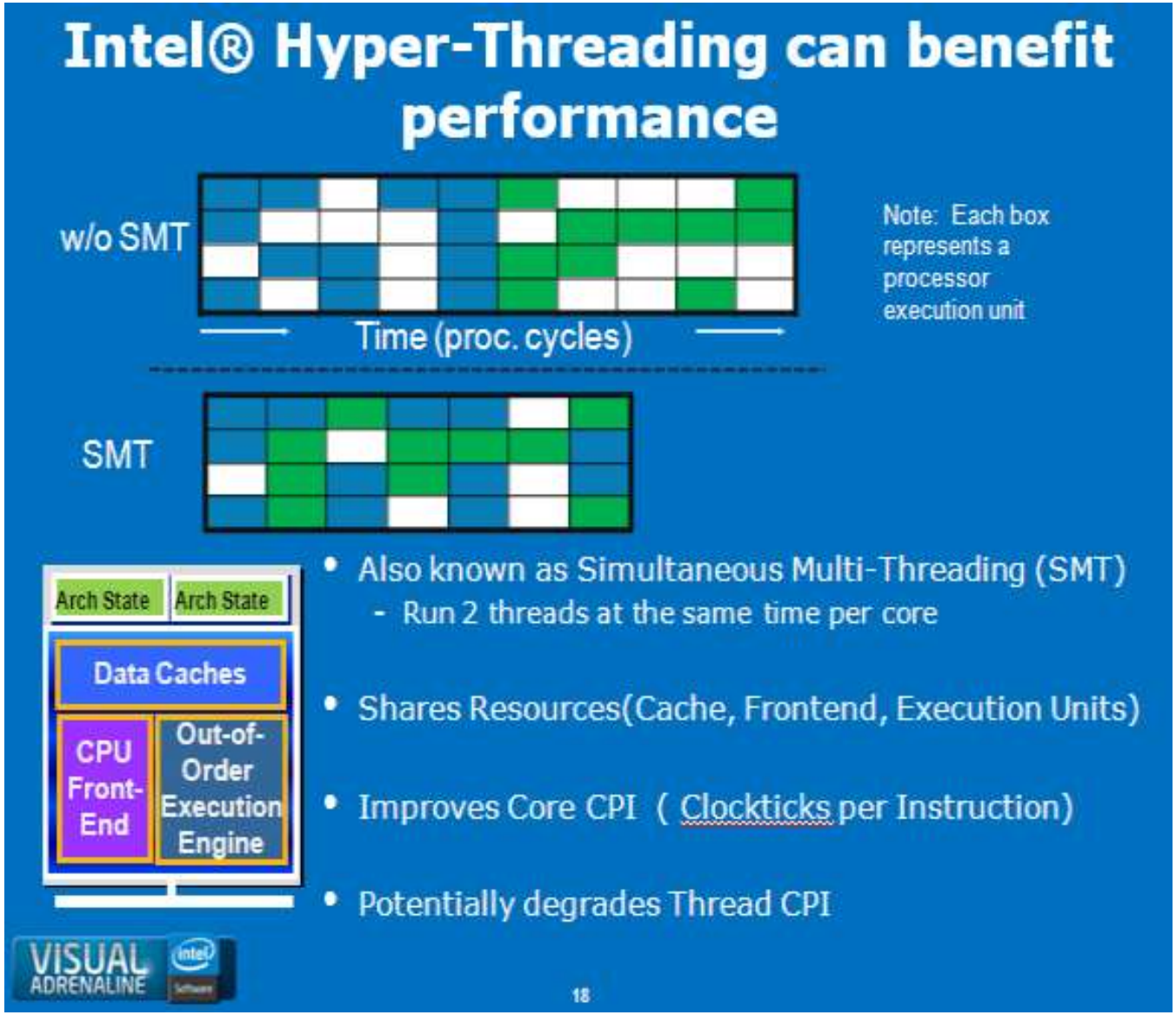


diagram: Intel Corporation

Other Processing Models

- In the “standard” CPU model, the CPU executes one instruction at a time, operating on one piece of data

- Eg

```
Load 1,A
Load  2,B
Add    2,1
Store  2,B
```

- This is known as a Single Instruction, Single Data or SISD computer

Parallel Processing

- Some applications, especially graphics related, perform the same calculations many times, but on different data

- This type of processing can be made much faster if we are able to perform the **same** calculation on many **different** pieces of data at once

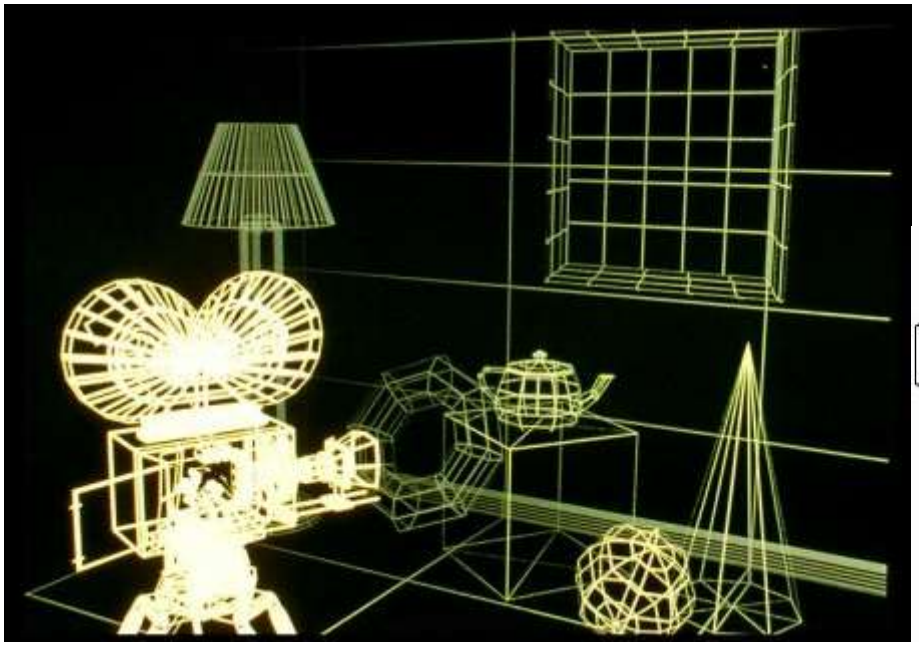
- To see how this can work we first need a basic understanding of the main steps involved in graphics processing…

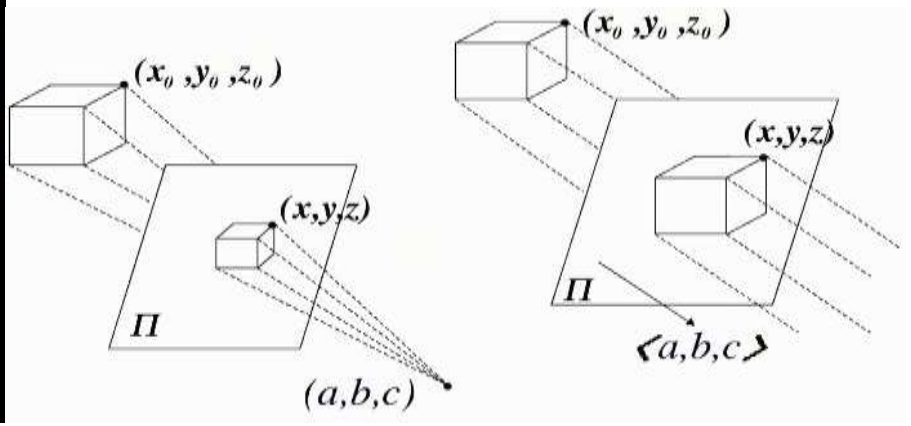
Graphics Processing

- Ultimate goal: display a series of images on a screen (2D)
- Images consist of pixels, which have colour and brightness
- However the internal model of an image is a collection of 3D objects and a lighting source

Graphics Processing

- Shapes are represented as coordinates in 3D space)





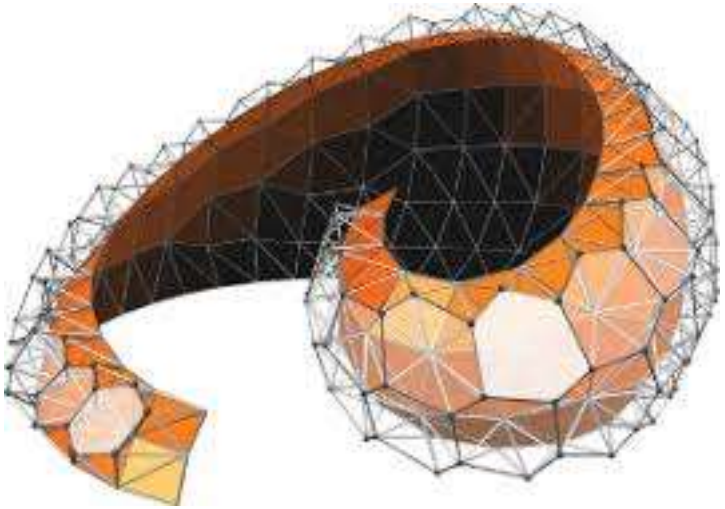
Graphics

- We also have some shading, or texture information to add in

Graphics

- Surfaces are normally broken up into triangles, and these are very easy to manipulate mathematically.

- Curved surfaces are approximated by using a very large number of triangles



Graphics

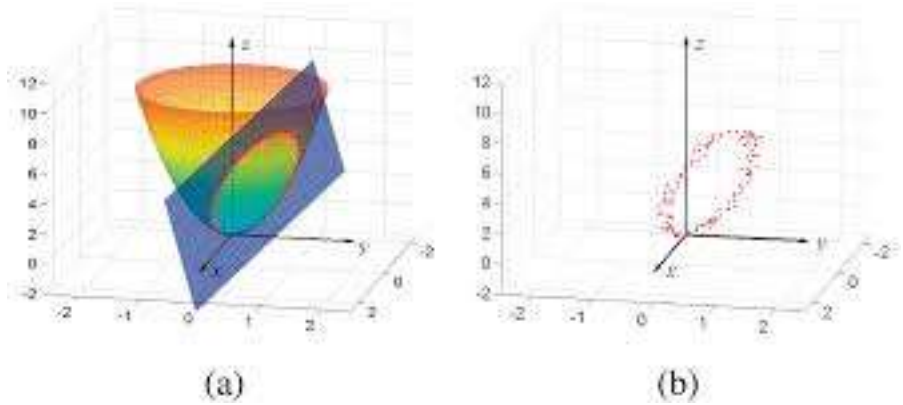
- As the “camera” or view changes the objects need to be rotated around an arbitrary point

- The change in position can be calculated by applying a rotation matrix to all of the vertices (ie same instructions applied to a large amount of data)

Perspective

- Once rotation is complete, we need to determine which objects, or parts of objects are visible from a given position (Camera Position)

- This requires calculating points of intersection



(a) (b)

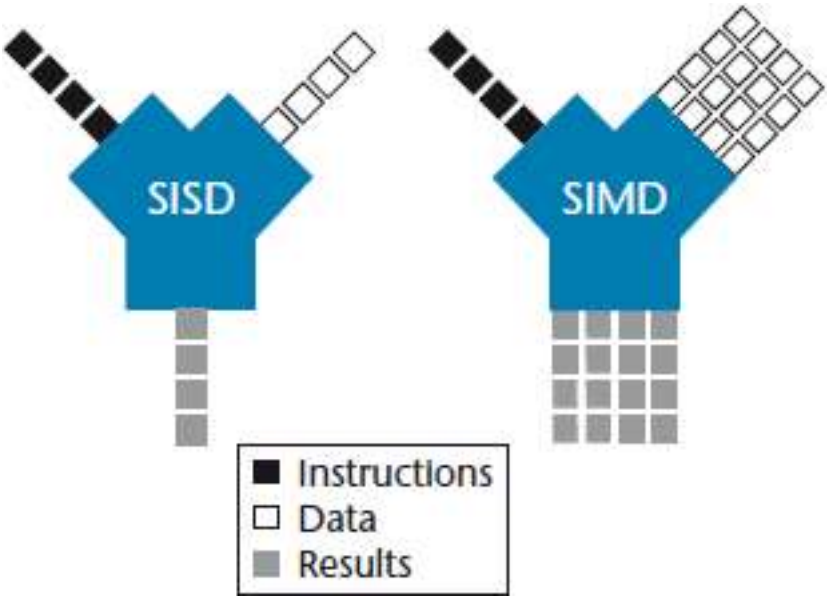
- Again same calculations applied to lots of different data

- All these steps involve lots of calculations on three dimensional coordinates, and hence a lot of matrix calculations

- However in many cases the same calculation is repeated performed on different data

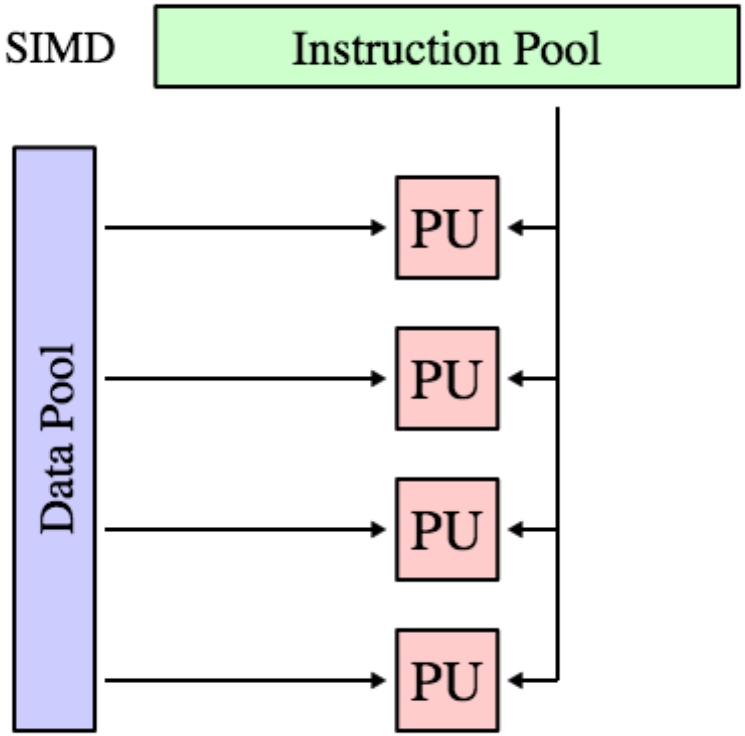
SIMD

- In a Single Instruction Multiple Data Computer model, one instruction can operate on multiple data sets



SIMD

- This is implemented as multiple processors that each follow the same instructions, but have some local memory where they can store the data it is working on



GPU vs CPU

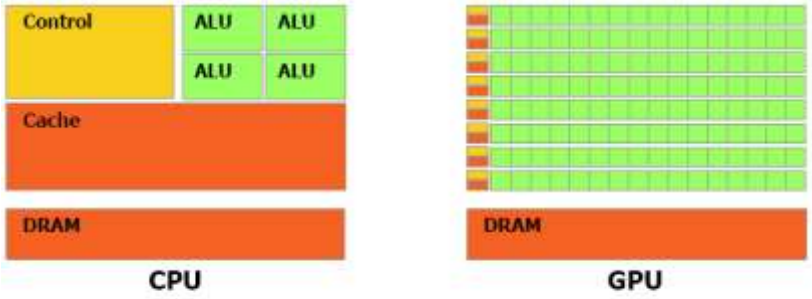


Figure 1-2. The GPU Devotes More Transistors to Data Processing

A typical GPU has many more processors compared to a CPU

Nvida Maxwell Architechure

MAXWELL "GM204"

Top Level

- 5.2 Billion Transistors
- 2x performance vs GK104
- 16 SMM
- 2048 CUDA Cores
- 16 Geometry Units
- 128 Texture Units
- 64 ROP Units
- 256-bit GDDR5

SIMD

- SIMD processors are partiincularly useful in applications where the same operation needs to be applied to large amounts of data such as
- Digital Signal Processing (DSP)
- Computer Graphics

Summatu

- System Processors
  - Assembler / machine language
  - CPU Architecture
    - ALU
    - Pipelining
    - Multiprocessing
    - RISC / CISC
    - SIMD
      - Hyper Threading
      - Graphics