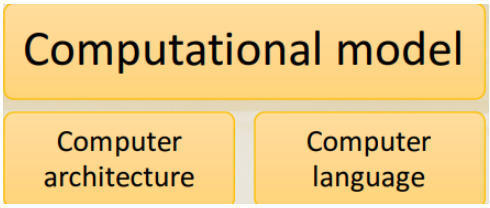
Contents

SzA1. Computational Model, Computer Architecture

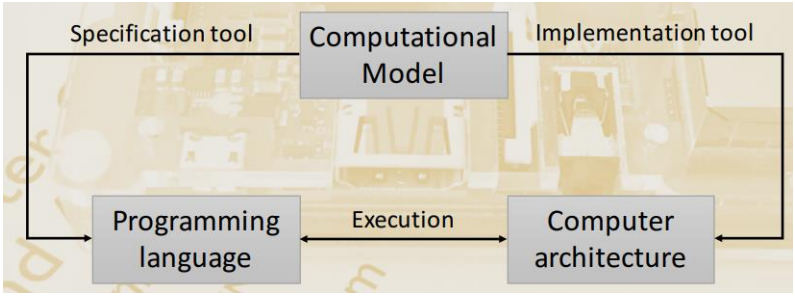
(definition, connection between C.M. and C.A., most important C.Ms, the Von-Neumann C.M., the Dataflow C.M., definition of the Computer Architecture, Logical and Physical Architecture of processors)

Computational Model



- Fundamental term of computing
- Provides a **higher level of abstraction** than Computer architecture and Computer language, but covers both
- Comprises of:
- **Basic item of computation**
  - The items we do computation on, and the kind of computations we perform on them
- **Problem description model**
  - The style and method of problem description
  - **Procedural style**: algorithm for solving the problem is stated
  - **Declarative style**: all facts and relationships relevant to the problem are stated
- **Execution model**
  - How to perform the computation (strongly related to problem description)
  - Execution semantics (how to perform one step [Von Neumann: state transition, Dataflow: dataflow)
  - Control of the execution sequence
    - Control-driven: execution based on the sequence of instructions; order can be modified by control instructions (e. g. jump)
    - Data-driven: operation active as soon as all needed input is available (eager)
    - Demand-driven: operation activate only when its execution is needed for final result (lazy)

Relationship between Computational model and architecture (Von Neumann example)



- Programming language
  - allows variables to be declared and their values to be manipulated
  - provides control instructions to allow explicit control of execution sequence
- Von Neumann architecture
  - Variables are implemented as addressable locations in memory or registers, they can be altered
  - Sequence control implemented by Program Counter

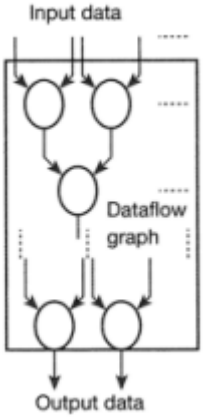
Most important computational models

- Turing model: procedural, data based
- Von Neumann model: procedural, data based
- Dataflow model: procedural, data based
- Object-based model: procedural, object based
- Applicative model: declarative, argument & function
- Predicate-logic based model: declarative, sets and predicates

Von Neumann Computational Model

- **Basic item of computation are data**, operations performed on data
- Data items identified by names -> **variables** in programming languages
- Memory or register **addresses are mapped to variable names**
- **Multiple data assignment**: variables can be assigned new values as many times as needed
- Problem description model is **procedural**, task is specified as a sequence of instructions
- **State transition semantics**, model behaves like a finite state machine
    - State is defined by the actual values of variables
    - State transition: from present state to the next one
- Languages should
    - **allow declaration of variables** and altering their values
    - **provide control statements** to control execution sequence

Dataflow Computational Model



- **Basic item of computation are data**
- **Procedural** style
- **Values of variables cannot be altered**, one time value setting
- Input data **flows through the graph**, output is produced
- **Parallel execution** is possible
- **Data-driven**

Computer Architecture

- How a set of software and hardware interacts to form a computer system
- How a computer system is designed and what technologies it is compatible with
- Attributes of the system visible to a programmer
    - Direct impact on the logical execution of a program

Logical Architecture

- Abstraction of the physical design
- Logical components used (registers, execution units, etc.) and their interconnections
- Sequence of information transfers
- **Processors**: data, data manipulation tree, state space, space operations

Physical Architecture

- Concrete circuit elements used, and their interconnections
- Initiated signal sequences
- **Processors:** execution unit, control unit, I/O system, interrupt system

SzA2. Data files, Data manipulations

(Definition, memory, logical and physical memory, evolution of register files, data manipulation tree, data types, addressing modes, indexed addressing)

Memory: stores data directly accessible by the CPU

Memory	Register
Greater	Smaller
Slower	Faster
Cheaper	More expensive
External chip(RAM)	internal chip in CPU
address space can be shared with I/O devices address space	Separate address space

Physical memory:

- What the CPU addresses on its address bus
- Sequence of 8-bit bytes, each with a physical address

Logical memory:

- Every application cannot manage its memory at a physical level
- CPUs introduce a level of abstraction, **Memory Management** -> logical memory

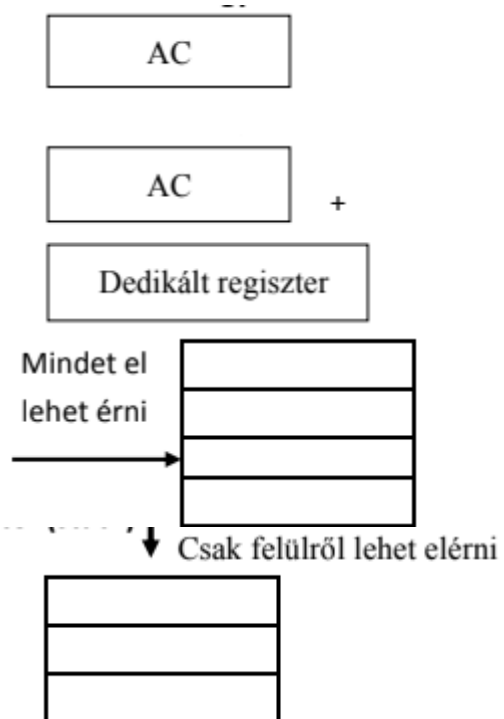
Linear memory

- Used by user-level programs
- Seen as one contiguous address space, but behind the scenes linear addresses are mapped to physical addresses
- User-level programs can address memory in a common way, physical memory is managed by the kernel
- Segmented memory model: applications can be broken up to diff. parts of memory
- Flat memory model: all of the linear address space is accessible

Registers

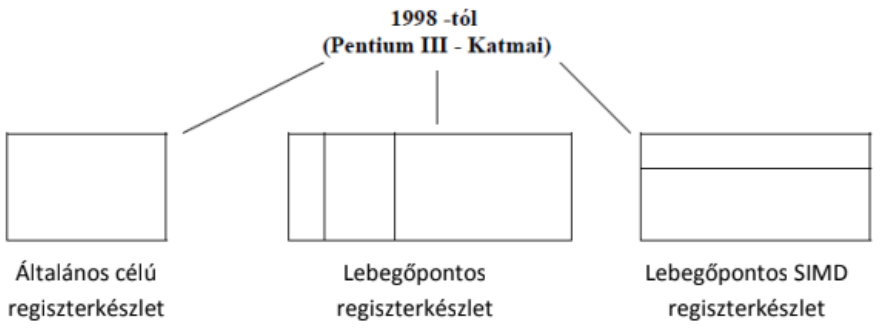
Simple

- just accumulator
  - Result must be saved out
  - Multiple results?
- accumulator + dedicated register
  - Faster, more expensive
  - Often underutilized
- universal register set
  - Any element is accessible
- stack register
  - Only the topmost element is accessible
  - This can be a bottleneck



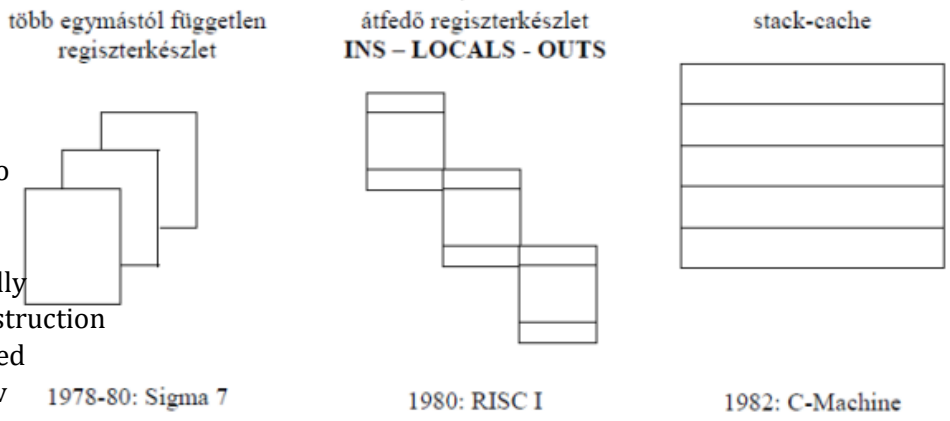
Registers for different datatypes

- For speeding up data processing
- Mostly used for floating points
- 80s: extra floating point register set
- 1998: + extra floating point SIMD register set
- SIMD: Single Instruction Multiple Data



Multiple register set

- **Independent register set**
  - Ideal for independent processes
  - Not faster when parameters have to be transferred
- **Overlapping register set**
  - Calling instruction OUTS is physically the same as the INS of the called instruction
  - No inter-register operation is needed
  - Fixed number of registers, overflow
- **Stack cache**
  - Speed of cache + organization of stack
  - Compiler assigns a variable length activation record (register set) to every instruction
  - Caller OUTS=Called INS
  - Limited only by physical size of stack-cache, no overflow



Data manipulation tree

- Shows the potential data-manipulations
- Parts of a data manipulation tree:
  - Data types: fix point, floating point, Boolean..
  - Operations: +,-,*,/
  - Operand types: rrr, rmr, mmm.. (r=register, m=memory)
  - Addressing modes: R+D, PC+D, RI+D.. (D=displacement, R=register, PC=program counter, RI=index reg.)
  - Machine code (instructions): 10011110

Data types

- Elemental:
  - numeric (integer (fix point), floating point, binary coded decimal)
  - character(ASCII, Unicode)
  - logical (1 byte, 2 byte, 4 byte, variable length)
- Complex: list, tree, set, string, stack

Addressing modes

- **Immediate:** operand is stored in the instruction
- **Indirect:** address of the address of the operand is stored in the instruction
- **Direct:** address of the operand is stored in the instruction
- **Register:** name of the register containing the operand is stored in the instruction
- **Indexed:** address is calculated by summing addresses in the specified registers

SzA3. Sequential instruction execution

(Arithmetical instructions, load and store, conditional and unconditional jumps)

Computer instructions

opcode	operand

- Set of machine language instructions that a CPU understands and executes
- Size is variable, in modern processors 16-64 bit
- Instruction: **opcode** – what operation, **operand** – register, memory address, literal data
- **Arithmetic instructions**
  - Perform +,-,*,/
  - Result is stored in the first operand
  - Flags are set by the CPU: Zero, Sign, Parity, Carry

Load and store

Fetch instruction:
MAR <- PC
MDR <- [MAR]
IR <- MDR
PC <- PC + 1

Decode instruction

Load operand:
DEC <- IR
MAR <- DEC (address of operand part)
MDR <- [MAR]
AC <- MDR

Arithmetic instruction:
DEC <- IR
MAR <- DEC (address of operand part)
MDR <- [MAR]
AC <- AC (+ | - | * | /) MDR

Store operand:
DEC <- IR
MAR <- DEC (address of operand part)
MDR <- AC
[MAR] <- MDR

Unconditional jumps (goto)

- Update the PC with the operand of the instruction (may be address, offset, or dynamically calculated address)

Conditional jumps (evaluation)

- If true, update PC with the operand
- If false, execute next instruction

SzA4. Types of instructions and operands, state space and state transitions

(types of instructions and operands, architectural types, regular architectures, state space and state transitions)

Types of instructions

- Arithmetic and logical (+,-,*,/,and,or)
- Data transfer (between registers, or register and memory -- load, store, move operations)
- Control (branch, jump, unconditional, conditional, procedure calls)
- Floating point, Decimal, String
- System (operating system, virtual memory management)

Types of instruction addressing

op – operandus, s – source, d - destination, @ - operation

- 4 operand: opd = opS1@opS2, op4 op4: next operation
- 3 operand: opd = opS1@opS2 next operation in PC
- 2 operand: ops1=ops1@ops2
- 1 operand: AC LOAD[100], ADD[102], STORE[100]
- 0 operand: STACK (push, pop), NOP – no operation

Types of operands:

- Accumulator, Memory, Register, Stack, Immediate (value)

Types of architectures:

Stack-based architecture

- Stack is Last in First Out
- Stack implemented in main memory
- Only one pointer, to the topmost element of the stack
- Two operands: push (put onto top of stack) and pop (get topmost element)
- Get (pop) arguments from stack and push the result back

Accumulator based architecture

- One register for operations, called accumulator
- Stores one of the operands, the other is read from memory
- Result is stored in the accumulator

General purpose register architecture

- There are general purpose registers that can be the source and destination of operations
    - Use of registers may be restricted -> special-purpose register architecture
- Two (sub)classes of register computers
    - Register-memory ISA: memory can be accessed from any instruction, one of the operands for ADD operation may be in memory, while the other is in a register.
    - Load-store ISA: memory can be accessed only by load and store instruction, both operands for an ADD operation must be in registers before the ADD.

State space and state transitions

- State space contains information
- Visible: PC, State Indicator, etc
- Transparent (only visible by system): Virtual Memory Management, Interrupts, Stack Management

A finite-state machine (FSM) is a mathematical model of computation. It is an abstract machine that can be in exactly one of a finite number of states at any given time. The FSM can change from one state to another in response to some inputs; the change from one state to another is called a transition.

- State indicators: ‘flags’ for handling exceptional results (e. g. zero, negative, overflow, carry)
- Different flag sets are assigned for every register set type
- State transitions
    - Program Counter: increment, overwrite (jump)
    - Flag: set, reset, save, load, clear

The Arithmetic Logic Unit I.

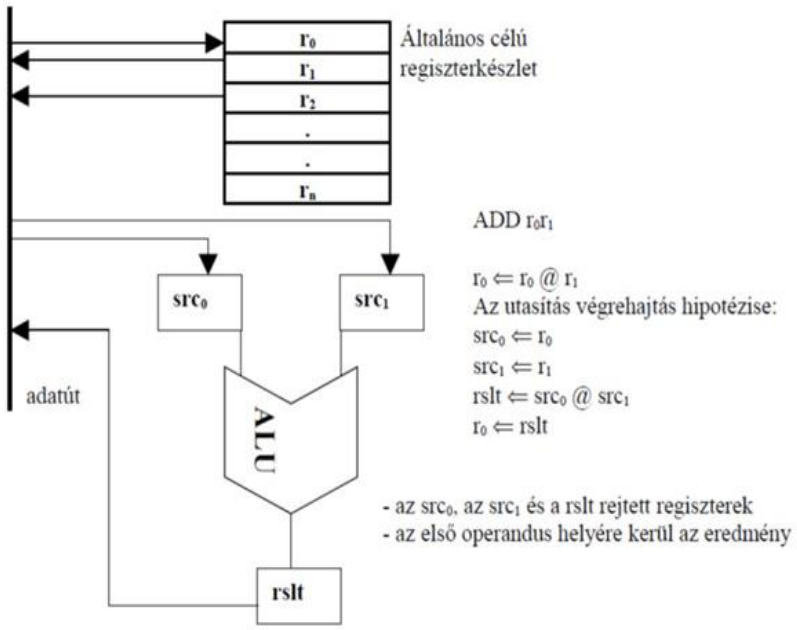
a. **parts of the ALU,**

b. **data paths and switching points,**

c. **the single bit adder,**

d. **the n-bits sequential and parallel adder,**

e. **n-bits adder with carry**

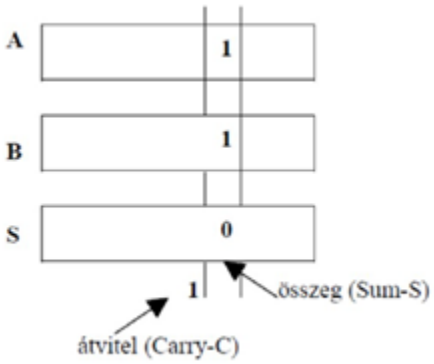


Parts of the Arithmetic and Logic Unit (ALU)

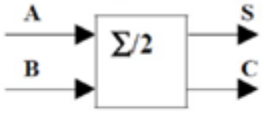
- **Registers**
  - Visible (r0,r1,r2..)
  - Transparent (src0,src1,rslt
  - Buffer registers
- **Data paths**
  - The ALU does not access main memory -> role of the Control Unit
  - Does not interpret memory addresses -> not a bus, but a channel
  - Single path (sequential)
  - Two path (input1 + output, input2) (parallel)
  - Three path (input1, input2, output) (parallel)
- **Switching points (transistors)**
  - Inward: 2 states (open or closed)
  - Outwards: 3 states (1, 0, closed)
- **ALU (execution unit)**
  - Fix point, floating point, BCD, logical, increment operations

Fixed-Point Addition

- Since addition makes the basis of every arithmetical operation, it must be fast
- **Half-adder**:
  - It does not handle carry inputs
  - Input: A and B bits
  - Output: R result and C carry



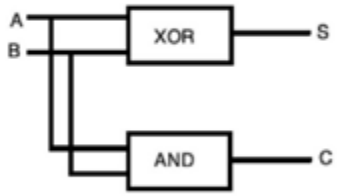
Jelölése: (félösszeadó)



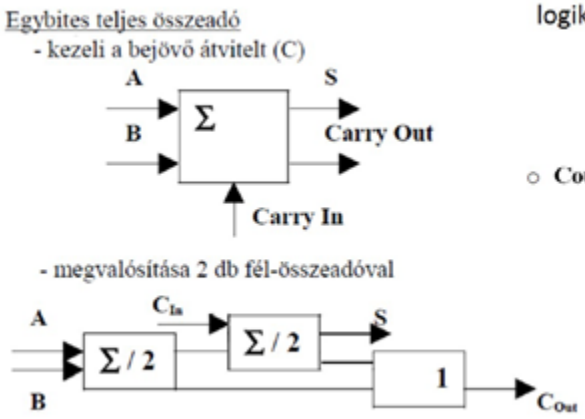
Igazságtáblázat:

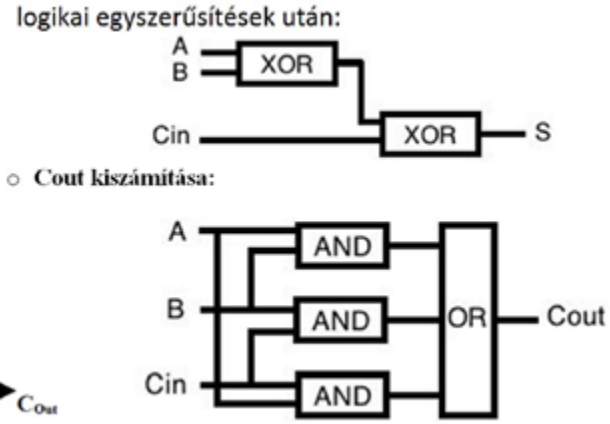
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Megvalósítása:

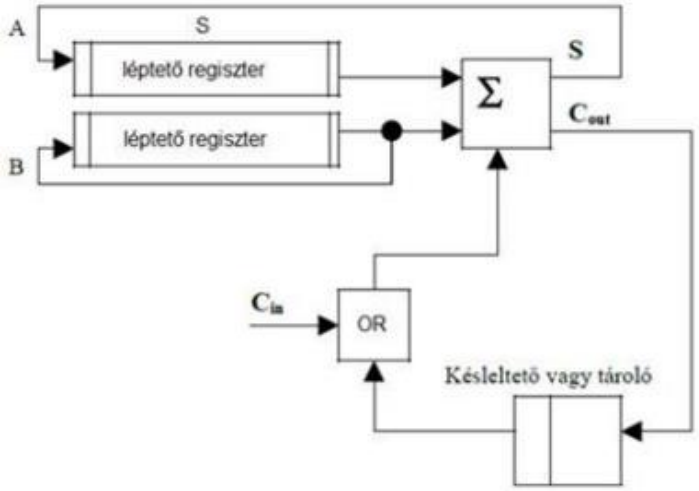


- **Full-adder**:
  - It does handle carry inputs
  - Input: A, B and Cin bits
  - Output: R result and Cout carry





- **N-bit Sequential Adder:**
  - Numbers are typically represented by multiple binary digits -> add them together

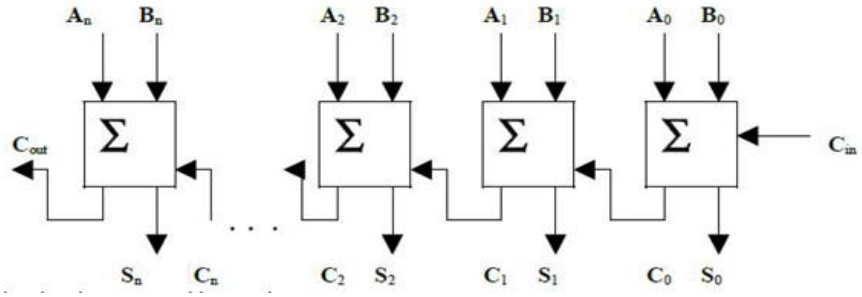


- **How it works:**
  - Inputs are placed in two queue registers A and B
  - For each clock cycle the queue registers forward one bit from LSB to MSB
  - The actual result is placed back into register A (this will store the result at the end)
  - During the first cycle we forward the input carry to the adder circuit
  - The produced (own) carry bits are placed in a container and get forwarded in the next cycle to the adder
  - The execution's duration is n * t, where t is the full adder's execution time, n is the number of bits

N-bit Parallel Adder:

- Contains n number of full adder
- The carry propagates through the adders, starting with the least significant adder, and if it produces 1 then it is added with the second least significant adder and so on
- If Cin is 0 then the execution time is t, if 1 then it is t * n (because each adder waits for the previous to produce carry)

N bites párhuzamos összeadó



Carry Look Ahead (CLA)

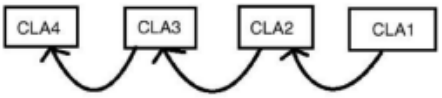
- With the help of CLA, adders do not have to wait for carry bits to be produced by the previous adder
- The circuit defines the carry bits for each adder
- Provides faster execution but uses extra circuitry which gets more complicated as the number of bits increases

$\mathbf{C_{out} = AB + AC_{in} + BC_{in}} = A*B + C_{in} * (A+B)$

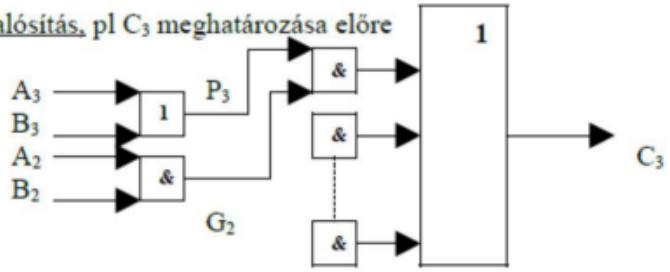
Az egyes bithelyiértékeken képződő átvitel függ a két bejövő operandustól és a kívülről bejövő átviteltől.

$A*B = G$ (generate)
$A+B = P$ (propagate)
$C_{out} = A*B + C_{in} * (A+B) = G + C_{i-1} * P$

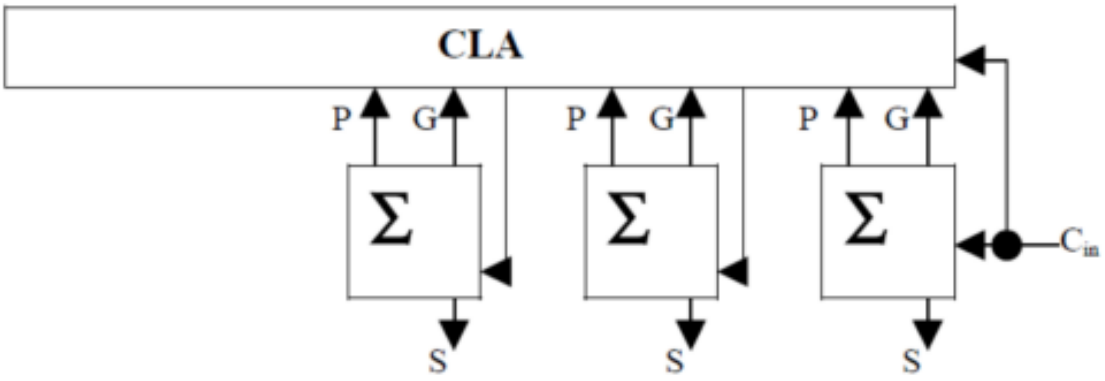
$C0 = G0 + C_{in}*P0$
$C1 = G1 + C0 * P1 = G1+(G0 + C_{in}P0)\ P1 = G1 + G0*P1 + C_{in}*P0*P1$
$C_i = G_i + C_i\text{-}1 * P_i$ => OR AND kapuk, T = 2t + 1t (lényegesen gyorsabb)



Megvalósítás, pl C_3 meghatározása előre



Az egybites teljes összeadót kiegészítjük 2 új áramkörrel (egy ÉS, és egy VAGY kapuval), hogy meghatározzuk a P és a G értékét.



The Arithmetic Logic Unit II.

f. **properties of BCD encoding,**

g. **the properties, formats, and encoding of floating-point numbers,**

h. **rounding and exception handling,**

i. **algebraic operations and their implementations,**

j. **the IEEE 754 standard**

Binary Coded Decimals (BCDs)

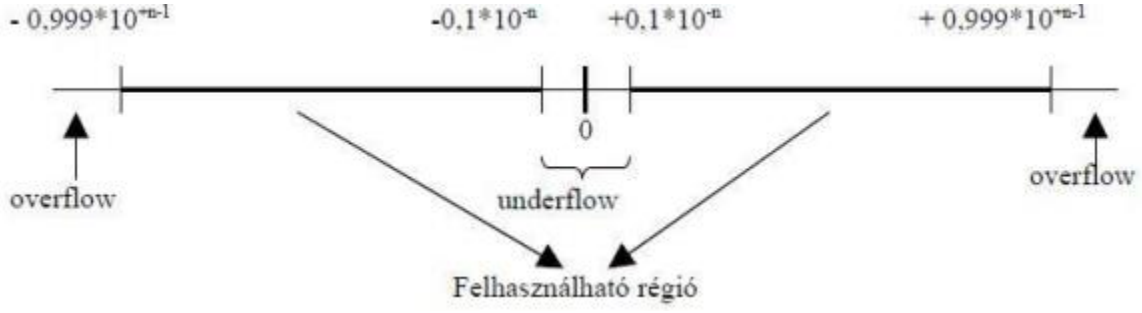
- FX number representation for broken numbers is imprecise
- With FP numbers is “more precise”
- In case of BCD, we convert from decimal to binary and then back, more precise
- A digit (0 to 9) can be represented by 4 bits
  - 4 bits of a digit are called a tetrade
  - 0000 -> 1001
- Invalid tetrades (bigger than 9) have to be noticed and corrected
  - Tetrade is invalid if the first bit is 1, and the second or third bit is 1
  - To correct, we subtract 10 and step 10 (1111 -> 0101 -> 0001 0101)
- **Zoned**:
  - A byte contains two tetrades (a higher Zone and a lower BCD number)
  - Zones can be for example ‘1111’
  - For signed numbers, the rightmost (LSB) holds the sign value
- **Packed**:
  - The first bit of first byte is the sign (the other bits are unused)
  - The remaining 9 bytes contain the BCD numbers (Intel: 10 bytes)
- **Operation on BCDs**
- **BCD adder**:
- 3 stages: addition, invalid tetrade filtering and correction
- Contains both full adders and half adders
- Its advantage is that it is completely precise
- Its drawback is that it has more complex logic, slower (but not today) and consumes more memory
- It is used in bank transactions the most

Floating point:

$\pm M * r^{\pm k}$

M : mantissza
r : radix (számrendszer alapja)
k: karakterisztika

- Fixed point (integers) have a small range, and they are imprecise with fractions
- **Binary format with three fields**: Sign, Mantissa, Exponent
- Sign is 0 for positive, 1 for negative
- **Normalization**: We move the point before the first significant number (123.45 -> 0.12345)
- **Range:** depends on the number of bits for the characteristic



- **Precision:** depends on the number of bits for the mantissa
- **Hidden bit:** improve precision using hidden bit
    - Thanks to normalization, the bit after the point will always be 1
    - There is no need to store this bit, we can include one more bit for precision
- **Guard bits:** they guard precision
    - Both the mantissa and floating-point execution unit is 4-15 bit longer than the stored format (e.g. 32+11)
    - Add an additional valuable bit with hidden bit
    - Helps with rounding
- **Formats**
    - **Arithmetic format:** all mandatory operations are supported
    - **Basic format:** covers three binary and two decimal formats
    - **Interchangeable format:** fixed-length binary encoding for data interchange between platforms
- **Encoding**
    - Mantissa: two's complement
    - Exponent: biased

Rounding rules

The standard defines five rounding rules. The first two rules round to a nearest value; the others are called directed roundings.

- **Roundings to nearest**
    - **Round to nearest, ties to even** – rounds to the nearest value; if the number falls midway, it is rounded to the nearest value with an even least significant digit; this is the default for binary floating point and the recommended default for decimal.
    - **Round to nearest, ties away from zero** – rounds to the nearest value; if the number falls midway, it is rounded to the nearest value above (for positive numbers) or below (for negative numbers); this is intended as an option for decimal floating point.
- **Directed roundings**
    - Round toward 0 – directed rounding towards zero (also known as truncation).
    - Round toward +∞ – directed rounding towards positive infinity (also known as rounding up or ceiling).
    - Round toward −∞ – directed rounding towards negative infinity (also known as rounding down or floor).

Exception handling

The standard defines five exceptions, each of which returns a default value and has a corresponding status flag that (except in certain cases of underflow) is raised when the exception occurs. No other exception handling is required, but additional non-default alternatives are recommended (see below).

The five possible exceptions are:

- **Invalid operation**: mathematically undefined, *e.g.*, the square root of a negative number. By default, returns qNaN.
- **Division by zero**: an operation on finite operands gives an exact infinite result, *e.g.*, 1/0 or log(0). By default, returns ±infinity.
- **Overflow**: a result is too large to be represented correctly (*i.e.*, its exponent with an unbounded exponent range would be larger than *emax*). By default, returns ±infinity for the round-to-nearest modes (and follows the rounding rules for the directed rounding modes).
- **Underflow**: a result is very small (outside the normal range) and is inexact. By default, returns a subnormal or zero (following the rounding rules).
- **Inexact**: the exact (*i.e.*, unrounded) result is not representable exactly. By default, returns the correctly rounded result.

- **Addition:** exponents must be equal
    - If they are not, the number with the smaller exponent will be changed, the point will be moved leftward, and its exponent will be increased
    - If exponents are equal, we add the mantissas together, exponent is unchanged.
- **Multiplication:** mantissas multiplied; exponents added
- **Division:** mantissas divided; exponents subtracted
- **The ALU has a dedicated FP execution unit**
- One unit for calculating mantissas
- And one for calculating exponents
    - This unit only needs addition and subtraction, therefore it is simpler
- The two units operate in parallel

The IEEE Standard for Floating-Point Arithmetic (IEEE 754) is a technical standard for floating-point arithmetic established in 1985 by the Institute of Electrical and Electronics Engineers (IEEE). The standard addressed many problems found in the diverse floating-point implementations that made them difficult to use reliably and portably. Many hardware floating-point units use the IEEE 754 standard.

The standard defines:

- ***arithmetic formats****:* sets of binary and decimal floating-point data, which consist of finite numbers (including signed zeros and subnormal numbers), infinities, and special "not a number" values (NaNs)
- ***interchange formats****:* encodings (bit strings) that may be used to exchange floating-point data in an efficient and compact form
- ***rounding rules****:* properties to be satisfied when rounding numbers during arithmetic and conversions
- ***operations****:* arithmetic and other operations (such as trigonometric functions) on arithmetic formats
- ***exception handling****:* indications of exceptional conditions (such as division by zero, overflow, *etc.*)

Accelerating FX multiplications:

- Determine the number of bits for the result:
    - Bits of first number (m) and bits of second number (n)
    - Results length is <= m + n
- This means that the result must be stored in double length register
- Instead of single bit shifting, we operate on 2 bits at a time
- This means that we multiply a number with 2 bits at once
    - 00 – shift to left by 2
    - 01 – add 1 * the victim to the victim number and shift to left by 2
    - 10 – add 2 * the victim to the victim number and shift to left by 2
    - 11 – add 3 * the victim to the victim number and shift to left by 2

Booth's Algorithm

- If there are many 1s in a number, it means more addition operations
- Therefore, choose a number that has less 1s and is close to the target, then subtract the difference between the chosen and original number
- Example: 62 -> 00111110, but 64 -> 100000
- A * 62 = A * 64 – A * 2
- So we multiply by 64 instead of 62
- Then we multiply the original number (A) by 2, and subtract it
- This way we spared 40% of work

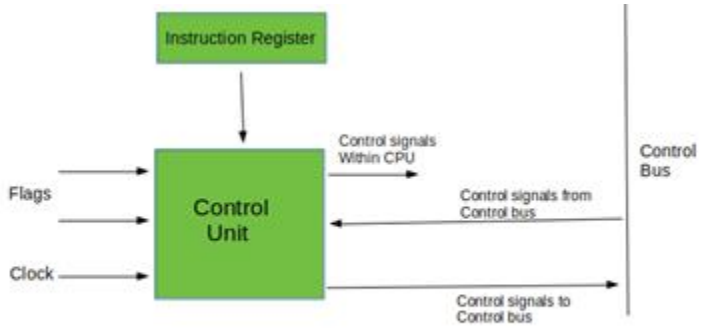
The Control Unit

k. Differences between hard-wired and microprogrammed CU,

l. implementation and operation of the hard-wired CU

Control Unit is the part of the computer's central processing unit (CPU), which directs the operation of the processor. It was included as part of the Von Neumann Architecture by John von Neumann. **It is the responsibility of the Control Unit to tell the computer's memory, arithmetic/logic unit and input and output devices how to respond to the instructions that have been sent to the processor.** It fetches internal instructions of the programs from the main memory to the processor instruction register, and based on this register contents, the control unit generates a control signal that supervises the execution of these instructions.

A control unit works by receiving input information to which it converts into control signals, which are then sent to the central processor. The computer's processor then tells the attached hardware what operations to perform. The functions that a control unit performs are dependent on the type of CPU because the architecture of CPU varies from manufacturer to manufacturer. **Examples of devices that require a CU are: Control Processing Units(CPUs), Graphics Processing Units(GPUs)**



Block Diagram of the Control Unit

Functions of the Control Unit:

- It coordinates the sequence of data movements into, out of, and between a processor's many sub-units.
- It interprets instructions.
- It controls data flow inside the processor.
- It receives external instructions or commands to which it converts to sequence of control signals.
- It controls many execution units(i.e. ALU, data buffers and registers) contained within a CPU.
- It also handles multiple tasks, such as fetching, decoding, execution handling and storing results

There are two types of control units: Hardwired control unit and Microprogrammed control unit.

Microprogrammed Control unit:

The fundamental difference between these unit structures and the structure of the hardwired control unit is the existence of the control store that is used for storing words containing encoded control signals mandatory for instruction execution.

In microprogrammed control units, subsequent instruction words are fetched into the instruction register in a normal way. However, the operation code of each instruction is not directly decoded to enable immediate control signal generation but it comprises the initial address of a microprogram contained in the control store.

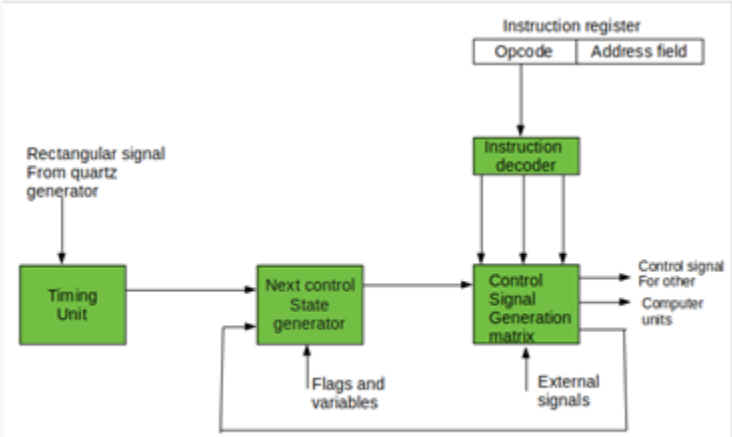
- **Advantages**: flexible, more transparent, cheaper
- **Disadvantages**: always slower

- The control signals associated with operations are stored in special memory units inaccessible by the programmer as Control Words.
- Control signals are generated by a program are similar to machine language programs.
- Micro-programmed control unit is slower in speed because of the time it takes to fetch microinstructions from the control memory.

Hardwired Control Unit:

In the Hardwired control unit, the control signals that are important for instruction execution control are generated by specially designed hardware logical circuits, in which we cannot modify the signal generation method without physical change of the circuit structure. In the instruction decoder, the operation code is decoded.

As a result, few output lines going out from the instruction decoder obtains active signal values. These output lines are connected to the inputs of the matrix that generates control signals for executive units of the computer. This matrix implements logical combinations of the decoded signals from the instruction opcode with the outputs from the matrix that generates signals representing consecutive control unit states and with signals coming from the outside of the processor, e.g. interrupt signals.



Block diagram of a hardwired control unit of a computer

Control signals for an instruction execution have to be generated not in a single time point but during the entire time interval that corresponds to the instruction execution cycle.

A number of signals generated by the control signal generator matrix are sent back to inputs of the next control state generator matrix. When a new instruction arrives at the control unit, the control units is in the initial state of new instruction fetching.

When an external signal appears, (e.g. an interrupt) the control unit takes entry into a next control state that is the state concerned with the reaction to this external signal (e.g. interrupt processing).

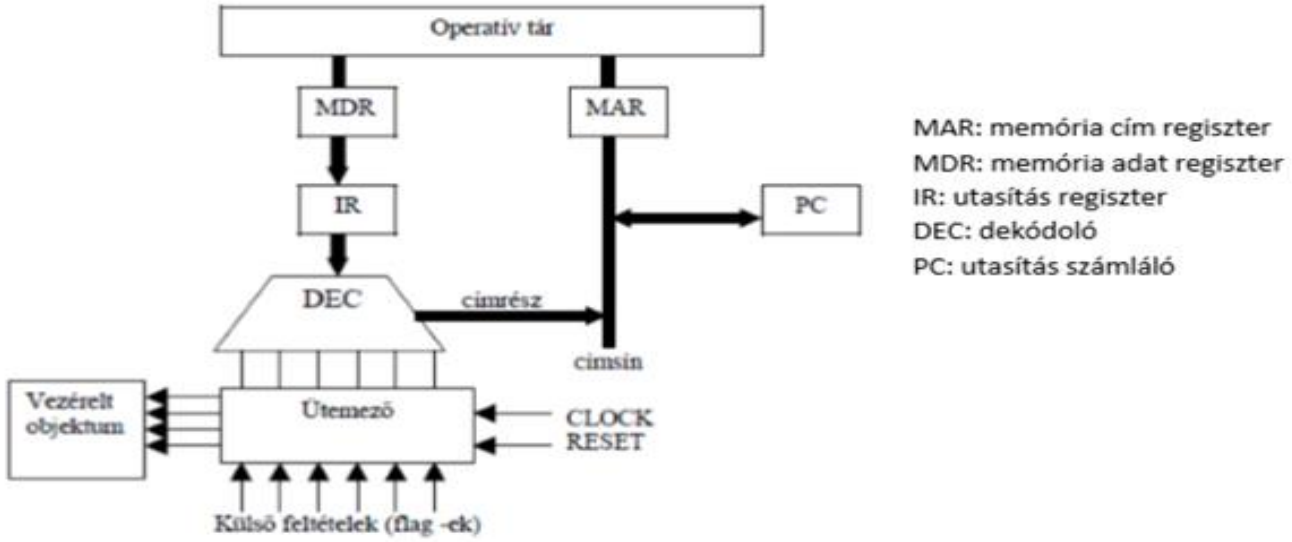
The last states in the cycle are control states that commence fetching the next instruction of the program: sending the program counter content to the main memory address buffer register and reading the instruction word to the instruction register of computer.

The control hardware can be viewed as a state machine that changes from one state to another in every clock cycle, depending on the contents of the instruction register, the condition codes and the external inputs. The outputs of the state machine are the control signals. The sequence of the operation carried out by this machine is determined by the wiring of the logic elements and hence named as “hardwired”.

- Fixed logic circuits that correspond directly to the Boolean expressions are used to generate the control signals.
- Hardwired control is faster than micro-programmed control.

- A controller that uses this approach can operate at high speed.
- RISC architecture is based on hardwired control unit
- Advantages:
    - Extremely fast
- Disadvantages:
    - Not transparent
    - Too complex, rigid, hard to modify
    - expensive

Huzalozott vezérlés:



Ütemező feladata:

1. vezérlőegység többi blokkjának a vezérlése
2. vezérelt objektumok szekvenciális vezérlése (MEM, ALU,…)

Forrás és célregiszterek:

- ALU (AC, általános regiszterek)
- vezérlőbusz regiszterei (IR, PC)
- memóriával kapcsolatos regiszterek (MAR, MDR)
- I/O rendszerekhez kapcsolódó regiszterek és vezérlők (parancsregiszterek, adatregiszterek, állapotregiszterek)

Módosító áramkörök:

- összeadás
- léptetés
- inkrementálás
- invertálás
- komparálás
- stb.

Semiconductor Memories

m. **properties,**

n. **classification,**

o. **different types of DRAMs, their operation and timing,**

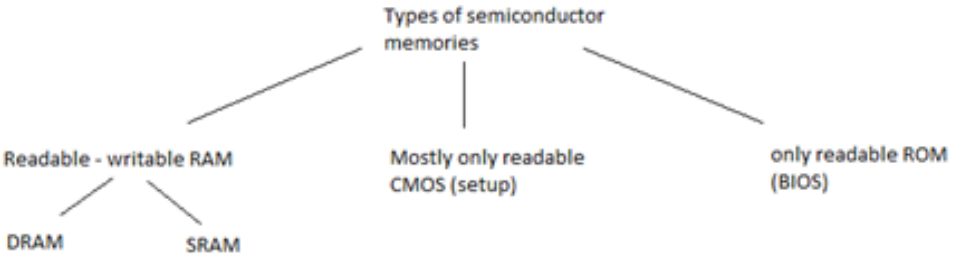
p. **the Reading cycle,**

q. **different types of DIMMs**

The early systems used ferromagnetic memory but the semiconductor memories long since vanquished the magnetic memory.
Semiconductor memory is a digital electronic semiconductor device used for digital data storage, such as computer memory. It typically refers to MOS memory, where data is stored within metal–oxide–semiconductor (MOS) memory cells on a silicon integrated circuit memory chip.
The basic element of a semiconductor memory is the memory cell. Although a variety of electronic technologies are used, all semiconductor memory have the following properties.

- They exhibit two (semi)stable states (represents binary 0 and 1)
- They are capable of being written into at least once to set the state
- They are capable of being read unlimited number (to read out the state)

Classification:



CMOS:
Complementary metal–oxide–semiconductor (CMOS), also known as complementary-symmetry metal–oxide–semiconductor (COS-MOS), is a type of MOSFET (metal–oxide–semiconductor field-effect transistor) fabrication process that uses complementary and symmetrical pairs of p-type and n-type MOSFETs for logic functions.[1] CMOS technology is used for constructing integrated circuit (IC) chips, including microprocessors, microcontrollers, memory chips (including CMOS BIOS), and other digital logic circuits.

ROM:

- Read-only memory (ROM) contains permanent data that cannot be changed.
- A ROM is non-volatile so no power source is required to keep the bit values in memory. Application of ROMs
    - microprogramming of CPUs
    - initialization and boot code of the computer (e.g. EFI or BIOS)
    - fixed configuration data
- A ROM is created like any other integrated circuit with the data is wired into the chip as the fabrication process.
    - There is a large fixed cost which is feasible only with large production volumes. There is no way to correct even one bit error. Any modification needs a new fabrication process
- For a small quantities a less expensive alternative is the Programmable ROM (PROM)
    - It can be written only once

    - The price of the PROM circuit is higher than the price of the ROM and it needs a programmer device but there is no fixed circuit fabrication cost
- Another variation is read-mostly memory (the read operations are more frequent than write). There are three main types: EPROM, EEPROM, FLASH

RAM technology is divided into two technologies:

Dynamic RAM (DRAM):

- Dynamic RAM (DRAM) is made with cells that store data as charge on capacitors
- The presence or absence of charge in a capacitor interpreted as 1 or 0
- Because capacitors have a natural tendency to discharge DRAMS require periodic charge refreshing to maintain storage
    - The term dynamic refers to this tendency of the stored charge leak away, even with power continuously applied
- In practice: operative memory

Static RAM (SRAM)

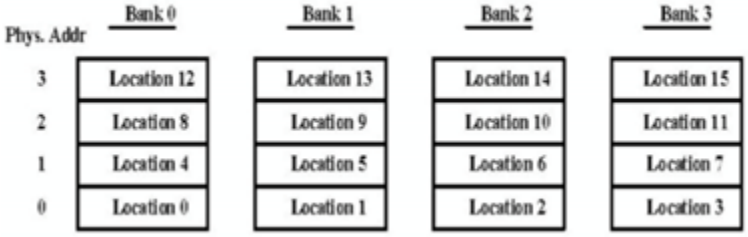
- A DRAM is essentially an analogue device. In contrast the SRAM building from same logic elements as the CPU
- A static RAM will hold its data as long as the power supplied
- In SRAMS binary data stored in traditional flip-flop logic configuration
- In practice: cache
- Advantages:
    - Simplicity – a refresh circuit is not needed
    - Performance
    - Reliability
    - Low idle power consumption

DRAM vs. SRAM

- DRAM: needs to be refreshed vs SRAM: hold its data as long as the power supplied
- DRAM: writing time:60-100ns, reading time: 60-100ns SRAM: writing time: 10-25ns, reading time: 10-25ns faster than DRAM
- DRAM: cheap and easy to produce, high power consumption, SRAM: harder to produce, low power consumption

Interleaved memory

- Main memory is composed of a collection of DRAM memory chips
- A number of chips can be grouped together to form a memory bank
- It is possible to organize the memory banks in a way known an interleaved memory
- Each bank is independently able to service a memory read or write operation
    - system with K banks can service K operations at one time



Types of DRAM:

- Classic DRAM (asynchronous interface)
- SDRAM (Synchronous DRAM): higher performance, dominant since 2000
    - SDR SDRAM (Single Data Rate)

- DDR SDRAM (Double Data Rate)

SDRAM:

- Unlike the traditional DRAM which is asynchronous the SDRAM exchanges data with the processor synchronized to and external clock signal
  - Running at the full speed of CPU/memory bus without any wait states
- SDRAM employs so called burst mode which enables sequential access of memory without continuously resending the addressing information
- SDRAM has a multiple-bank internal architecture for supporting on-chip parallelism

SDR SDRAM:

Synchronous dynamic random-access memory (SDRAM) is any dynamic random-access memory (DRAM) where the operation of its external pin interface is coordinated by an externally supplied clock signal. Typical clock frequencies are 100 and 133 MHz. Chips are made with a variety of data bus sizes (most commonly 4, 8 or 16 bits), but chips are generally assembled into 168-pin DIMMs that read or write 64 (non-ECC) or 72 (ECC - Error-correcting code memory) bits at a time.

Important timing parameters:

- **tCL (CAS Latency)** : the time between supplying a column address and receiving the corresponding data. Again, this has remained relatively constant at 10–15 ns through the last few generations of DDR SDRAM.
- **tRCD (RAS to CAS delay):** the minimum time between opening the bank and issue column select command
- **tRAS:** time between activation and the closure of the bank (precharge)
- **tRP, tRow Precharge time:** mandatory wait time after bank queue closing before opening new bank queue
- **tRC:** cycle time minimum time between reads from the same bank

A read cycle

- Open bank
- Read blocks of columns from opened rows
- Close bank (precharge)
- At least tRP wait before reopening this bank

The best case is that when we are reading from opened bank rows, because then we can continuously transfer data at each memory-clock signal.
Trends: not compatible with each other, clock frequency increases continuously – one factor of the increase of bandwidth, the number of bits transmitted in parallel in the pre-fetch increases continuously, the size of the chip decreases, the power consumption decreases- the cooling demand decreases

DDR SDRAM:

- DDR SDRAM dramatically improves the data rates of DRAMs (and SDRAMs)
- There are three ways to achieve higher data rates
  - the data transfer is synchronized to both the rising and falling edge of the clock (it doubles the transfer rate)
  - DDR uses higher clock rate on the bus to increase transfer rate
  - a buffering scheme is used (with internal prefetching)
- Extensions of this idea are the current (2012) techniques being used to increase memory access rate and throughput. Since it is proving difficult to further increase the internal clock speed of memory chips, these chips increase the transfer rate by transferring more data words on each clock cycle
  - DDR2 SDRAM – Transfers 4 consecutive words per internal clock cycle
  - DDR3 SDRAM – Transfers 8 consecutive words per internal clock cycle.
  - DDR4 SDRAM – Transfers 16 consecutive words per internal clock cycle

Chip logic:

- Semiconductor memory comes in packed chips
- Each chip contains an array of memory cells
- For semiconductor memories the key design issues is the number of bits of data that may be read/written at a time
    - The array is organized into W words of B bits each
    - For example, a 16-Mbit chip could be organized as 1M of 16-bit words
- A memory array is typically consist of rows and columns
- Because one chip typically store only few bits (e.g. 4 bits) per address we need combine more chip onto a memory module

DIMM:

DIMM (dual in-line memory module) is a type of computer memory that is natively 64 bits, enabling fast data transfer. DIMM is a module that contains one or several random access memory (RAM) chips on a small circuit board with pins that connect it to the computer motherboard. The DIMM stores each data bit in a separate memory cell. DIMMs use a 64-bit data path, since processors used in personal computers have a 64-bit data width. DIMMs are typically used in desktop PCs, laptops, printers and other devices.

Registered DIMMs (RDIMMs): Also known as buffered memory, RDIMMs are often used in servers and other applications that require robustness and stability. RDIMMs feature on board memory registers that are placed between the memory and the memory controller. The memory controller buffers command, addressing and clock cycling, and directs instructions to the dedicated memory registers rather than directly accessing the DRAM. Consequently, the instructions could take about one CPU cycle longer. However, the buffering reduces the strain on the CPU's memory controller.

ECC DIMM module: on one side, the 9^{th} DRAM chip is used for storing of parity bit or ECC bit. With the help of the parity bit, the memory controller is able to discover 1 error (cannot correct it). Furthermore, it cannot consistently detect multiple errors.

Interrupt (sub)System

r. **definition,**

s. **reasons of interrupts,**

t. **the process of interrupt handling,**

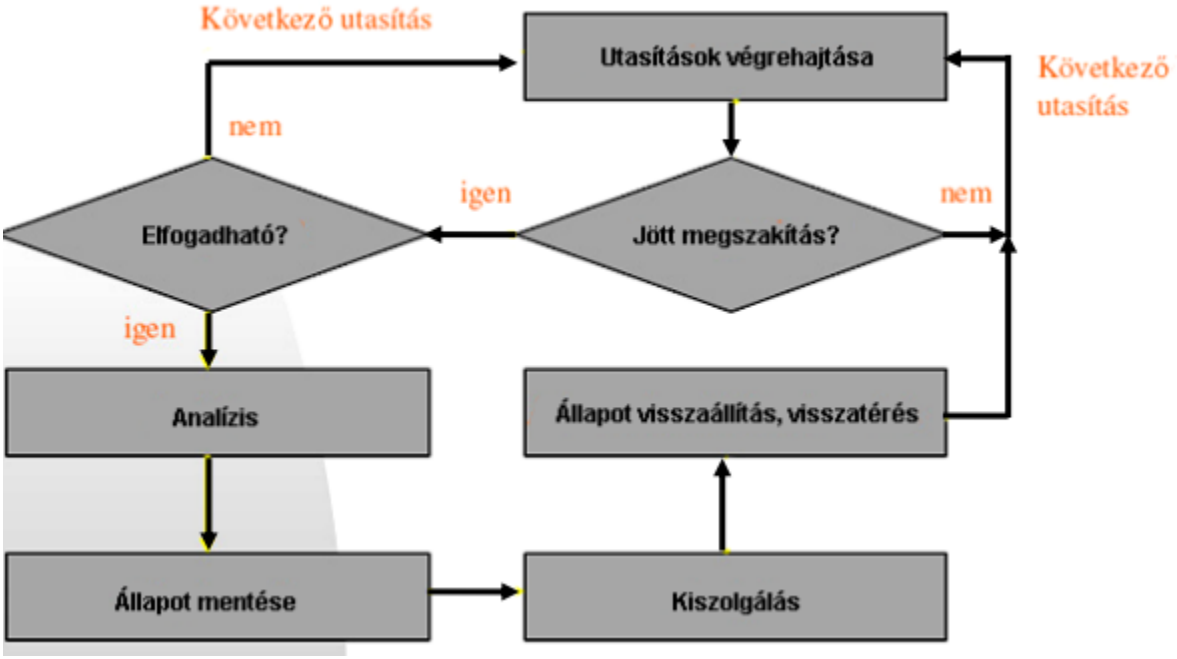
u. **classification,**

v. **one- and multilevel interrupt systems**

Interrupts

The system handles unexpected events from the perspective of execution

Goal: in a changing environment provide optimal workflow



Sources of interrupts:

1. **Machine errors**: Must be handled immediately, affects the whole system!
   - Typically hardware error
   - Devices can detect this with error signaling code
   - Power-supply errors
   - cooling system errors
2. **I/O sources:**
   - Peripheral events (e.g., printer, cpu requests printer, printer sends interrupt when available)
3. **External sources:**
   - Network events
4. **Programming sources:** after execution of instruction
   - Intentional (example OS, BIOS calls)
   - Programming errors:
     - Memory protection violation (program uses address that doesn't belong to it)
     - Memory overflow
     - Arithmetical or Boolean operation interrupts

Interrupt flow:

A device sends an interrupt request to the CPU

- It activates the INTR control line

It is accepted if:

- The actually running operation can be interrupted
- The level of priority of interrupt is high enough
- The interrupt is not blacklisted

The CPU signals the acceptance of the interrupt (INTACK)

The sender of interrupt deactivates the INTR

Hardware-related tasks:

- The CPU saves the currently executed program's state information in a stack
  - PC + state registers
- Loads the interrupt-manager program's starting address and updates the PC

Software-related tasks:

- Saving the currently running program's data
- Identifying the source of interrupt
- Serving the interrupt
- Loading the program's data back

Levels of interrupts:

One-level interrupt: Two levels, normal and interrupt. Third, higher priority interrupt cannot be activated, has to wait

multi-level interrupt: Multiple levels, higher priority can interrupt lower ongoing interrupt

multi-level multi-lane interrupt: multiple priorities and within them classes 0/a>1/a>1/b>2/a>2/b

Megszakítások csoportosítása

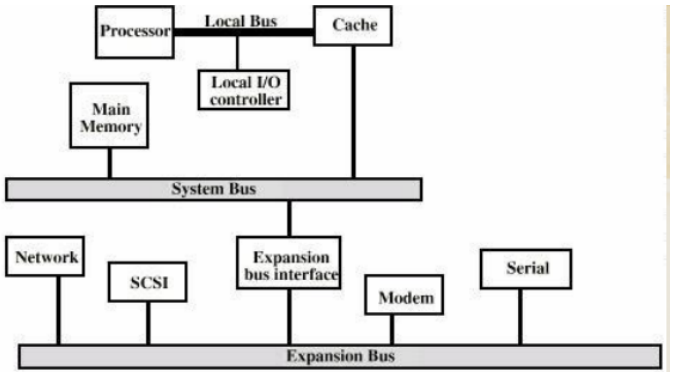
Aszinkron - véletlenszerű	**Szinkron** – a program végrehajtása során mindig ugyanott jelentkezik a probléma
Várható – OS szintű	**Nem várható** – Hardver hiba
Utasítások végrehajtása között fellépő - túlcsordulás	**Utasítások végrehajtása alatt fellépő** – HW vagy I\O hiba
A felhasználó által kért – oprendszer szolgáltatás	**A felhasználó által nem kért** – ALU vagy HW hiba
Megszakítás befejezése után a program futása folytatódik	Megszakítás befejezése után a program futása **nem** folytatódik.
A felhasználó által: maszkolható – nyomkövetés, vagy I\O kérés	**A felhasználó által: nem maszkolható** – súlyos hardverhiba

SzA10. External Bus

(definition, properties, classification, data, address and control lines, parallel and sequential buses (pros and cons), PCI and PCIe bus, bus arbitration methods; control of data movement; the FSB, the HyperTransport and the QPI)

Definition

- Dominant solution for computer system interconnection
- Communication pathway connecting two or more devices
- Shared transmission medium (multiple connected devices, only one transmits at a time)
- Consists of multiple lines, transmitting bits in parallel
- Standardized -> devices are easily interchangeable
- Transparent to the user

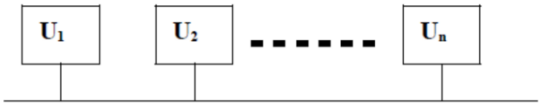


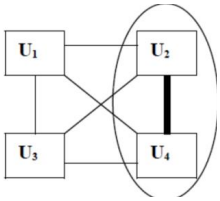
Direction

- simplex
- half-duplex
- duplex

Nature of transmission

- shared
  - cheap
  - easy to implement
  - control issues, error prone
- dedicated bus
  - fast and reliable,
  - expensive and too many wires
  - difficult to expand





Transferred data

- data lines
  - path for moving data between modules
  - may consist of 32 to hundreds of separate lines
  - the number of lines determines how many bits can be transferred at a time
- address lines
  - the source/destination of data on the data bus
  - width: determines the maximum possible memory capacity of the system
- control lines
  - control the access to and the used of the data and address lines

Connected areas

- Processor lines: processor-memory, bus controller
- Expansion bus: connect peripherals to processor-memory

Mode of transmission

- **Parallel**
  - Many lines, complex, expensive, delay skew, electromagnetic interference, noise, crosstalk, hard to extend
- **Serial**
  - One fast line is better than multiple slow, higher signaling frequency, does not have to deal with crosstalk or multi drop, expandable with software

PCI (Peripheral Component Interconnect)

- Popular high-bandwidth, processor independent bus that functions as a peripheral bus
- Compared to other bus specifications, delivers better system performance for high speed I/O devices
- Wasn't fast enough to serve new peripherals

Point to point Interconnection

- At higher data rates becomes more difficult to synchronize buses
- With higher core counts, we need more bandwidth, buses became a bottleneck
- Compared to shared buses, point-to-point interconnect has:
  - low latency
  - high data rate
  - better scalability

PCI-E

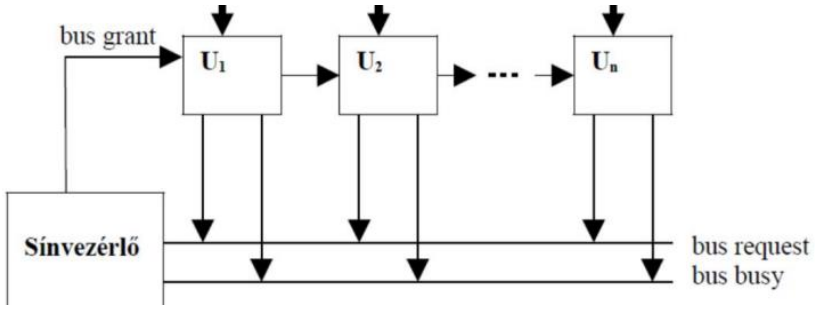
- point-to-point, high-capacity interconnect
- Physical layer
  - Consists of a number of bidirectional lanes (1,4,8…)
- Data link layer
  - Ensures reliable delivery of packets
- Transaction layer
  - Supports multiple address spaces and transaction types

PCI	PCI-e
Devices use the same shared parallel architecture	Point-to-point technology
All the devices use a common address, data, and control line	Devices are connected to control unit with the help of separated lines
Bus arbitration in case of multiple bus master	Full duplex transmission
Only one master can communicate in one direction at a time	Multiple units can communicate parallel at the same time
1 shared bus with high performance	PCI express link may contain multiple lanes (x1,x4 etc) → flexible

Bus Arbitration (busz foglalás)

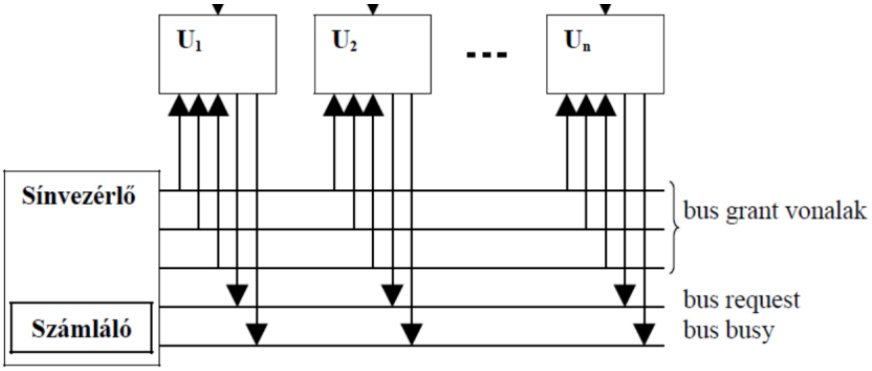
Serial bus arbitration

- **Hardware polling**



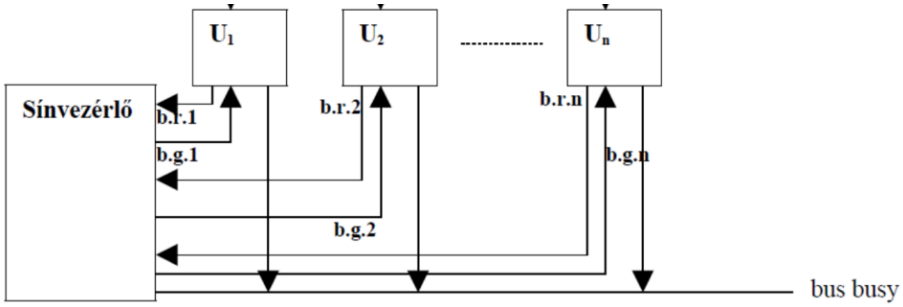
- **Advantages:** simple, cheap, unlimited number of units (theoratically)
- **Disadvantages:** hardware-based priority, units in the front may overshadow the units farther away, error prone

- **Software polling**



- **Advantages:** less error prone, software-based priority, flexible
- **Disadvantages:** more lines -> more expensive, # of units limited by bus grant lines

- **Independent requesting**



- **Advantages:** fastest response, priority controlled by bus controller
- **Disadvantages:** costly and complex, a lot of lines
- **Hidden bus arbitration:** two independent hardware control arbitration and data transfer -> the next unit can be selected while the data transfer is still ongoing -> used by PCI bus

Data transfer

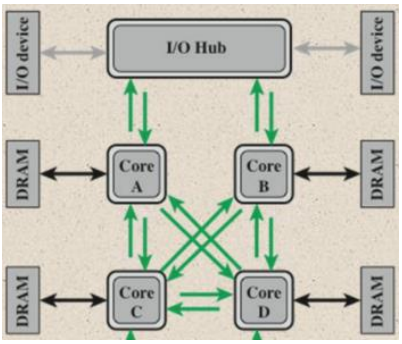
- **Synchronized**
  - Data transfer occurs in an interval previously known to sender and receiver, provided by clock signal
  - Both sender and receiver have their own clock signaler, they have to be synchronized
  - Cheap, simple
  - Slowest device will stall faster ones
- **Asynchronous**
  - The end of the current operation signals the start of the next one
  - One path: sender or receiver controlled, no acknowledgement

- Two path or handshake: sender receives acknowledgement

FSB (front side bus)

- communication interface that serves as the main link between CPU and other components through a chipset
- communication outside of the processor -> slow!
- Intel implemented chipset as North and South bridge, North for fast devices, South for others
- big overhead, addressing
- limited speed made it a major bottleneck
- not really used today

Intel's QuickPath interconnect



- Intel's replacement for the FSB
- Point-to-point interconnection
- Multiple direct connections
  - Direct pairwise connections
  - Eliminates the need for arbitration
- Layered protocol architecture
  - Physical: wires carrying the signal and logic to support transmission
  - Link: reliable transmission and flow control
  - Routing: directs packets through the fabric
  - Protocol: high-level packet format and rules
- Packetized data transfer
  - Data sent as packets
  - Headers and error control codes
- QPI layers
  - Phit: Physical unit
  - Flit: flow control unit
  - Packet: integer number of Flits

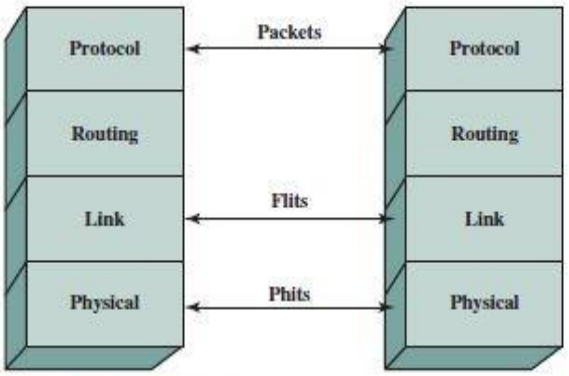


Figure 3.18 QPI Layers

HyperTransport

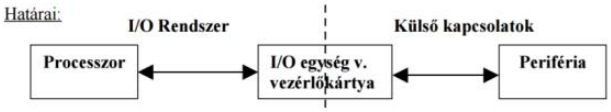
- AMD's replacement for the FSB
- low latency, high speed, point to point link
- packet-based
- parallel communication between memory and CPU
- packet based = 32 bit words
- has power management
- low pin count
- transparent to the OS
- Transport bus -> dedicated bus structure for memory and CPU (constant communication)
- HyperTransport for communication between the CPU and all other parts
- allows multiple processors to be interconnected
- separate links for CPU input/output

SzA11. Processor controlled I/O and DMA

(evolution of ~, programmed I/O, interrupt controlled I/O, I/O channel, DMA definition and implementation)

I/O Definition

- A system connecting the CPU-Memory pair with the external world



Evolution of I/O:

1. Processor controlled the peripheral devices
    a. The programmer needed to program it to manage I/O lines
    b. Time-consuming for the CPU
2. A dedicated I/O module controlled the peripheral devices
    a. Status flags of the module had to be polled continuously
3. The I/O module controls the peripherals (interrupts inform the CPU of different events)
    a. No need for continuous flag polling
4. Direct Memory Access
5. Channel: processes I/O instructions using the main memory
6. I/O Processor: processes I/O instructions using its own memory

I/O Types:

- Involving CPU (programmed I/O):
    - I/O operations are controlled by the CPU
    - For each I/O operation a CPU instruction is associated
    - Easy realization but huge CPU load
- Without CPU (additional I/O Control Unit):
    - The control unit should:
    - Generate memory addresses
    - Transfer data on the system bus
    - Allocate the system bus for its operations

Programmed I/O

- For each I/O operation the CPU issues an instruction
- CPU handles coding and decoding tasks and error situations
- Most primitive solution, very CPU intensive
- There is an address bus that transfers memory addresses and I/O addresses
- An additional line tells which type of address is on the bus at the moment

I/O Ports: those registers that the CPU communicates with the peripheral devices through

- Command Register: this includes the tasks from the CPU (what it desires to do)
- Data Registers: input (from peripherals, the CPU reads them) and output (for peripherals, the CPU writes them)
- Status Register: CPU reads status information of the peripheral
- Today Command and Status register as well as Input and Output register are unified

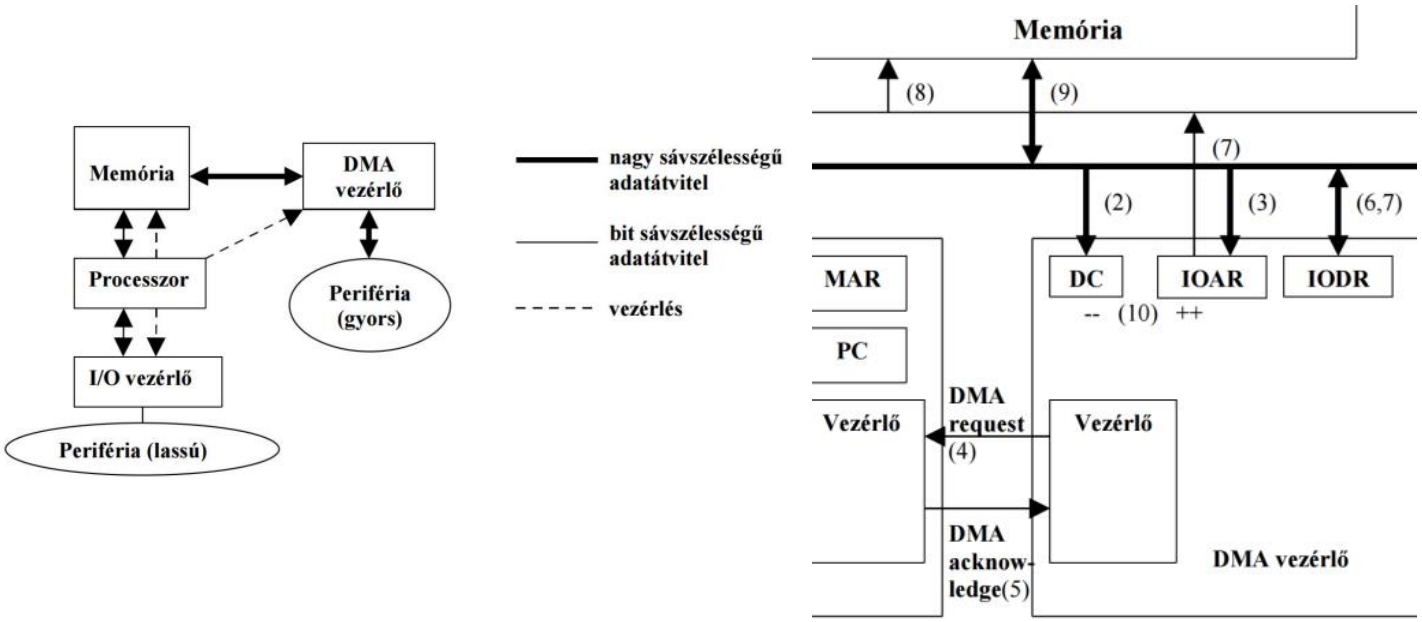
Programmed I/O

- **Unconditional Transfer**:
    - the receiver is always ready to gather data, we do not check success of transmission
        - E. g. LEDs
- **Conditional Transfer**:
    - “wait for flag”:
    - The CPU writes the command register, then reads the status register of the I/O controller
    - If it is “not ready” then reads the status again

- If it is “ready” then the CPU reads the data register
- This method causes millions of wasted CPU cycles (the CPU waits for the “ready” flag)

- **Interrupt Transfer**:
  - The CPU writes the command register, then starts doing something else
  - The I/O controller handles the writing in the data register and sets the “ready” flag and sends an interrupt to the CPU
  - The CPU then reads the status register of the I/O and start the reading of the data register
- **Address Space**:
  - Can be a dedicated I/O Address Space
    - Independent of memory
    - Simple but slow, inefficient for I/O device with huge data transfers
    - Keyboards, serial and parallel ports use this approach
  - Can be part of main memory
    - The I/O device sees a small part of the main memory where it writes and reads data
    - Much faster than its dedicated counterpart
    - Found in monitors with visual data

Direct Memory Access (DMA)



- Used where high volume of data is transferred quickly between the computer and the peripherals
  - It avoids bothering the CPU
- The DMA controller is connected to the main memory and the fast peripheral device (with high-bandwidth channel), and can be controlled by the CPU
- The data transfer is initiated by the CPU, but the number of interrupts is significantly less
- The DMA Controller consist of:
  - Data Counter (amount of transferred data)
  - Address Register
  - Data Register
  - DMA Controller
- **Operation**:
- **Preparation**:
  - Transfer the information needed for data transfer from the CPU to the DMA Controller
  - We put the number of transferable data units into the Data Counter
  - We put the to-be memory address of the data into the Address Register
- **Block transfer**: for high volume data

    - DMA request by the DMA Controller (for using system bus)
    - DMA acknowledge by the CPU
    - * DMA Controller writes the first transferable data from into the Data Register
    - DMA Controller writes the data through the bus into the address specified in Address Register
    - DMA decrements Data Counter and increments Address Register
    - if DC is not null then back to *, if null send interrupt about block transfer done
- **Cycle stealing**: for fast, small data
    - The DMA and the CPU uses the system bus alternately
    - The DMA Controller can use the system bus in its own cycles, between the cycles of the CPU
    - Fetch -> decode -> fetch operand -> execute -> write back to memory

I/O Channel

- Expansion of the DMA concept for slower peripherals
- The channel fetches instruction from the main memory (no dedicated memory)
    - Then executes them
- The CPU still initiates the channel's operations, but the handling of different peripherals is aggregated into a single channel controller
- Selector Channel:
    - For aggregating faster peripherals
    - Only one can be active at a time
- Multiplexer:
    - For aggregating slower peripherals
    - Multiple can be active at a time

2nd version / interpretation:

I/O Channels

- I/O modules became more and more complex
- It is a dedicated processor with specialized Instruction Set which can execute programs from the main memory
    - They can handle complex I/O situations without the CPU
    - The CPU still initiates an I/O operation but it is handed over to the channel
- They can increase performance
- I/O Processors: I/O Channels with dedicated memory

Classification of computer architectures

(Flynn's classification, modern classification methods)

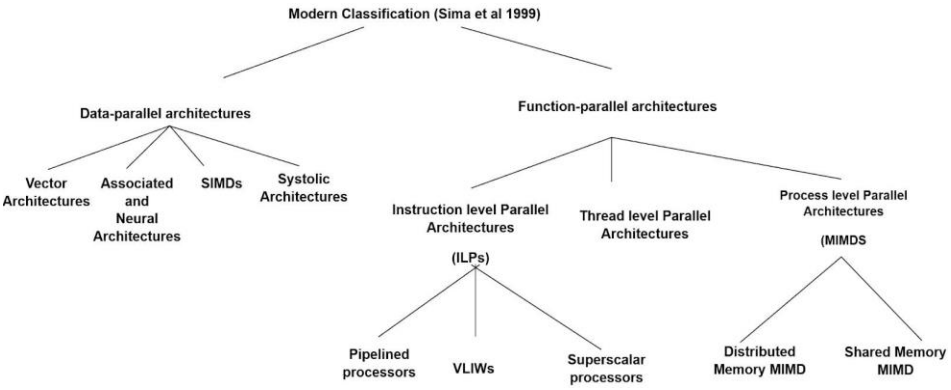
Flynn's classification:

- The "classic" taxonomy 1966
- Based on the number of control units as well as the number of processors available in a computer
- Definitions:
- SI: Single Instruction Stream:
  - A machine has a single control unit, producing a single stream of instruction
- MI: Multiple Instruction Stream:
  - A machine has multiple control units, each producing a distinct stream of instruction
- SD: Single Data Stream:
  - A single processor is available which processes a single stream of data
- MD: Multiple Data Stream:
  - Multiple processors are available each processing a distinct stream of data
- Based on these ions Flynn categorized computers by combining the possible instruction streams and processing instruction streams and processing aspects

	Single Data Stream	Multiple Data Stream
Single Instruction Stream	SISD	SIMD
Multiple Instruction Stream	~~MISD~~	MIMD

Parts:

- SISD: Traditional Sequential Processing
- SIMD: Multimedia processing
- MISD: Theoretical processing, does not make sense
- MIMD: Parallel processing (SMP, NUMA)
- Straightforward scheme, it does not cover key aspects such as:
  - what kind of parallelization is utilized
  - what level of parallel execution is implemented
  - how parallel execution is implemented



Data Parallel Architectures:

- Vector Architectures:
  - Vector or Array processor is the CPU that operates on one-dimensional arrays
  - Great performance improvements
- Associative and Neural Architectures:
  - Arose naturally out of one of the human method of problem solving
  - Neural computers are operated in a way which is completely different from the operation of normal computers
- SIMD Architectures:
  - Multiple processing elements that perform the same operation on multiple data points simultaneously
  - Applicable to common tasks in connection with image, video and audio manipulation
- Systolic Architectures:
  - SIMD is general purpose, less optimally efficient for any specific purpose
  - Systolic array is a homogeneous network of tightly coupled data processing units

Functional-parallel architectures:

- Instruction level Parallel Architecture:
  - Pipelined processor:
    - Temporal overlapping of processing
    - Prone to dependency problems
  - VLIW processor:
    - VLIW processor depends on the program to decide which instruction to execute simultaneously and how to resolve conflicts
  - Superscalar processor:
    - Executes more than one instruction during a clock cycle
- Thread level parallel architecture
- Process level Parallel architectures:
  - Distributed Memory MIMDs:
    - Each processor has its own private memory
    - Computational tasks can only operate on local task, for remote communication between processors is needed
  - Shared Memory MIMDs:
    - Uses memory that may be simultaneously accessed by multiple programs with an intent to provide communication among them or avoid redundant copies.

Data dependencies

w. definition,

x. Types,

y. influence of computational performance

Data dependency: when subsequent instructions are dependent on each other because of data

Data dependency:

A data dependency in computer science is a situation in which a program statement (instruction) refers to the data of a preceding statement. In compiler theory, the technique used to discover data dependencies among statements (or instructions) is called dependence analysis.

Flow dependency, RAW:

- A Flow dependency, also known as a data dependency or true dependency or read-after-write (RAW), occurs when an instruction depends on the result of a previous instruction.
- RAW dependency is true dependency, because it cannot be abandoned

```
i1: load r1, x          r1 ← x
i2: add r2, r2, r1      r2 ← r2 + r1
```

- RAW dependencies can be broken down into
  - load-use and
  - define-use dependencies
  - Load-use dependency means that the requested source operand has to be loaded first
- By contrast in case of define-use dependency the requested source has to be defined in the preceding instructions

```
i1: mul r1, r4, r5      r1 ← r4 × r5
i2: add r2, r2, r1      r2 ← r2 + r1
```

- (Instruction level parallelism is therefore not an option in this example.)

Anti-dependency, WAR:

- An anti-dependency, also known as write-after-read (WAR), occurs when an instruction requires a value that is later updated.

```
i1: mul r1, r2, r5      r1 ← r2 × r5
i2: add r2, r6, r3      r2 ← r6 + r3
```

- 1 uses r2 as a source operand and i2 uses r2 as a destination operand
  - i1 cannot be correctly executed if i2 update (rewrite) r2 earlier than i1 use it in the calculation
  - An anti-dependency is an example of a name dependency. WAR dependency is false dependency and can be eliminated by register renaming

- A new variable, B2, has been declared as a copy of B in a new instruction, instruction N. The anti-dependency between 2 and 3 has been removed, meaning that these instructions may now be executed in parallel. However, the modification has introduced a new dependency: instruction 2 is now truly dependent on instruction N, which is truly dependent upon instruction 1. As flow dependencies, these new dependencies are impossible to safely remove.

Output dependency, WAW:

- An output dependency, also known as write-after-write (WAW), occurs when the ordering of instructions will affect the final output value of a variable.

```
i1: mul r1, r2, r5        r1 ← r2 × r5
i2: add r1, r2, r3        r1 ← r2 + r3
```

- Both i1 and i2 use r1 as a destination operand
    - i1 cannot be correctly executed if i2 update (rewrite) r2 earlier than i1 use it in the calculation
    - WAW dependency is false dependency and can be eliminated by register renaming. As with anti-dependencies, output dependencies are name dependencies. That is, they may be removed through
    - renaming of variables, as in the below modification of the above example :

```
1. B2 = 3
2. A = B2 + 1
3. B = 7
```

Data dependencies in loops:

- The instructions in the loop body may have RAW, WAR or WAW dependencies among themselves
- Instructions belonging to a particular iteration may also be dependent on instructions belonging to previous loop iterations
    - We call it 'recurrence'
    - Recurrences are references in loop bodies to values that have been captured in previous iterations

```
for I = 2 to n
      X(I) = A×X(I-1) + B
```

    - In general a recurrence is kth order, if the associated computation needs results computer in the precious k iterations
    - A most common form of recurrences is first order linear recurrences. General form of first order linear recurrence

Data dependency graphs (DDGs):

- Data dependency graphs are directed acyclic graphs
- We can use them to represent data dependencies
    - The nodes of the graphs symbolizes instructions
    - The directed arc between two nodes show precedence dependency between nodes (instructions)
- If we restrict ourselves to straight-line code, the directed graph is acyclic

Control dependencies and possibilities to cope with them

z. definition,

When a conditional branch is met, the subsequent execution path depends on the outcome of the condition

In case of unconditional branches, we are calling instructions for nothing, as we surely won't continue the execution there. With pipelining, we execute n-1 instructions (n=pipeline levels) => jump bubble

aa. influence of computational performance,

Studies show that general-purpose programs behave quite differently than scientific/technical programs. Running a general-purpose code (e.g. operating system code, non-numerical applications) we can expect every 3rd-6th instruction on average to be a conditional branch instruction. While executing scientific programs, every 10th-20th instruction on average is likely to be a conditional branch. Frequent conditional branches impose a heavy performance constraint on ILP processors

bb. methods to cope with them,

ILP processors can boost performance mainly by executing more and more instructions in parallel. To achieve this the processor must incorporate more and more Execution Units and if forced to raise the instruction issue rate. The more instructions issued each cycle, the higher the probability of encountering a control dependency in each cycle. Control dependencies are the major obstacles to increase the performance of ILP-processors

For **unconditional branches:**

Static: we fill the jump bubble with NOP instructions.

Dynamic: change sequence of instructions, put data manipulating instructions after JMP

For **conditional branches:**

Branch prediction, speculative execution

cc. static and dynamic branch prediction,

Static Branch Prediction in general is a prediction that uses information that was gathered before the execution of the program.

Dynamic Branch Prediction on the other hand uses information about taken or not taken branches gathered at run-time to predict the outcome of a branch. There are several dynamic branch predictor in use or being researched nowadays. Those include One-Level Branch Predictors, Two-Level Branch Predictors and Multiple Component Branch Predictors.

dd. speculative execution

Speculative execution is an optimization technique where a computer system performs some task that may not be needed. Work is done before it is known whether it is actually needed, so as to prevent a delay that would have to be incurred by doing the work after it is known that it is needed. If it turns out the work was not needed after all, most changes made by the work are reverted and the results are ignored.

Sequential consistency

ee. sequential consistency of instruction execution and interrupts/exception,

ff. reordering buffer

gg. reservation stations

Parallel Execution

- During parallel instruction execution the instructions are executed out-of-order
  - Due to unequal execution times even in-order issuing can yield out-of-order operation
- Three important terms:
  - Finished instruction: the required operation is accomplished but the result has not yet been written back to memory or register, status bits haven't updated
  - Completed instruction: the result is written back to register or memory, status bits are updated
  - Retired instruction: results are correct and visible in original order

Aspects of Consistency

- Processor Consistency: the order in which instructions are completed
  - Whether superscalar processor executes in same order as a sequential processor
  - Weak: instructions may complete out-of-order
    - Might be reordered by the CPU if no dependencies are violated
    - Data dependencies have to be detected and resolved during superscalar execution
  - Strong: instructions are forced to complete in strict program order
    - Usually achieved by using a ReOrder Buffer (ROB)
- Memory Consistency: the order in which memory is accessed (due to load and store instructions)
  - Whether superscalar processor accesses memory in same order as a sequential processor
  - Weak: if memory accesses are out-of-order
    - Allows Load / Store reordering
    - Helps to increase processor performance
  - Strong: memory accesses occur strictly in program order
    - Forbids any Load / Store reordering
- Load / Store Reordering:
  - Load / Store Bypassing
    - Load can bypass a pending store operation or vice versa
    - If no preceding stores have the same target address as the load
  - Speculative Loads
    - The load is executed before every address is known
    - In case of collision the load is repeated
  - Hide Cache Misses
    - With Load / Store reordering

Types of Sequential Consistency

- Strong: the sequence of operations is the same during sequential execution
- Weak: the execution sequence can differ from sequential execution but does not damage the logic of the sequential operation

Sequential Consistency of Instruction Execution

DIV R3, R2, R1
ADD R6, R4, R5
JZ label

- Sequential Execution: first the DIV then the ADD executes, the jump is compared to the ADD's result
- Parallel Execution: It can happen that the DIV operation finishes later and the JZ instruction is executed according to DIV's result
    - The jump instruction must wait for both to finish

Sequential Consistency of Exceptions

DIV R3, R2, R1
ADD R6, R4, R5
JZ label

- If the interrupt is registered by the CPU immediately, the execution order might change
- If the ADD instruction results in overflow, the first instruction is cancelled (?)
- Strong (precise): interrupts can be "accepted" by the CPU only in the order of original execution
    - The state of the processor remains the same
- Weak (imprecise): the processor is unable to reconstruct the state by itself

The Reorder Buffer (ROB)

- First described in 1988
- It was designed to solve the precise interrupt problem but is suitable for assuring sequential consistency in the case of multiple Executional Units
- Circular buffer with two pointers
    - The Head pointer indicates the location of the next free entry
    - The Tail pointer marks the retired instructions
    - Instructions are written into the buffer in strict program order
- Every entry (address and instruction) has a status:
    - I: issued
    - X: execution
    - F: finished
- An instruction can retire only if it has finished and all previous instructions are retired

Reservation Stations

Inst	Operands	Is	Exec	Wr	Commit	Comments
DIV	R2, R3, R4	1	2	42	43	
MUL	R1, R5, R6	2	3	13	44	
ADD	R3, R7, R8	3	4	5	45	
MUL	R1, R1, R3	14	15	25	46	NEED RS → ISSUE
SUB	R4, R1, R5	15	26	27	47	EXE DEP ON R1
ADD	R1, R4, R2	16	43	44	48	EXE DEP ON R2

- Approach invented by Robert Tomasulo
- The scheme tracks when operands for instructions are available and introduces register renaming
    - RAW hazards are avoided by executing an instruction only when its operands are available
    - WAR and WAW hazards are eliminated by register renaming
- They buffer operands of instructions waiting to be issued
- How it works:
    - Instruction goes through 3 steps
        - Each can take an arbitrary number of clock cycles

- **Issue**: get the next instruction from the head of the instruction queue (FIFO)
  - If there is an empty matching reservation station issue the instruction with operands
  - If none is available, then stall the issue until there is one
  - If operands are not in registers, keep track of the units that produce the operand
- **Execute**: when the operands are available execute instruction at the corresponding functional unit
  - RAW hazards are avoided
  - No instruction can be executed until all branches preceding the instruction are completed
- **Write**: when available, write the result on the data bus and from there into the registers + any reservation stations waiting for this result

- In Tomasulo's scheme, results are broadcasted on a Data Bus (CDB) that is monitored by the reservation station
- Each reservation station has 7 fields:
  - Qp - Operation to perform on source operands
  - Qj, Qk - 2 reservation stations that will produce the source operands
  - Zero means the value is available in Vj or Vk
  - Vj, Vk - 2 values of source operands
  - A - An address field for L / S instructions
  - Busy - A busy field meaning the station is occupied
- The register file has a Qi field which shows the reservation station that contains the operation whose result should be stored into this register

		Instruction status		
Instruction		Issue	Execute	Write result
L.D	F6,32(R2)	√	√	√
L.D	F2,44(R3)	√	√	
MUL.D	F0,F2,F4	√		
SUB.D	F8,F2,F6	√		
DIV.D	F10,F0,F6	√		
ADD.D	F6,F8,F2	√		

	Reservation stations						
Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	No						
Load2	Yes	Load					44 + Regs[R3]
Add1	Yes	SUB		Mem[32 + Regs[R2]]	Load2		
Add2	Yes	ADD			Add1	Load2	
Add3	No						
Mult1	Yes	MUL		Regs[F4]	Load2		
Mult2	Yes	DIV		Mem[32 + Regs[R2]]	Mult1		

	Register status								
Field	F0	F2	F4	F6	F8	F10	F12	...	F30
Qi	Mult1	Load2		Add2	Add1	Mult2			

Most important methods of parallel execution of instructions

These techniques are used for pipelining. The execution of an instruction can be divided into stages:

Fetch: fetch instruction
Decode/Source op: decode instruction, get source operands
Execute: execute instruction
Write back: write result

hh. prefetching,

Prefetching occurs when a processor requests an instruction from main memory before it is actually needed. Once the instruction comes back from memory, it is placed in a cache. When an instruction is actually needed, the instruction can be accessed much more quickly from the cache than if it had to make a request from memory. Since programs are generally executed sequentially, performance is likely to be best when instructions are prefetched in program order.

ii. overlapped instruction execution,

To keep the pipeline full, parallelism among instructions must be exploited by finding sequences of unrelated instructions that can be overlapped in the pipeline (e.g. floating point mantissa and exponent calculation)

jj. pipelining, potential bottlenecks and methods of avoiding them

Pipelining is an implementation technique where multiple instructions are overlapped in execution. The computer pipeline is divided in stages. Each stage completes a part of an instruction in parallel. The stages are connected one to the next to form a pipe - instructions enter at one end, progress through the stages, and exit at the other end. Pipelining does not decrease the time for individual instruction execution. Instead, it increases instruction throughput. The throughput of the instruction pipeline is determined by how often an instruction exits the pipeline.

Memory is a bottleneck in High-Speed computers. Memory cycles are much slower than processor cycles. With pipelining, memory access could happen in every cycle, instead of once / instruction cycle. The memory cannot keep up with the speed of the CPU. Solution: introduction of caches, 1980s

Branch prediction can also be a bottleneck. With, unconditional branches, the pipeline will be delayed with the jump bubble (n-1). With conditional branches, the evaluation and jump destination address calculation take time. Methods: RISC delayed branch (instruction interchange to avoid wasting clock cycles), early CISC hardware (evaluation and address calculation happens during decoding), fix branch prediction (always jump)

Dependency stalls: We can use bypassing or forwarding to prevent stalls due to RAW. Results are immediately forwarded to be used as input for the next pipeline cycle

The pipeline-based instruction execution

the base of pipeline;

Pipelining is a technique where multiple instructions are overlapped during execution. Pipeline is divided into stages and these stages are connected with one another to form a pipe like structure. Instructions enter from one end and exit from another end.

Requirements: time requirement of stages should uniform, stages should be independent execution units, the output of one stage should be the input of the next

Pipelining increases the overall instruction throughput.

characteristics;

Number of stages: The number of stages will determine the number of parallelly executed instructions. This will always be less in practice thanks to stalls, dependencies. The deeper the pipeline, the more performance gain, up to a given limit. Dependencies occur more frequently.

Subtask specification: naming the stage activity (fetch,decode..) or specific activity for different instruction types (fx,fp..)

Layout: Sequential, cycled

Operand forwarding / Bypass: We can use bypassing or forwarding to prevent stalls due to RAW. Results are immediately forwarded to be used as input for the next pipeline cycle

Timing:

- **Synchronous:** nowadays this is used, synchronized execution based on clock signal
- **Asynchronous:** continuous execution, end of operation signals start of the next one

Logical and physical pipelines

Logical Pipelines

- Has 2 levels:
    - 1st level: functional design (determine the main stages, such as Fetch Instruction, Execute etc.)
    - 2nd level: elementary design (microinstructions to be executed within stages)

Physical Pipelines

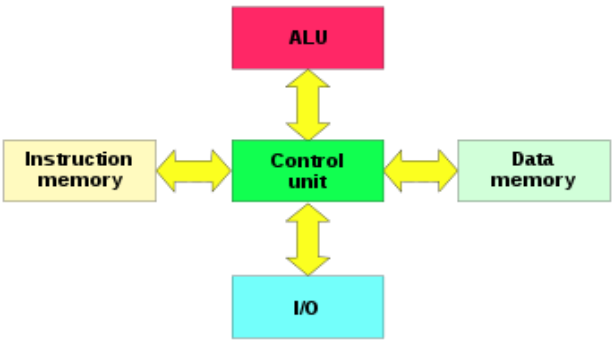
- Pipeline stages are independent Execution Units
- Separator Registers (hidden to programmers) help to separate the EUs
- Superscalar: introduced Multiplexers beside Separator Registers for multiple Eus
- Universal pipeline: 1 physical pipeline, multiple logical, slow.
- Master pipeline: Master pipeline for any instruction, other pipeline only for simple
- Dedicated pipeline: Every logical pipeline on separate physical pipeline

Superscalar architectures

kk. principles of operation,

A superscalar processor can execute more than one instruction during a clock cycle by simultaneously dispatching multiple instructions to different execution units on the processor. Using Instruction Window. Each execution unit is not a separate processor (or a core if the processor is a multi-core processor), but an execution resource within a single CPU such as an arithmetic logic unit

ll. Harvard architecture,



- Parallel buses, performance increase
- Ideal of parallel execution
- Separate data and instruction storage and address space
- Used today in a modified manner (L1)
- Length may be different

mm. CISC-RISC comparison,

Reduced Instruction Set Computer	Complex Instruction Set Computer
50-150 instruction	Hundreds of instructions
Hardwired control unit	Microprogrammed control unit
Every instruction uses registers	Instructions might use register + memory
Fix instruction length	Variable instruction length
Complex compiler	Simple compiler
Very fast	Slower
Simple decoding	Complex encoding
Easy pipelining	Difficult pipelining (memory access)
No use of external memory for calculations	Using external memory for calculations
Hard code expansion	Easy code expansion
Typically 3 operands / instruction	Typically 1-2 operands / instruction
Lot of general purpose registers	One instruction max complete multiple elemental task
Uses more registers	Direct memory access
1 Instruction per clock cycle	1 Instruction is usually more than 1 clock cycle

nn. first and second generation of superscalar processors

First Generation Superscalars

- **Direct instruction issuing**: instructions are immediately forwarded to execution units after decoding
- **Static branch prediction:** based on code, after fetch
- **2-level cache**:
  - L1: on the CPU chip
    - Data cache
    - Instruction cache
  - L2: on separate chip
- **2 or 3 Execution Units**
- **IPC:** 1 because of dependencies

Second Gen. Superscalars

- **Dynamic instruction scheduling**: using built-in delay stations
  - These stations are placed between decoders and EUs
  - Buffered instruction scheduling
  - Can schedule out-of-order
- **Instruction Window**:
  - Buffer that contains the instructions to be issued in the current cycle
  - Decoding and dependency check
- **Register Renaming**:
  - To remove false data dependencies
- **Multiple Execution Units**:
  - Deal with resource dependencies
- **True Data dependencies stall:** but dynamic branch prediction partially handles it
- **Dynamic branch prediction:** speculative prediction with 90-95% accuracy
- **Sophisticated, extended cache and branch subsystems**
- **IPC:** 4 for RISC, 3 for CISC

Instruction fetch and branch prediction

oo. the concept of the local branch prediction,

In computer architecture, a branch predictor is a digital circuit that tries to guess which way a branch (e.g., an if–then–else structure) will go before this is known definitively. The purpose of the branch predictor is to improve the flow in the instruction pipeline. Branch predictors play a critical role in achieving high effective performance in many modern pipelined microprocessor architectures such as x86.

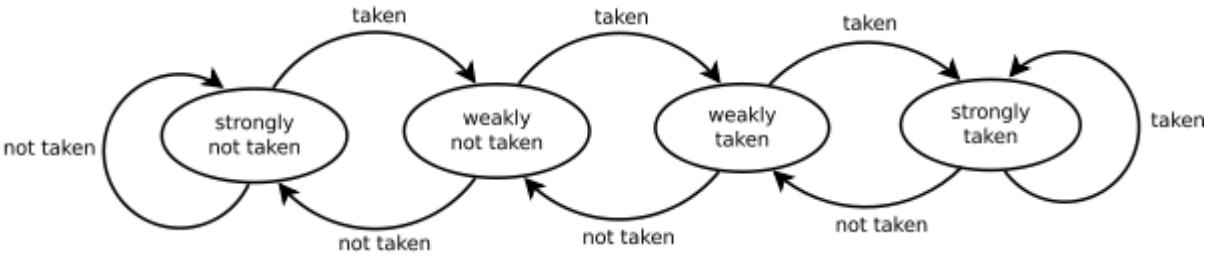
The time that is wasted in case of a branch misprediction is equal to the number of stages in the pipeline from the fetch stage to the execute stage. Modern microprocessors tend to have quite long pipelines so that the misprediction delay is between 10 and 20 clock cycles. As a result, making a pipeline longer increases the need for a more advanced branch predictor.

pp. the local one-level and two-level dynamic branch prediction,

Saturating counter

A 1-bit saturating counter (essentially a flip-flop) records the last outcome of the branch. This is the simplest version of dynamic branch predictor possible, although it is not very accurate.

A 2-bit saturating counter is a state machine with four states:



Two-level predictor

The Two-Level Branch Predictor, also referred to as Correlation-Based Branch Predictor, uses a two-dimensional table of counters, also called "Pattern History Table". The table entries are two-bit counters.

Two-level adaptive predictor

If an if statement is executed three times, the decision made on the third execution might depend upon whether the previous two were taken or not. In such scenarios, a two-level adaptive predictor works more efficiently than a saturation counter. Conditional jumps that are taken every second time or have some other regularly recurring pattern are not predicted well by the saturating counter. A two-level adaptive predictor remembers the history of the last n occurrences of the branch and uses one saturating counter for each of the possible 2n history patterns.

qq. branch target prediction

Branch prediction is not the same as branch target prediction. Branch prediction attempts to guess whether a conditional jump will be taken or not. Branch target prediction attempts to guess the target of a taken conditional or unconditional jump before it is computed by decoding and executing the instruction itself. Branch prediction and branch target prediction are often combined into the same circuitry.

Third generation superscalar processors: the parallel execution inside the instruction

- **Third generation superscalar processors**
  - Parallel execution within instructions
  - SIMD (multimedia)
  - Higher memory bandwidth
  - L2 cache integrated
  - AGP (new bus for graphics)

rr. three-operand instructions,

- **Definition:** two different operations within one instruction
- **For example:** multiply-add: x=a*b+c, load-op: load and execute
- Speed up arithmetic operations, mostly used in RISC architecture

ss. SIMD instructions,

- Its introduction brought the third generation of superscalar
- **Definition:** Within a single instruction, we perform the same operation on multiple operands
- **Fix point multimedia:** sound and pixel based image, 2-8 operands in one instruction
- **Floating point multimedia:** vector based image, 2-4 operands in one instruction

It describes computers with multiple processing elements that perform the same operation on multiple data points simultaneously. Such machines exploit data level parallelism, but not concurrency: there are simultaneous (parallel) computations, but only a single process (instruction) at a given moment. SIMD is particularly applicable to common tasks such as adjusting the contrast in a digital image or adjusting the volume of digital audio. Most modern CPU designs include SIMD instructions to improve the performance of multimedia use.

With a SIMD processor there are two improvements to this process. For one the data is understood to be in blocks, and a number of values can be loaded all at once. Instead of a series of instructions saying "retrieve this pixel, now retrieve the next pixel", a SIMD processor will have a single instruction that effectively says "retrieve n pixels" (where n is a number that varies from design to design). For a variety of reasons, this can take much less time than retrieving each pixel individually, as with a traditional CPU design.

Another advantage is that the instruction operates on all loaded data in a single operation. In other words, if the SIMD system works by loading up eight data points at once, the add operation being applied to the data will happen to all eight values at the same time.

tt. vector multimedia execution,

Vector architectures

- place data into large, sequential register files
- operate on data in those register files
- a single instruction operates on vectors of data
- **Vector Registers**: fixed-length bank holding a single vector, needs enough ports to feed all vector functional units
- **Vector Functional Units:** Each unit is fully pipelined, and can start new operation every cycle
  - **Control Unit** is needed to detect hazards
- **Vector Load/Store Unit:** loads and stores vectors from/to memory, pipelined -> one word/cycle
- **Scalar Registers:** provide data as input to the functional unit, or used for computing addresses

uu. VLIW-architectures

- **Instruction Level Parallelism**
  - Revolution -> new architecture
  - Last possibility (thread and process parallelism remains)
- **Space and Time Parallelism**
- **Dependencies handled by compiler (static dependency handling)**
- **New Instruction format:** one very long instruction, containing multiple independent instructions

Very-Long Instruction Word (VLIW) architectures are a suitable alternative for exploiting instruction-level parallelism (ILP) in programs, that is, for executing more than one basic (primitive) instruction at a time. It is a concatenation of several short instructions and requires multiple execution units running in parallel, to carry out the instructions in a single cycle.

Advantages of VLIW architecture:

- Simpler structure than superscalar
- Faster than superscalar
- Increased performance.
- Potentially scalable i.e. more execution units can be added and so more instructions can be packed into the VLIW instruction.

Disadvantages of VLIW architecture:

- Static scheduling
- Very complex compiler needed
- New physical and instruction set architecture
- High costs, requires lot of time
- Not compatible with previous architectures, like x86

Caches

(the main reasons of appearance, structure, development, types according to storing data, characteristics, organizational alternatives, type of retrieval, more level caches, optimal sizes and reasons, the role of tags, inclusive- and exclusive cache, cache types)

In computing, a cache is a hardware or software component that stores data so that future requests for that data can be served faster; the data stored in a cache might be the result of an earlier computation or a copy of data stored elsewhere. A cache hit occurs when the requested data can be found in a cache, while a cache miss occurs when it cannot. Cache hits are served by reading data from the cache, which is faster than recomputing a result or reading from a slower data store; thus, the more requests that can be served from the cache, the faster the system performs.

To be cost-effective and to enable efficient use of data, caches must be relatively small. Nevertheless, caches have proven themselves in many areas of computing, because typical computer applications access data with a high degree of locality of reference. Such access patterns exhibit temporal locality, where data is requested that has been recently requested already, and spatial locality, where data is requested that is stored physically close to data that has already been requested.

- We want bigger, faster, higher hit rate => contradiction => cache levels
- **Size:** 32KB-30MB
- **Placement:** on chip (nowadays), off chip
- **Data transfer:** happens in blocks
- **Stores**
    - Data
    - Tags
    - Status bits
        - V: data is valid if it corresponds to the data in memory at the given address
        - D: data is “dirty” if it was modified and not yet saved into memory
- **Retrieval:**
    - **Content based:** Content Address Memory
    - **Address based:** With Memory Management Unit, Physical, Virtual, or Tag

Addressing

- Almost all non-embedded processors (and many of embedded ones) supports virtual memory
    - When virtual memory is used addresses in machine instructions contain virtual addresses
        - The MMU has to translate virtual addresses to physical ones
- The designer can place cache between Processor and MMU or between MMU and memory
    - In the first case the cache uses virtual addresses
    - In the second case the cache uses physical ones
- The advantage of using logical addresses is that the cache access is faster because it can omit address translation
    - But every process has own address space → after a process switch the content of the cache becomes obsolete (and must be flushed)
- The selection between addressing modes is complex, and needs a lot of design work

Sizing

- The bigger the cache is, the higher the footprint on the chip (which is limited)
- The bigger the cache, the higher the cost of it
- The bigger the cache, the smaller the probability of cache miss (and the need of costly main memory access)

There is no universal solution, there are a lot of tradeoffs when designing

Mapping function

- Because there are fewer cache lines than main memory blocks, we need an algorithm for mapping main memory blocks into cache lines
- The choice of mapping function defines the organization of the cache
- There are three techniques used
- **Fully Associative**
    - A block can be placed in any line, determined by the replacement algorithm
    - N comparing circuits (N = # of lines)
    - CPU checks every line for match in parallel
    - Very rare
    - Fast, flexible, high hit rate
    - Lot of comparing circuits, expensive
- **Direct Mapped**
    - A block has a dedicated line it can be placed (determined with block number modulo N)
    - 1 comparing circuit
    - Fast, simple, cheap
    - Rigid, low hit rate
- **Set Associative**
    - N way associative: N lines in a set
    - N comparing circuit
    - Search based on set index
    - Used today
    - Fast, flexible, few comparing circuits

Replacement algorithms

- Once a cache has been filled, we have to drop the contents of a line if we want to load a new one
- In case of direct mapping there is only one possible line for every address – we don't need complex algorithms
- In case of fully associative and set associative mappings we can select between lines – so we need an algorithm to do this
- Most common algorithms:
    - Last Recently Used (LRU) – replaces the block in the cache longest with no reference to it (it is the most effective algorithm)
    - First-In-First-Out (FIFO) – replaces the block in the set loaded at the earliest
    - Least Frequently Used (LFU) – replaces the block in the cache that experienced the fewest references
    - Random algorithm (choose a line without any usage information)

Write policy

- When a block that is resident in the cache is to be replaced, there are two cases to consider
    - If the old block in the cache has not been altered, it may been overwritten
    - If it is altered the main memory must be updated before the replacement
- A variety of write policies is possible
    - The '**write through'** algorithm updates cache and main memory at the same time, ensuring that the content of the main memory is always valid. It generates substantial memory traffic, and can lead to bottlenecks
    - The '**write back'** separates the update of cache and the update of memory (we update memory only during page replacement) and lower the memory traffic
        - With this solution we need to keep track the modification of cache data ('dirty bit')

- With multiprocessor systems the cache coherency is a serious design issue

Line size

- The line size defines the size of a cache block - this is the amount of data moved in once
  - Due to the principle of locality, the data that is next to the accessed data is likely to be referenced
- There are different consequences of line size (the larger block, is not always the better)
  - there are fewer blocks in the cache
  - the principle of locality has limited extent

Number of caches

- There are two dimensions
  - Levels of caches
  - Using Unified vs Split caches
- Multilevel caches
  - The on-chip cache is far the fastest cache improving the processor's external bus activity – but its size is very limited
  - Using bigger and bigger (and slower and slower) second and third level caches helps us to minimize the waste of CPU time due to memory stalls
- Unified vs split caches

Organization: exclusive, inclusive, both for multicore

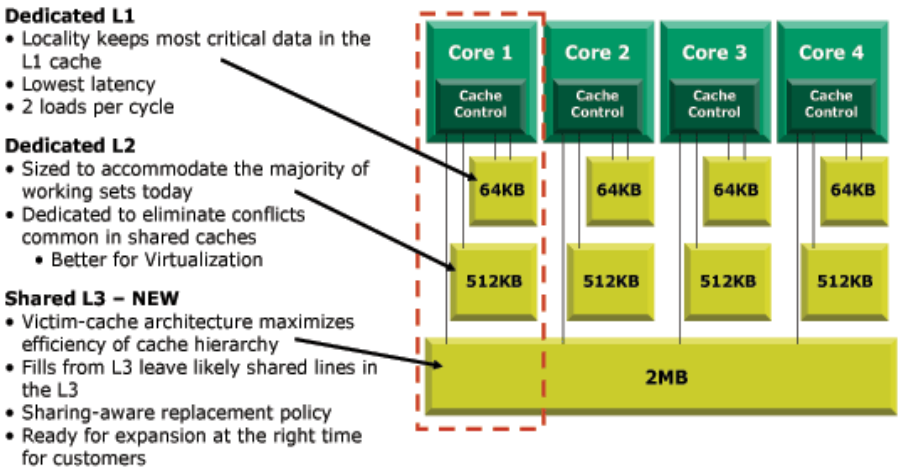
Tags

A cache is made up of a pool of entries. Each entry has associated data, which is a copy of the same data in some backing store. Each entry also has a tag, which is a part of the memory address based on which the starting address of the memory block can be found. Identifies which block is stored.

Cache inclusion policy

Multi-level caches can be designed in various ways depending on whether the content of one cache is present in other level of caches. If all blocks in the higher level cache are also present in the lower level cache, then the lower level cache is said to be inclusive of the higher level cache. If the lower level cache contains only blocks that are not present in the higher level cache, then the lower level cache is said to be exclusive of the higher level cache. If the contents of the lower level cache are neither strictly inclusive nor exclusive of the higher level cache, then it is called non-inclusive non-exclusive (NINE) cache.

Cache types



Development limits

(types, the main reasons of appearance, handling, the technological limits of superscalar processors: the consequences of increasing the clock frequency, TDP, heat problems)

Available performance improvement from superscalar techniques is limited by three key areas:

- The degree of intrinsic parallelism in the instruction stream (instructions requiring the same computational resources from the CPU).
- The complexity and time cost of dependency checking logic and register renaming circuitry
- The branch instruction processing.

Existing binary executable programs have varying degrees of intrinsic parallelism. In some cases, instructions are not dependent on each other and can be executed simultaneously. In other cases they are interdependent: one instruction impacts either resources or results of the other.

When the number of simultaneously issued instructions increases, the cost of dependency checking increases extremely rapidly. This is exacerbated by the need to check dependencies at run time and at the CPU's clock rate. This cost includes additional logic gates required to implement the checks, and time delays through those gates.

Even though the instruction stream may contain no inter-instruction dependencies, a superscalar CPU must nonetheless check for that possibility, since there is no assurance otherwise and failure to detect a dependency would produce incorrect results.

No matter how advanced the semiconductor process or how fast the switching speed is, this places a practical limit on how many instructions can be simultaneously dispatched. While process advances will allow ever greater numbers of execution units (e.g., ALUs), the burden of checking instruction dependencies grows rapidly, as does the complexity of register renaming circuitry to mitigate some dependencies. Collectively the power consumption, complexity and gate delay costs limit the achievable superscalar speedup to roughly eight simultaneously dispatched instructions.

However even given infinitely fast dependency checking logic on an otherwise conventional superscalar CPU, if the instruction stream itself has many dependencies, this would also limit the possible speedup. Thus the degree of intrinsic parallelism in the code stream forms a second limitation.

Alternatives

Collectively, these limits drive investigation into alternative architectural changes such as very long instruction word (VLIW), explicitly parallel instruction computing (EPIC), simultaneous multithreading (SMT), and multi-core computing.

With VLIW, the burdensome task of dependency checking by hardware logic at run time is removed and delegated to the compiler. Explicitly parallel instruction computing (EPIC) is like VLIW with extra cache prefetching instructions.

Simultaneous multithreading (SMT) is a technique for improving the overall efficiency of superscalar processors. SMT permits multiple independent threads of execution to better utilize the resources provided by modern processor architectures.

Other limitations

The transistor count of a processor is the number of transistors that the processor is equipped with. Since the CPUs stay roughly the same size, the transistor count is directly related to the size of the transistors.

So, by going from the 170 million transistors that an Intel® Pentium® 4 processor from 2004 was equipped with to the 4.3 billion transistors of a 15-core Intel® Xeon® Ivy Bridge processor from 2013, we see that the sizes of the transistors have shrunk enormously. This increase was observed by Moore and described by his law that states that the integration density of transistors doubles every 18 to 24 months.

Furthermore, thermal losses occur when you are putting several billions of transistors together on a small area and switching them on and off again several billion times per second. The faster we switch the transistors on and off, the more heat will be generated. Without proper cooling, they might fail and be destroyed. One implication of this is that a lower operating clock speed will generate less heat and ensure the longevity of the processor. Another severe drawback is that an increase in clock speed implies a voltage increase and there is a cubic dependency between this and the power consumption.

But how can we get more computing power out of more transistors without increasing the clock speed? Through the application of multicore computing. The overwhelming benefit of multicores can be derived from the following reasoning: When cutting down the clock speed by 30%, the power is reduced to 35% of its original consumption, due to the cubic dependency (0.7*0.7*0.7 ~ 0.35).

Yet, computing performance is also reduced by 30%. But when operating two compute cores running with 70% of the original clock speed, we have 140% of the original compute power using only 70% of the original power consumption (2 x 35%). Of course, to reach this type of efficiency, you would have to program the parallelization of the process code to perfectly exploit both cores operating at the same time.

Thread level parallel architectures

(problems of the single-core processors, performance increase vs. complexity, forms of parallelism, forms of multithreading, differentiates from single threaded processors, SMT implementation)

Reason for Switching to Multiprocessor Systems

- After 2005 the only way to increase performance was multiprocessing
- ILP started reaching limitations
- Growing interest in high-end servers and cloud-computing
- Growth in data-intensive applications

Threads

The smallest individually executable part of a program

Realization Possibilities:

- Low-level realization:
  - Parallel architectures
  - Appropriate compilers
- High-level realization:
  - Multi-threaded Operating Systems (concurrent instruction execution)

Utilization Possibilities:

- Implicit: trying to find and manage parallel execution without the programmer
  - With compiler or extra hardware
- Explicit: programmer writes code capable of parallel execution

Threads can be:

- From separate applications (multiprogramming)
- From same application (multithreading / multitasking)

Multithreading types:

- Software-level: running multithread programs (or OSes) on one CPU on one thread (scheduled in time)
- Hardware-level: running multithread programs (or OSes) on one CPU but on more threads

SMP (Symmetric MultiProcessing): there are two (or more) cores with shared L2 or L3 cache through which individual instructions go

SMT (Simultaneous MultiThreading): there is one core connected to an L2 or L3 cache processing two independent instructions

How does the system change among threads:

- Fine-grained: the system changes thread in every clock cycle
  - In case of dependencies the system can take advantage of unused cycles (?)
- Coarse-grained: event changes threads (Switch On Event MT, SOEMT)
  - 1-2 wasted cycle
- SMT: multiple threads are running at once
  - Needs more EUs per cores
  - As many PCs as threads are needed
  - As many registers as threads are needed

    - Threads do not alternate

Hyper Threading Technology (version of SMT)

- Supports out-of-order execution
- The OS and applications interpret them as 2 separate CPU cores
- The complexity in hardware is around 5-10% while it provides 20-30% performance increase
- 3 modes of operation (depending on which logical CPU is active)
    - ST0 (Single Task 0)
    - ST1 (Single Task 1)
    - MT (Multi Task)
- Changing between the modes happens with HALT instruction
    - It interrupts the CPU's operation

Process level parallel architectures

vv. general description,

ww. motivations of development,

xx. rating, limits,

yy. Amdahl's law,

zz. shared memory,

aaa. distributed memory paradigms

General Description

- Process: version of the program code currently under execution
- On single-core CPUs, during multi-process execution switching among processes takes many clock cycles (2 to 3 thousand)
- The status of a process is contained in the Translation Lookaside Buffer (TLB)
- The process is a Translation Lookaside Entry in the table

Motivation of Development

- Burdens of Instruction-Level Parallelism:
- Branch prediction never yields 100% correct results
- At most 4 to 8 EUs can it handle
- Energy Efficiency:
- Increasing clock frequency of a single CPU by 15% yields 50% more energy-consumption
- Therefore, it is more beneficial to use n number of CPUs with unified frequency rather than one CPU with equivalent performance

Cost Efficiency:

- It is cheaper to create a system with more, cheaper CPUs
- Easier Expandability
- Fault tolerant
- Possible Limitations:
- Finding and utilizing parallelism must happen in software

Amdahl's Law:

- It shows how much performance gain can we achieve using n number of CPUs compared to a 1-CPU system
- $Sp(n)=1/((1-P)+ P/n)$
- P: the ratio of executable program part in parallel
- 1 – P: the ratio of sequentially executable program part
- $P+(1-P)=1$
- Example:
- 5% of our program can execute sequentially, others in parallel, number of CPUs is 100
- $N = 100, P = 0.95$
- $Sp(100)=1/((1-0.95)+ 0.95/100)=16.8$
- Therefore, we can execute the given code 16.8 times faster on 100 CPUs

Classification:

- Shared memory paradigm: every CPU can use shared memory
    - Time of Load / Store instructions is same for every CPU it is not important where the data is placed inside the memory
    - Uniform Memory Access (UMA)
- Distributed memory paradigm: there is no shared memory, processes communicate through messages
    - Processing L / S instruction is non-uniform
    - It is important to place related data close to the CPU
    - Non-Uniform Memory Access (NUMA)
- Cache Coherent NUMA (CC-NUMA)
    - Always keeps track of the location of the most recent instance of data
    - The important is to synchronize data copies all times
    - This is a serious administrative task, it decreases scalability
- Non-Cache Coherent NUMA (NC-NUMA)
- The system does not care about data given to the CPU that it modifies but yet not writes back

Difference between UMA and NUMA:

UMA:

- Used in early multi-core CPUs
- A memory bus is reachable by all CPUs and all Memory modules
- Memory bus is highly overloaded = limited bandwidth

NUMA:

- Memory is physically distributed
    - This way higher memory bandwidth for CPUs is supported
- High bandwidth with low latency
- Highly scalable
- But communicating data among processors is more complex
    - Requires better software to take advantage of memory bandwidth
- However, memory reference can be made by any processor at any memory location in both systems

The roadmap of Core family

bbb. The cancelation of Pentium 4 family,

ccc. the tick-tock model,

ddd. main technological development points

Milestones of the evolution of Desktop and Laptop processors before the Core 2 family:

a) Emergence of 64-bit RISC processors (in the middle of the 1990's):

RISC processors made their move to 64 bit already a couple of years earlier than their CISC counterparts, that is mostly around the middle of the 1990's.

b) Decline of RISC processors (from the end of the 1990's):

At the end of the 1990's clock speeds of CISC processors (Intel's Pentium and AMD's Athlon lines) surpassed that of contemporary RISCs from HP, MIPS, DEC (Alpha line) and others. Due to the more and more serious handicap of RISC processors in terms of clock speed and performance, vendors one of the other cancelled their respective RISC lines.

c) Emergence of 64-bit CISC processors (AMD: 2003, Intel: 2004):

The widening of the width of CISC processors from 32-bit to 64-bit happened in the first half of the 2000's. This move was started by AMD in 2003, followed by Intel in 2004 when they transformed their entire processor spectrum from 32 to 64 bit.

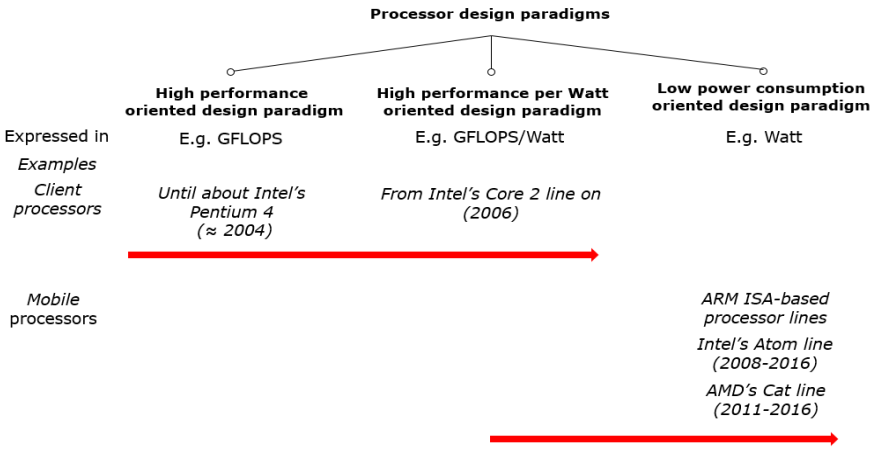
d) Emergence of the multicore era (Intel/AMD: 2005):

An important step in the evolution of processors was the emergence of the multicore era mostly around 2005.

Pentium 4 is a brand by Intel for an entire series of single-core CPUs for desktops, laptops and entry-level servers. Intel released its Pentium 4 family in 2000 while stating that the family will hit a clock rate of 10 GHz and a lifetime of 10 years. Nevertheless, Intel never reached 4 GHz and was forced to cancel the Pentium 4 line after a life cycle of about 4 years in 2004. The reason behind this move is the shifting of Intel's design paradigm from performance orientation to power efficiency orientation to cope with raising dissipation.

In 2003 Intel shifted their focus of processor development from the pure performance goal to power efficiency. Power efficiency is understood as the performance per watt, given e.g. in GFLOPS/W.

Evolution of processor design paradigms



Tick–tock was a production model adopted in 2007 by chip manufacturer Intel. Under this model, every microarchitecture change (tock) was followed by a die shrink of the process technology (tick). Every "tick" represented a shrinking of the process technology of the previous microarchitecture (sometimes introducing new instructions) and every "tock" designated a new microarchitecture. These occurred roughly every year to 18 months. In 2016, Intel announced that it deprecated the tick–tock cycle in favor of a three-step "process–architecture–optimization" model, under which three generations of processors are produced under a single manufacturing process, with the third generation out of three focusing on optimization.

Intel Core is a line of mid- to high-end consumer, workstation, and enthusiast central processing units (CPU) marketed by Intel Corporation. These processors displaced the existing mid- to high-end Pentium processors of the time, moving the Pentium to the entry level, and bumping the Celeron series of processors to the low end. As of 2017, the lineup of Core processors includes the Intel Core i9, Intel Core i7, Intel Core i5, and Intel Core i3, along with the X-series Intel Core CPUs. In mid 2018, the majority of Intel Core processors were found to possess a defect (the Foreshadow vulnerability), which undermines the Software Guard Extensions (SGX) feature of the processor.

The first Intel Core desktop processor—and typical family member—came from the Conroe iteration, a 65 nm dual-core design fabricated brought to market in July 2006, based on the all-new Intel Core microarchitecture with substantial enhancements in micro-architectural efficiency and performance, outperforming Pentium 4 across the board (or near to it), while operating at drastically lower clock rates. Maintaining high instructions per cycle (IPC) on a deeply pipelined and resourced out-of-order execution engine has remained a constant fixture of the Intel Core product group ever since.

The new substantial bump in microarchitecture came with the introduction of the 45 nm Bloomfield desktop processor in 2008 on the Nehalem architecture, whose main advantage came from redesigned I/O and memory systems featuring the new Intel QuickPath Interconnect and an integrated memory controller supporting up to three channels of DDR3 memory.

Subsequent performance improvements have tended toward making additions rather than profound changes, such as adding the Advanced Vector Extensions instruction set extensions to Sandy Bridge, first released on 32 nm in 2011. Time has also brought improved support for virtualization and a trend toward higher levels of system integration and management functionality (making it faster the CPU) through the ongoing evolution of facilities such as Intel Active Management Technology.

Desktop, Laptop, HED CPUs

eee. core numbers,

fff. performance,

ggg. systems topology,

hhh. system architecture development

Core numbers:

When examining the Desktop and Laptop processor core numbers, the maximal number of cores were fixed at 2 cores for a long time. This was increased to 4 around 2011 and that stayed relevant until Coffee Lake in 2017 when the core number rose to 6 cores.(this was significant since AMD already brought out an 8 core processor at the time.) The maximum number of cores did not rise again until 2018 with Coffee Lake R. (8 cores) and then again in

2019 with Comet Lake (10 cores)

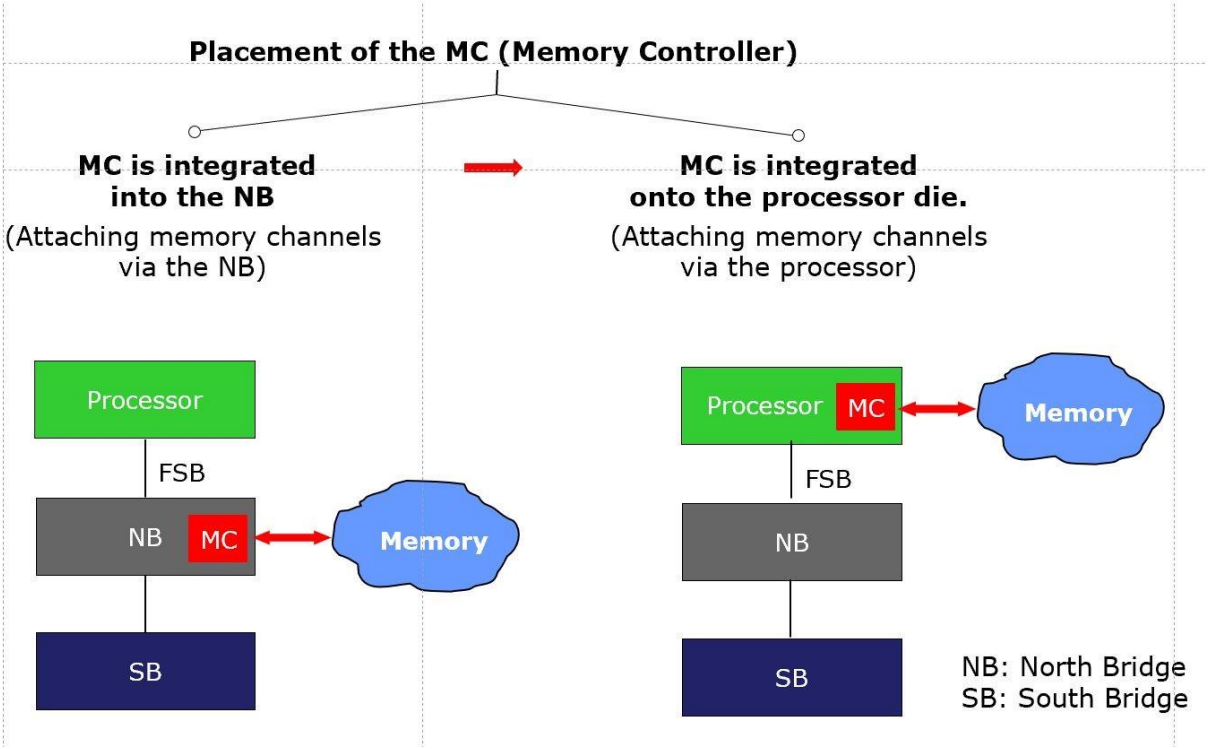
Designation of the platform	MC placem.	Processor	Technology	Max. no. of cores (n_c)	Year of intro.	Processor socket	MCH/PCH	Highest mem./ speed	No. of mem. channels
Anchor Creek	MC in NB	Pentium D	90 nm	2x1	5/2005	LGA 775	945-955	DD2-667	2
Bridge Creek		Core 2	65 nm	2C	6/2006	LGA 775	945-975	DDR2-800	2
Salt Creek		Core 2 Quad	65 nm	2x2C	6/2007	LGA 775	3 Series (Bearlake)	DDR3-1067	2
Boulder Creek		Penryn	45 nm	2x2C	6/2008	LGA 775	4 Series (Eaglelake)	DDR3-1067	2
Kings Creek	On-die MC	2. G. Nehalem (Lynnfield)	45 nm	4C	9/2009	LGA 1156	5 Series (Ibex Peak)	DDR3-1333	2
		Westmere (Clarkdale)	32 nm	2C+G	1/2010	LGA 1156	5 Series (Ibex Peak)		
Sugar Bay		Sandy Bridge	32 nm	2C/4C+G	1/2011	LGA 1155	6 series (Cougar Point)	DDR3-1333	2
Maho Bay		Ivy Bridge	22 nm	4C +G	4/2012	LGA 1155	7 series (Panther Point)	DDR3-1600	2
Shark Bay		Haswell	22 nm	4C+G	6/2013	LGA 1155	8 Series (Lynx Point)	DDR3-1600	
		Haswell refresh	22 nm	4C+G	5/2014	LGA 1150	9 Series (Wild Cat Point)	DDR3-1666	2
		Broadwell	14 nm	4C+G	6/2015	LGA 1150	9 Series (Wild Cat Point)	DDR3-1866	
		Skylake	14 nm	4C+G	10/2015	LGA 1151	100. Series (Sunrise Point)	DDR4-2133	2
		Kaby Lake	14 nm	4C + G	8/2016	LGA 1151	200 Series	DDR4-2400	2
		Coffee Lake	14 nm	6C +G	10/2017	LGA 1151	300 Series	DDR4-2666	2
		Amber Lake	14 nm	2C+G	8/2018	BGA 1515	300 Series	LPDDR3-2133	2
		Whiskey Lake	14 nm	4C+G	8/2018	BGA 1528	300 Series	DDR4-2400	2
		Coffee Lake R.	14 nm	8C +G	10/2018	LGA 1151	300 Series	DDR4-2666	2
		Comet Lake	14 nm	10C + G	8/2019	BGA 1528	400 Series	DDR4-2666	2
		Cannon Lake	10 nm	2C	5/2018	BGA	n.a.	DDR4-2400	2
		Ice Lake	10 nm	4C	7/2019	BGA 1526/1377	300-series PCH	DDR4-3733	2
		Tiger Lake	10 nm	4C	09/2020			LPDDR4x-4267	2

Number of Memory Channels:

Has to be examined in connection with the number of cores. A system with 2 cores can be fulfilled with 2 memory channels.

System Topology:

The placement of the Memory Controller is based on two topologies:



On-die MC:

- There is two kind of reach time (in case of 2 processor system):
- 1. Local reach time: The data is stored in the processor's own memory
- 2. Remote reach time: The data is stored on the other processor memory

Highest Memory speed:

In the case of the early systems it was DD2-667. The number represent how much data can be transferred in one step (effective data rate = memory frequency * number of transferable bytes per clock rate.). The modern memory speed rose to DDR4-3733 by 2019, while in 2020 it rose further to LPDDR4x-4267. This means a 6-7 times speed improvement in 15 years.

Graphics family evolution:

Intel Core generation	Graphics generation	Models	Graphics Technology level	No. of graphics slices	No. of EUs	eDRAM	OpenGL version	DirectX version	OpenCL version
Westmere	5th (Ironlake)	HD			12	--	2.1	10.1	n.a.
Sandy Bridge	6th	HD 2000	GT1	1 (2x3 EU)	6		3.1/3.3	10.1	n.a.
		HD 3000	GT2	1 (4x3 EU)	12				
Ivy Bridge	7th	HD 2500	GT1	1 (6 EU)	6	--	4.0	11.0	1.2
		HD 4000	GT2	1 (2x8 EU)	16				
Haswell	7.5th	HD 4200-HD 4700	GT2	1 (2x10 EU)	20	--	4.3	11.1	1.2
		HD 5000 Iris 5100	GT3	2	40				
		Iris Pro 5200				128 MB			
Broadwell	8th	HD 5300-HD 5600	GT2	1 (3x8 EU)	23/24	--	4.3	11.2	2.0
		HD 6000 Iris 6100	GT3	2	47/48				
		Iris Pro 6200	GT3e	2	48	128 MB			
Skylake	9th	HD 510	GT1	1 (3x4 EU)	12	--	4.4	12	2.0
		HD 515	GT1.5	1 (3x6 EU)	18				
		HD 520	GT2	1 (3x8 EU)	24				
		HD 535	GT3	2	48				
		HD 540	GT3e	2	48	64 MB			
		HD 580	GT4e	3	72	64/128 MB			

The Graphical Generations (5th to 9th):

The predecessor is Sandy Bridge, since it was the first processor with integrated graphics. Westmere is considered to be the first, since it was the first processor with a Graphical core in the processor.
The Graphics Technology Levels (GT1-GT4):

- Not related to generations.
- The graphics technology levels indicate the number of graphics slices (replicable sets of graphics EUs) within each graphics generation.
- Most important Level is GT2 The number of EUs:

The number of graphics EUs is increasing more or less according to Moore's rule (from 12 in 2010 to 72 in 2015).

HEDTs: (high-end desktop)
Definition: They aim at high performance desktops for hardcore gamers and grapichs enthusiasts.
Main requirements to HEDT lines for supporting multiple (up to 4) discrete graphics cards vs. mainstream desktops:

- More PCIe (Peripheral Component Interconnect Express) lanes (typically 8 or 16 lanes per card)
- More cores
- More memory channels (to suitably service more cores)

The Kaby Lake-X models in fact do not fit into the traditional HEDT lines, they are actually the highest performance models of the Kaby Lake line and are designated as mobile models. The biggest

difference between normal Desktop and HEDs is the PCIe lanes. HEDs have 40 on average. The PCIe technology used improves as well. (2.0 to 3.0 from 2012)

Unlocked clock multiplier:

- Intel HEDs are typically unlocked (clock multiplier is not fixed) to allow overclocking.
- The clock multiplier multiplies the bus frequency (100/133 MHz) by a number delivered by the PCU Memory channels:

- 2 memory channels are not enough to supply the constantly rising number of cores
- The number of the memory channels doubled to 4 channels. (this is expected to increase with the number of the cores) Thermal Design Power:
- Huge amount of dissipation is connected to the performance, currently around 165 W which started from 65 W.

HEDs key points:

- HEDs are equipped with a large number of PCIe lanes (32 to 48 lanes) to connect multiple discrete graphics cards vs only 16 as a typical in desktops
- No need for integrated graphics as the installation is assumed to provide high quality graphics by attaching discrete graphics cards
- HEDs have more cores than desktops, since more parallelism than typical desktop workloads.
- HEDs are unlocked
- HEDs have a higher TDP mostly of 130 to 165 W vs 65 to 95 W typical for DT processors

Server CPUs

iii. core numbers,

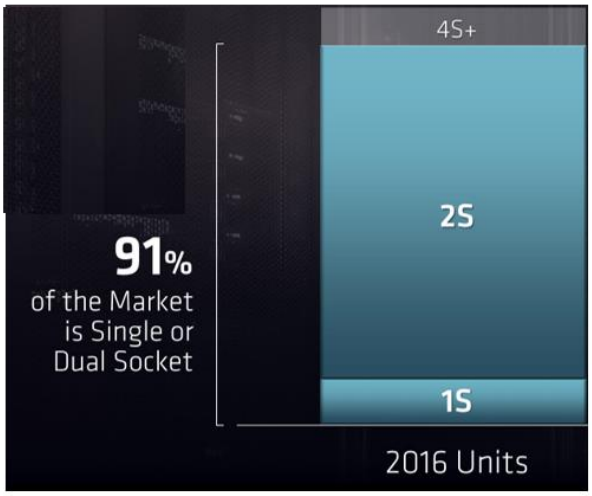
jjj. systems topology,

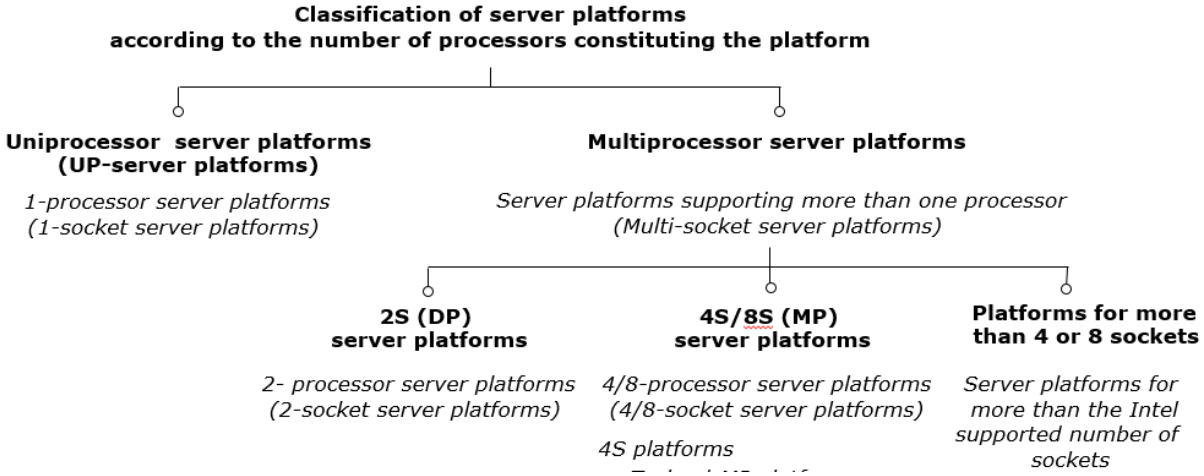
kkk. memory connection and bandwidth,

lll. UMA and NUMA

Server architectures usually include multiple processors, each of them having multiple cores.

Recent market share of 1S and 2S servers [2]

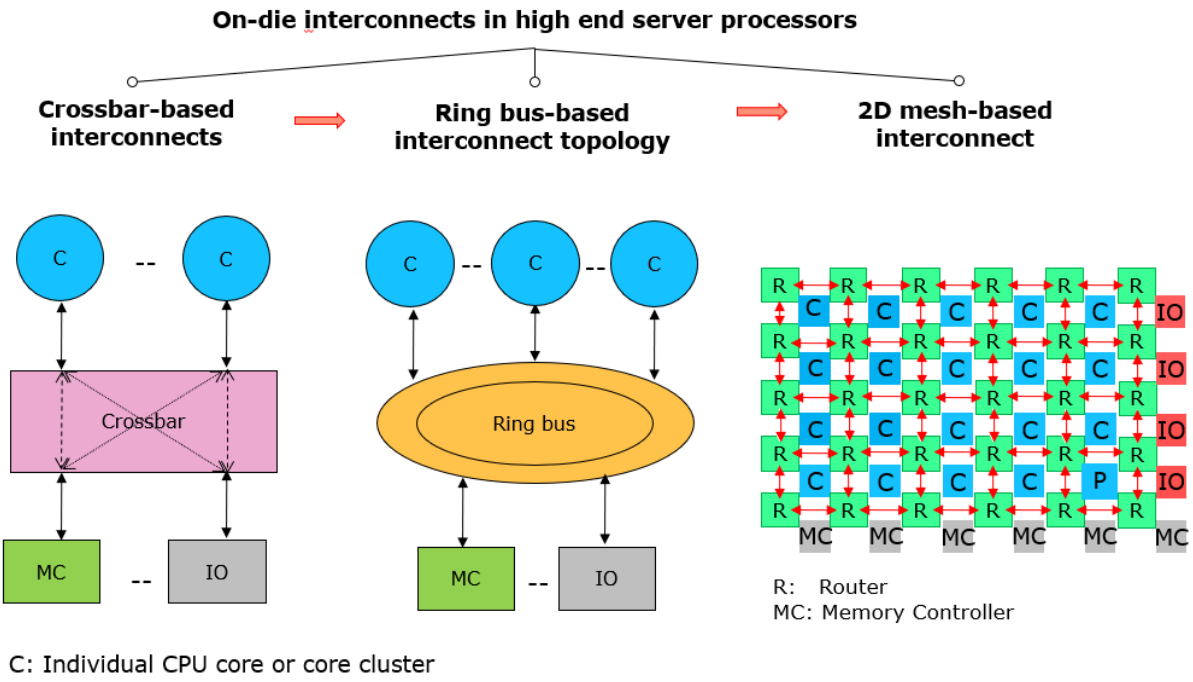




Intel's first multicore server processors were based on the 3. core of the Pentium 4 processor family (called Prescott), and were introduced about 2005.

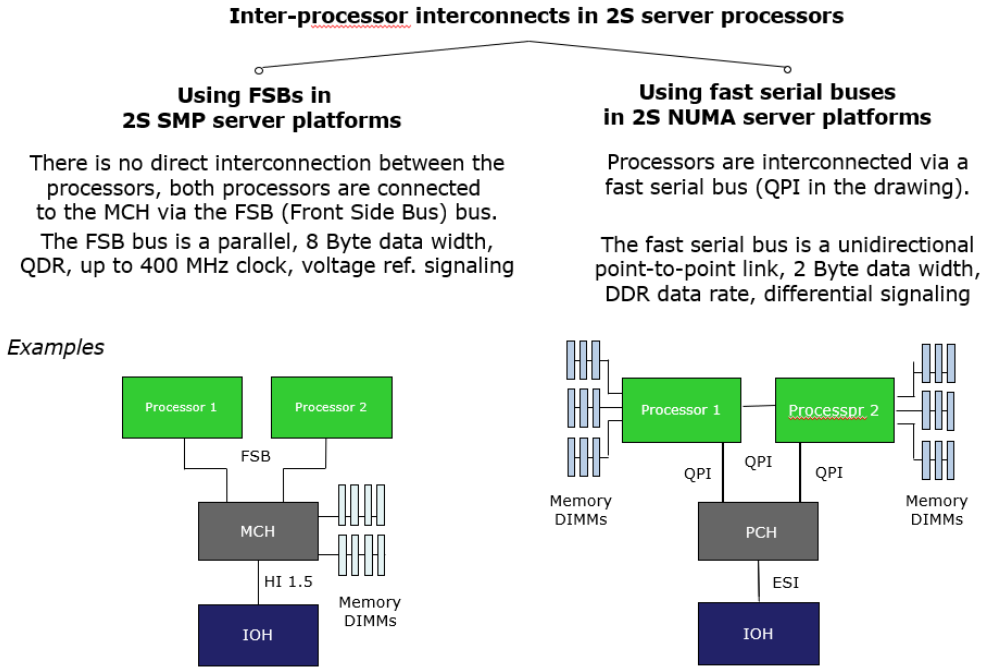
As far as Intel's server processor lines are concerned, we can talk about the Xeon and Itanium lines. For AMD, it is the EPYC line that is designed for the purpose of multicore server processors.

Intra-processor interconnects



Crossbar-based on-die interconnects were widely used in Intel's and AMD's multicore servers in the beginning of the 2010's and off-die interconnects are also employed in AMD's recent EPYC server lines to interconnect multiple dies. With higher core counts crossbar switches became more and more expensive to implement due to the highly increasing number of switches needed. Individual cores-based processors with more than 4 cores switched typically from crossbar type interconnects to ring interconnects. Ring interconnects typically have two counter rotating data rings to reduce latency. Long core-to-core transfer times may limit performance, this motivated the introduction of 2D mesh interconnects.

Inter-processor interconnects



Memory bandwidth bottleneck problem

Both in Intel's 2S and 4S server processors but also in AMD's previous Bulldozer based processor lines memory speeds has been raised slower than core counts. If in subsequent processor generations the memory transfer rate raises slower than the core count, the per core memory bandwidth becomes smaller. Consequently, a memory bandwidth gap would emerge unless the number of memory channels will be raised.

While a processor family evolves generations over generations and core counts rise at a higher rate than memory speeds, the resulting per core memory bandwidth deficit should be compensated by appropriately increasing the number of memory channels of the platform.

Providing enough memory channels and thus memory bandwidth per socket is one of the key challenges in designing server processors and platforms since multicores emerged since core counts continuously raise at a higher rate than memory speeds, but electrical and power issues constrain the extension of memory channels. In order to preserve the per core memory bandwidth the number of memory channels rose rapidly from 1 per socket to 12 per socket in Intel's high-end 2S server platforms.

Multiprocessor systems:

A multiprocessor system is defined as "a system with more than one processor", and, more precisely, "a number of central processing units linked together to enable parallel processing to take place". The term "multiprocessor" can be confused with the term "multiprocessing". While multiprocessing is a type of processing in which two or more processors work together to execute multiple programs simultaneously, multiprocessor refers to a hardware architecture that allows multiprocessing.

Multiprocessor systems can be classified according to how processor memory access is handled:

- Symmetric Multiprocessors (SMPs, UMAs)
- Distributed shared memory systems (DSM, NUMAs)

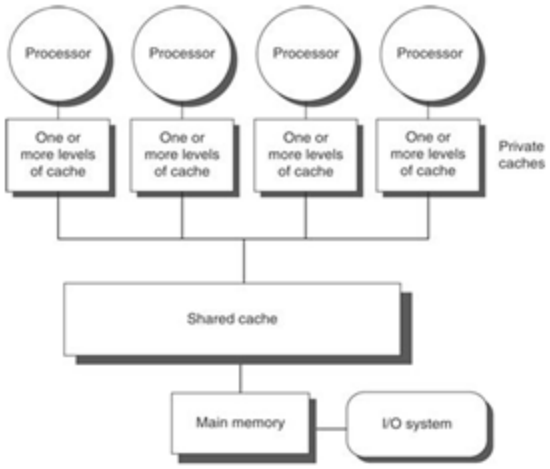
Symmetric Multiprocessors (SMPs, UMAs)

Symmetric Multiprocessors (SMPs) features small number of processors

- With such a small number of processor counts, it is possible for processors to share a single centralized memory
- In multicore chips the memory is shared in a centralized way, so the existing multicore CPUs are SMPs
- SMP architectures sometimes called uniform memory access multiprocessors (UMA)

Symmetric multiprocessing (SMP) involves a multiprocessor computer hardware and software architecture where two or more identical processors are connected to a single, shared main memory, have full access to all input and output devices, and are controlled by a single operating system instance that treats all processors equally, reserving none for special purposes.

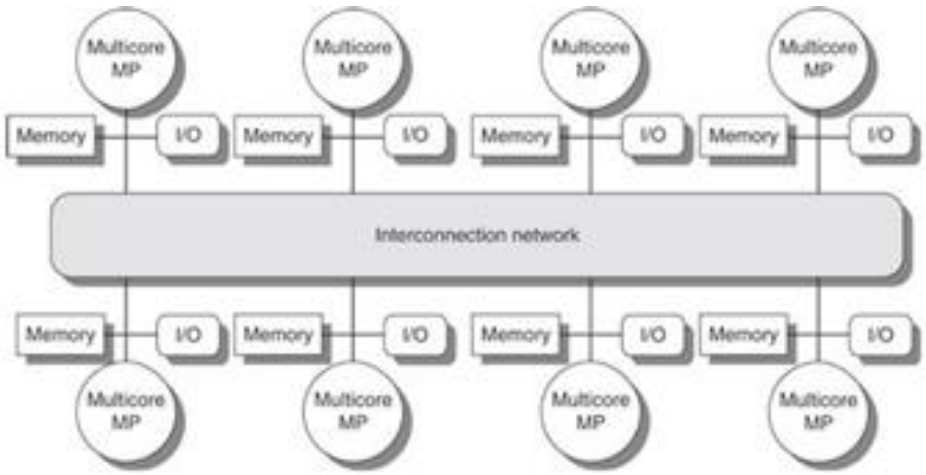
Uniform memory access (UMA) is a shared memory architecture used in parallel computers. All the processors in the UMA model share the physical memory uniformly. In a UMA architecture, access time to a memory location is independent of which processor makes the request or which memory chip contains the transferred data. In the UMA architecture, each processor may use a private cache. Peripherals are also shared in some fashion. The UMA model is suitable for general purpose and time sharing applications by multiple users. It can be used to speed up the execution of a single large program in time-critical applications.



Distributed shared memory systems (DSM, NUMAs)

In computer science, distributed shared memory (DSM) is a form of memory architecture where physically separated memories can be addressed as one logically shared address space. A distributed-memory system, consists of multiple independent processing nodes with local memory modules which is connected by a general interconnection network.

Non-uniform memory access (NUMA) is a computer memory design used in multiprocessing, where the memory access time depends on the memory location relative to the processor. Under NUMA, a processor can access its own local memory faster than non-local memory (memory local to another processor or memory shared between processors). The benefits of NUMA are limited to particular workloads, notably on servers where the data is often associated strongly with certain tasks or users.



Why - Basic concept:

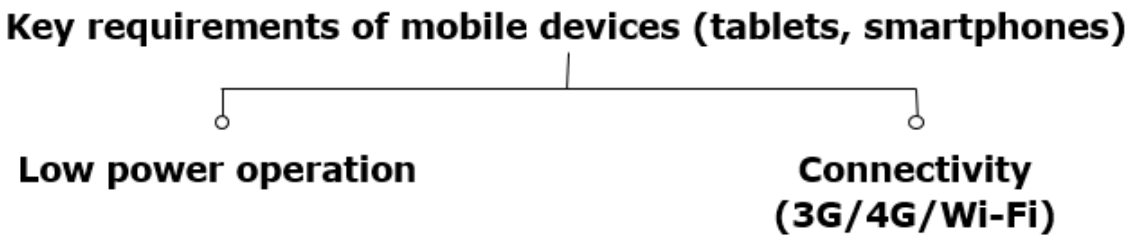
Modern CPUs operate considerably faster than the main memory they use. CPUs increasingly have found themselves "starved for data" and having to stall while waiting for data to arrive from memory. Limiting the number of memory accesses provided the key to extracting high performance from a modern computer. For commodity processors, this meant installing an ever-increasing amount of high-speed cache memory and using increasingly sophisticated algorithms to avoid cache misses. But the dramatic increase in size of the operating systems and of the applications run on them has generally overwhelmed these cache-processing improvements. Solution: NUMA, NUMA attempts to address this problem by providing separate memory for each processor, avoiding the performance hit when several processors attempt to address the same memory.

CPUs for tablets and mobile phones

mmm. **design constraints,**

nnn. **the reason for the failure of Intel and AMD mobile CPUs,**

ooo. **the ARM's advantage**



Low power consumption is expressed

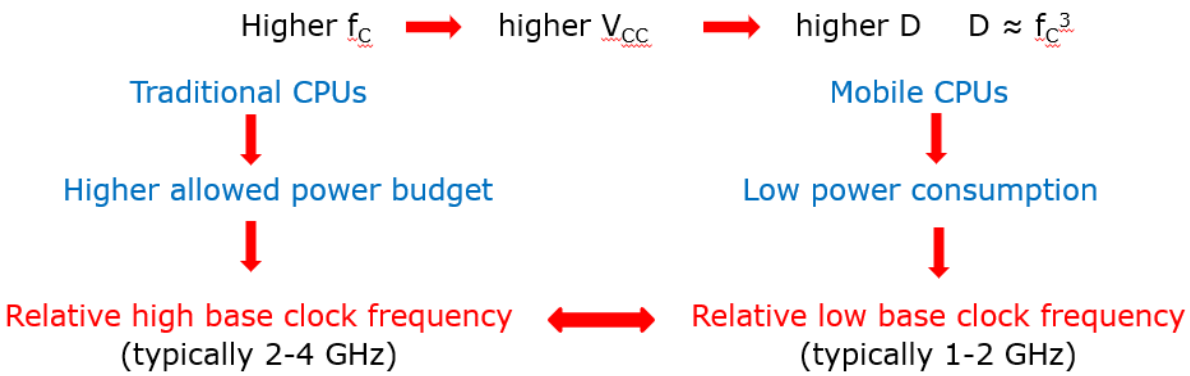
- either by specifying the power consumption, e.g. the TDP value of the processor in Watt,
- or in in length of the operating hours of the device under given conditions.

Contrasting the design paradigms of traditional and mobile processors

Traditional processors		Tablets and smartphones
High performance/power (Power efficiency) (e.g. GFLOPS/Watt)	⟷	Low power consumption (Watt) (Number of operating hours)

Intel's and AMD's traditional CPUs are designed for high performance/power, but mobile devices require low power consumption, consequently Intel's and AMD's traditional microarchitectures are not suited for mobile devices. To avoid shrinking market shares on the global processor market and benefit from the rapidly increasing mobile market Intel and AMD needed processors that are competitive with ARM based designs. Thus Intel and AMD were forced

- to introduce new, narrow (i.e. 2-wide), low-power microarchitectures for their CPUs and
- clock them at a relatively low rate.



To achieve low power consumption first application processors used in tablets and smartphones were "thin" designs with a typical decode width of 2. Later, advancements in IC technology and power management allowed to widen the microarchitecture of mobiles without noticeably raising their power consumption.

Apple's A10 (used in the iPhone 7 (2016) introduced the widest (7-wide) microarchitecture, and Apple's subsequent processors up to the latest A13 (Bionic 2019) chip (used in the iPhone 11 and iPhone 11 Pro) have a 7-wide front-end. The A13 is the most powerful mobile processor to date.

Intel introduced the Atom processors to penetrate into the market. In 2015 they achieved the 3. place in the worldwide market share in tablet application processor revenue.

Tablet application processors worldwide market share 2015 (revenue) [57]	
Apple (USA)	31 %
Qualcomm (USA)	16 %
Intel (USA)	14 %

Due to their high losses Intel first stopped paying subsidies and then in 2016 announced their withdrawal from the mobile market. AMD also withdrew from the market at approximately the same time.

When it comes to AMD, their Cat line (Bobcat, Jaguar, Puma) was aimed at the smartphone market. They were also unsuccessful in gaining a significant market share, but the Jaguar and Puma architectures from AMD are used in game consoles nowadays(Playstation, XBOX).

Connectivity may include (depending on the model features)

- LAN connectivity
- Wi-Fi connectivity (coined after Hi-Fi)
- mobile broadband connectivity (recently 3G/4G).

Integrating the modem onto the application processor die results in less costs and shorter time to market. Qualcomm pioneered this move designing integrated parts already about 1996.

ARM vs. x86

An ARM processor is a Reduced Instruction Set Computer (RISC) processor.

Complex Instruction Set Computer (CISC) processors, like the x86, have a rich instruction set capable of doing complex things with a single instruction. Such processors often have significant amounts of internal logic that decode machine instructions to sequences of internal operations (microcode).

RISC architectures, in contrast, have a smaller number of more general purpose instructions, that might be executed with significantly fewer transistors, making the silicon cheaper and more power efficient. Like other RISC architectures, ARM cores have a large number of general-purpose registers and many instructions execute in a single cycle. It has simple addressing modes, where all load/store addresses can be determined from register contents and instruction fields.

ISA Extensions

ppp. x86 ISA extensions,

qqq. ARM ISA extensions,

rrr. details of the x86 SIMD extensions

An instruction set architecture (ISA) is an abstract model of a computer. It is also referred to as architecture or computer architecture. A realization of an ISA, such as a central processing unit (CPU), is called an implementation.

In general, an ISA defines the supported data types, the registers, the hardware support for managing main memory fundamental features (such as the memory consistency, addressing modes, virtual memory), and the input/output model of a family of implementations of the ISA.

An ISA specifies the behavior of machine code running on implementations of that ISA in a fashion that does not depend on the characteristics of that implementation, providing binary compatibility between implementations. This enables multiple implementations of an ISA that differ in performance, physical size, and monetary cost (among other things), but that are capable of running the same machine code, so that a lower-performance, lower-cost machine can be replaced with a higher-cost, higher-performance machine without having to replace software.

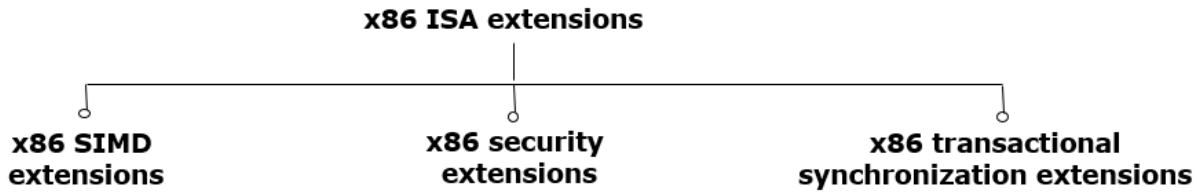
An ISA can be extended by adding instructions or other capabilities, or adding support for larger addresses and data values; an implementation of the extended ISA will still be able to execute machine code for versions of the ISA without those extensions. Machine code using those extensions will only run on implementations that support those extensions.

The binary compatibility that they provide make ISAs one of the most fundamental abstractions in computing.

x86 is a family of instruction set architectures initially developed by Intel based on the Intel 8086 microprocessor and its 8088 variant. Many additions and extensions have been added to the x86 instruction set over the years, almost consistently with full backward compatibility.

As of 2018, the majority of personal computers and laptops sold are based on the x86 architecture, while other categories—especially high-volume mobile categories such as smartphones or tablets—are dominated by ARM; at the high end, x86 continues to dominate compute-intensive workstation and cloud computing segments.

In the 1980s and early 1990s, when the 8088 and 80286 were still in common use, the term x86 usually represented any 8086 compatible CPU. Today, however, x86 usually implies a binary compatibility also with the 32-bit instruction set of the 80386. This is due to the fact that this instruction set has become something of a lowest common denominator for many modern operating systems and probably also because the term became common after the introduction of the 80386 in 1985.



x86 SIMD extension

Graphics, image processing, games etc., emerged in the first half of the 1990s. These new applications induced new requirements, like

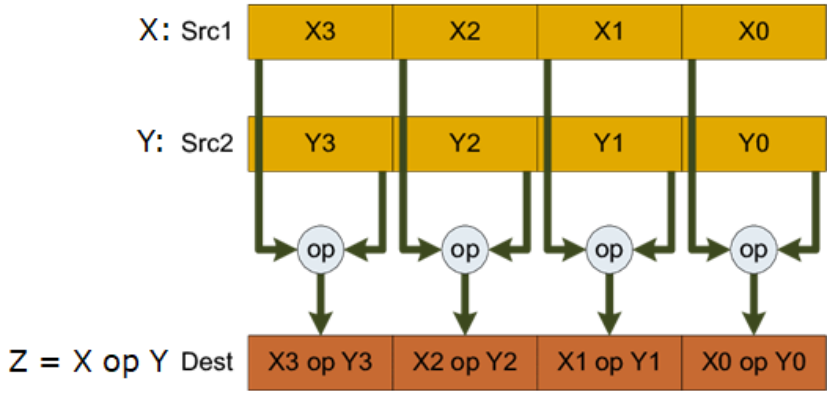
- speeding up graphics processing by using higher speed buses than the PCI bus for connecting graphics card to the platform,
- introducing low cost, low performance graphics processing by integrating graphics processing units into north bridges or
- speeding up calculations by vector processing.

Intel met these requirements by introducing

- the AGP graphics interface standard and bus in 1996,
- integrating graphics into the north-bridge, first, into the 810 GMCH chipset in 1999 and
- SIMD processing by means of the 64-bit MMX (Multi-Media eXtension) of the x86 ISA in their Pentium-MMX in 1997.

Note that at that time CISC processors were 32-bit wide (as opposed to the typically 64-bit wide RISC processors).

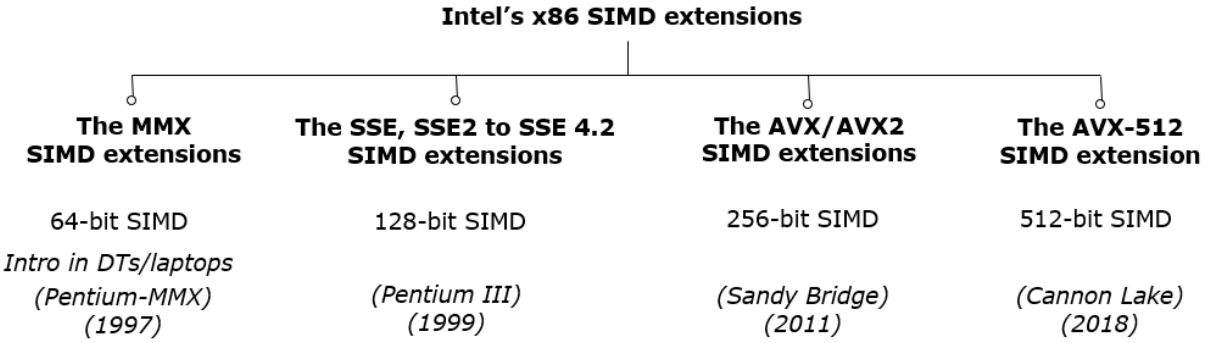
Interpretation of the execution of SIMD instructions



SIMD instructions perform at the same time the same operation on multiple fields of registers.

SIMD instructions may refer either to fixed point (FX) data or floating point (FP) data. FX SIMD instructions support mainly multimedia applications, by executing (2/4/8 or more) FX-operations on pixels in-parallel, whereas FP SIMD instructions accelerate in the first line 3D graphics by executing multiple FP-operations in parallel.

Main instruction groups of Intel's x86 SIMD extension



The 64-bit MMX (Multi-Media eXtension)

It provides 64-bit vector processing for FX data (1x64, 2x32, 4x16, 8x8-bit). MMX does not have its own register set, instead the 64-bit mantissa part of the existing FP register space is employed for SIMD processing. This forbids mixing FP and SIMD computation in the same routine. MMX offers 56 FX SIMD instructions.

The 128-bit SSE extension

It offers

- eight dedicated 128-bit registers (XMM0-XMM7)
- new 128-bit FP data type (4x SP FP)
- 70 new instructions.

The new 128-bit XMM registers can be used only for packed SP FP operands, but FX SIMD data will be processed further on in the 64-bit MMX0-MMX7 registers.

Intel's SSE2

It provides

- 6 new data types
- 140 new instructions

Intel's SSE3

13 new instructions are added.

AMD's move

In their x86-64 ISA extension – published in 1999 - AMD doubled the number of XMM registers (from 8 to 16) in the 64-bit operating mode.

Intel's move

Intel's 64-bit x86 extension was called EM64T (Extended Memory Technology). It is the same as AMD's x86-64 extension named differently.

256-bit AVX/AVX2 and 512-bit AVX-512

As part of the previous SSE extension, Intel introduced the 128-bit XMM [0, 15] register space in the 64-bit execution mode. The YMM register space provides 16 256-bit wide registers named (ymm0,ymm15). The lower 128 bits of the YMM registers are aliased to the respective 128-bit XMM registers.

Intel introduced 256-bit SIMD extensions in two steps:

a) first only FX SIMD operations in the AVX extension and then

b) added FP SIMD operations as well in the AVX2 extensions,

They introduced the appropriate instructions to process 256-bit FP data in the extended YMM [0, 15] register space.

x86 Security extensions

They provide a number of enhancements to improve security, including

- SMX (Safer Mode Extensions)
- AES-NI (Enhanced Encryption Standard-New Instructions) and

- MPX (Memory Protection Extensions),
- SGX (Software Guard Extensions)

Subsequently, in response to the revealed security risks, like Spectre or Meltdown, Intel introduced a large number of security extensions in their Ice lake line.

SMX (Safer Mode Extensions)

Introduced in the Core 2 family (2006). Safer Mode Extensions allow to establish a trusted execution environment by invoking the GETSEC group of instructions. It is part of Intel's Trusted Execution Technology (TXT), designated before 2006 as the LeGrande technology. TXT protects data within virtualized computing environments. TXT became part of Intel's business oriented vPro processor technology (released in 2007) that is designed to provide protection against software attacks, like viruses and other threats and also to enable remote PC management.

AES-NI (Enhanced Encryption Standard-New Instructions)

AES-NI includes 7 instructions and became supported first in Intel's 32 nm Westmere line of processors in 2010. It supports the AES (Advanced Encryption Standard - It is currently one of the most popular algorithms used for data transmission according to the SSL standard. The AES algorithm uses 128, 192 or 256 bit long cryptographic keys to encrypt and decrypt data in blocks of 128 bits).

MPX (Memory Protection Extensions)

Introduced in the Skylake (2015). It performs runtime buffer overflow checks to ensure that physical memory accesses fall within the bounds of the memory allocated to the calling process.

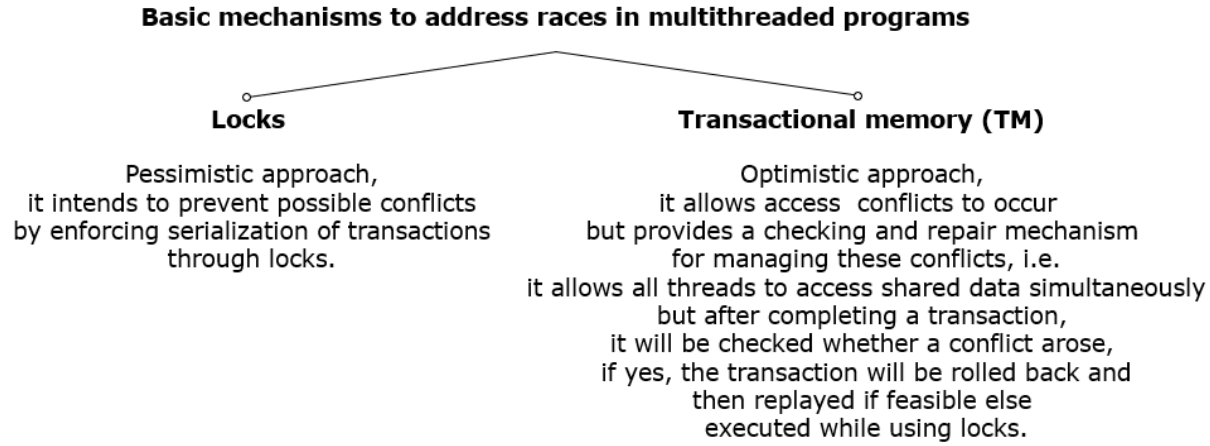
SGX (Software Guard Extensions)

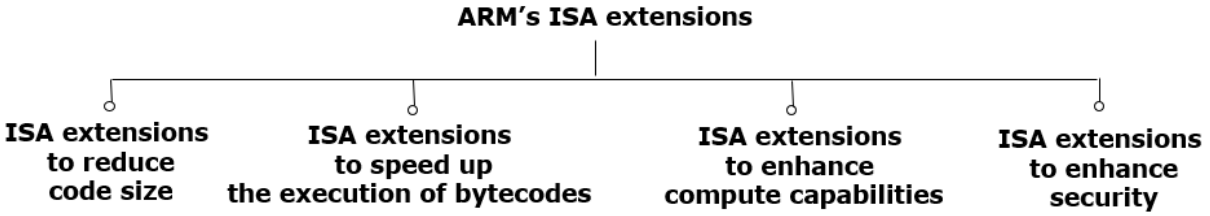
It is a set of security-related instructions introduced along with the Skylake architecture (2015). It allows user- or OS-level code to build private regions of memory, called enclaves, whose contents are protected from reading or saving by any process outside the enclave itself, even if the process runs at a higher privilege level.

x86 TSX (Transactional Synchronization Extensions)

TSX supports Transactional Memory in hardware. Transactional memory extensions (actually four instructions) provide an efficient synchronization mechanism in concurrent programming used to effectively manage race conditions occurring when multiple threads access shared data. Nevertheless, in 2014 Intel announced a bug in the TSX implementation on all current steppings of all Haswell models and disabled the TSX feature on affected CPUs via a microcode update. Subsequently, TSX became enabled first on a Broadwell model in 2014 and then on the Haswell-EX in 2015.

Basically there are two mechanisms to address race conditions in multithreaded programs





Jazelle DBX (Direct Bytecode eXecution) is a technique that allows Java bytecode to be executed directly in the ARM architecture.

Thumb

To improve compiled code-density, processors since the ARM7TDMI (released in 1994) have featured the Thumb instruction set, which have their own state. (The "T" in "TDMI" indicates the Thumb feature.) When in this state, the processor executes the Thumb instruction set, a compact 16-bit encoding for a subset of the ARM instruction set. Most of the Thumb instructions are directly mapped to normal ARM instructions. The space-saving comes from making some of the instruction operands implicit and limiting the number of possibilities compared to the ARM instructions executed in the ARM instruction set state.

Thumb-2 technology was introduced in the ARM1156 core, announced in 2003. Thumb-2 extends the limited 16-bit instruction set of Thumb with additional 32-bit instructions to give the instruction set more breadth, thus producing a variable-length instruction set. A stated aim for Thumb-2 was to achieve code density similar to Thumb with performance similar to the ARM instruction set on 32-bit memory.

The **Advanced SIMD extension** (aka **NEON** or "MPE" Media Processing Engine) is a combined 64- and 128-bit SIMD instruction set that provides standardized acceleration for media and signal processing applications. NEON is included in all Cortex-A8 devices, but is optional in Cortex-A9 devices. NEON can execute MP3 audio decoding on CPUs running at 10 MHz, and can run the GSM adaptive multi-rate (AMR) speech codec at 13 MHz. It features a comprehensive instruction set, separate register files, and independent execution hardware.

The **Security Extensions**, marketed as **TrustZone** Technology, is in ARMv6KZ and later application profile architectures. It provides a low-cost alternative to adding another dedicated security core to an SoC, by providing two virtual processors backed by hardware based access control. This lets the application core switch between two states, referred to as worlds (to reduce confusion with other names for capability domains), in order to prevent information from leaking from the more trusted world to the less trusted world. This world switch is generally orthogonal to all other capabilities of the processor, thus each world can operate independently of the other while using the same core. Memory and peripherals are then made aware of the operating world of the core and may use this to provide access control to secrets and code on the device.

Evolution of memory subsystems

sss. Options for attaching memory channels to a client platform (MCH or direct CPU attachment, direct or indirect interfaces),

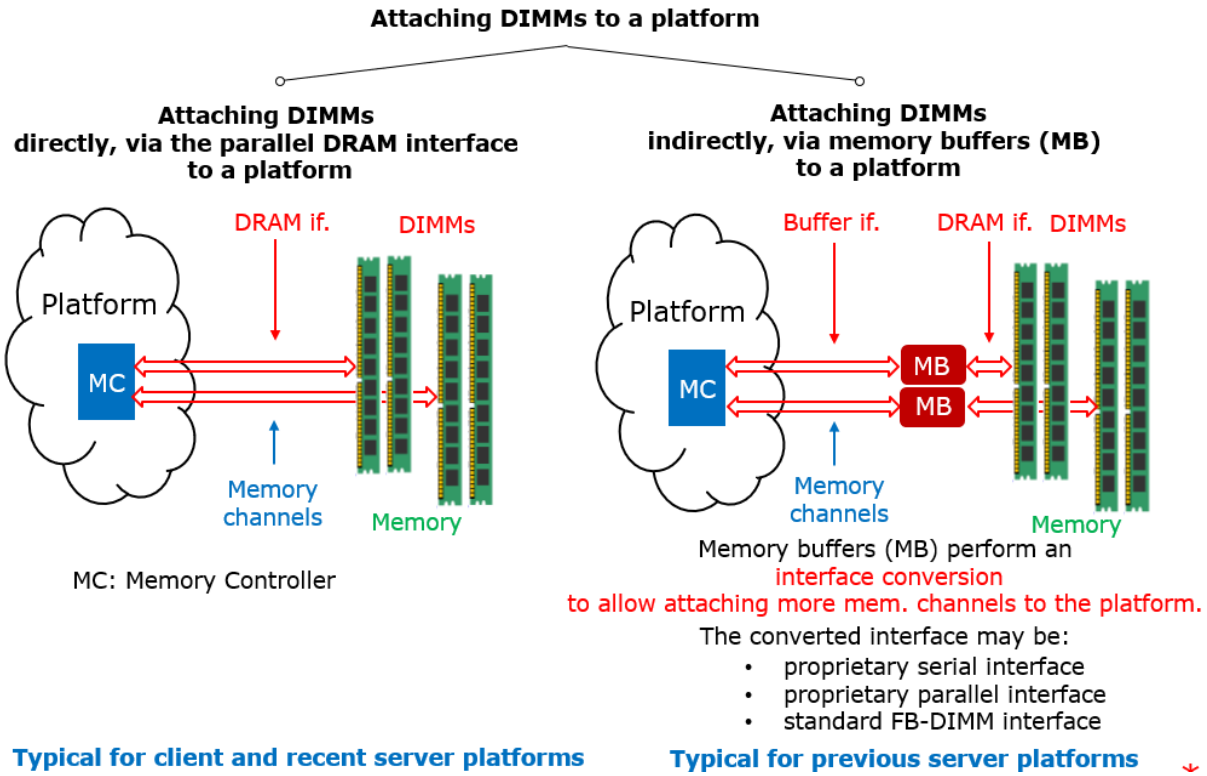
ttt. Importance of the bandwidth of the memory

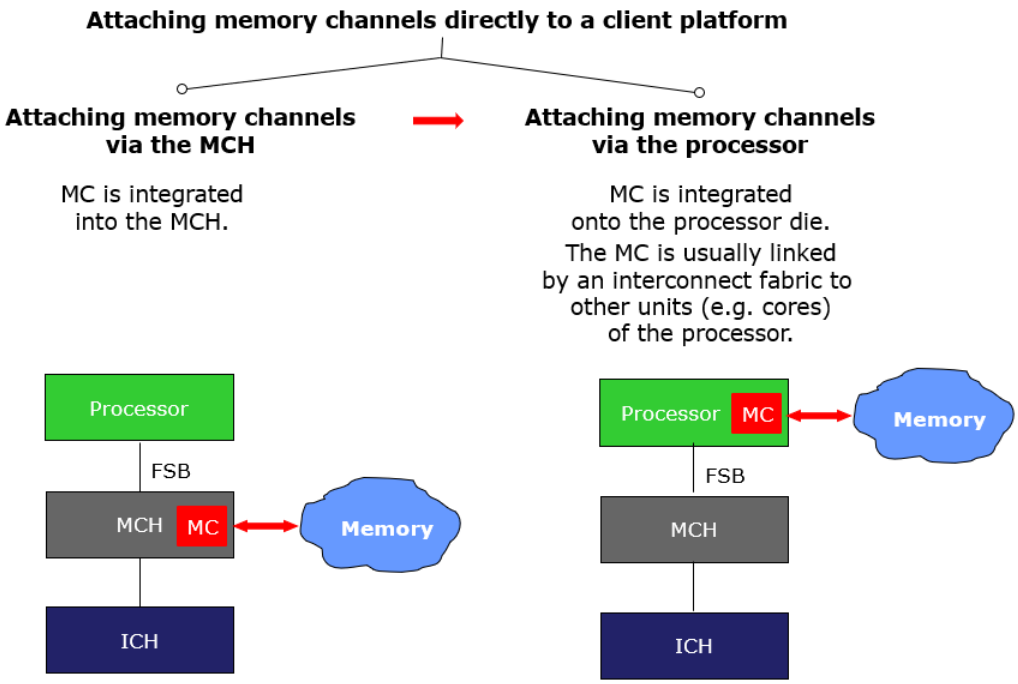
The memory subsystem serves the entire platform, including the CPU cores. Each CPU core operates on its own, accordingly each core needs the same memory bandwidth for accessing instructions and data, in other words, the memory bandwidth has to be linearly scaled with the core count.

The per core memory bandwidth in Intel's DT lines has astonishingly increased, rather than decreased due to the slow rise of core counts and massive boost of memory rates. AMD's client processors (with 2 to 12 cores) provide up to two memory channels, including AMD's Zen 2-based 7 nm 12 core Ryzen 9 3900X, introduced in 2019. Two memory channel for so many cores might seem less than required, but the per core memory bandwidth has remained almost the same despite the massive growth of the core counts due to a noteworthy increase of memory speeds. A comparison reveals that both vendors provide in their multicore client processors about the same per-core bandwidth of 3.2 GB/s to 6.4 GB/s.

Options for attaching memory to a client platform

In the platform one or more memory controllers are responsible for connecting the memory. Memory chips are placed on DIMMs (Dual-In-Line Memory Modules). DIMMs are connected to the memory controller or controllers either directly or indirectly, as indicated in the next Figure.





Note that AMD began attaching memory channels via the processor very early in 2003, 5 years before Intel.

Why attaching memory via the processors became widespread?

In client processors, memory was attached via the MCH (Memory Control Hub) for a long time, e.g. up to Intel's Nehalem line (2008), while implementing up to two memory channels. That configuration is more or less balanced for DTs and laptops of that time, that were built up of up to dual-core processors, as far as the computing performance and the memory bandwidth is concerned that is needed for supplying instructions and data, while running usual client workloads. E.g. the considered platform has an FSB (Front Side Bus) bandwidth of 8.5 GB/s

and is served by a memory bandwidth of 2x 6.4 GB/s = 12.8 GB/s. Nevertheless, when servers, e.g. 4-socket servers, would have been attached to the memory in a similar way, i.e. through memory channels connected to the MCH, this system architecture would easily be proved memory bandwidth limited, since now instead of one e.g. four processors would share the available memory bandwidth, as seen in the next Figure.

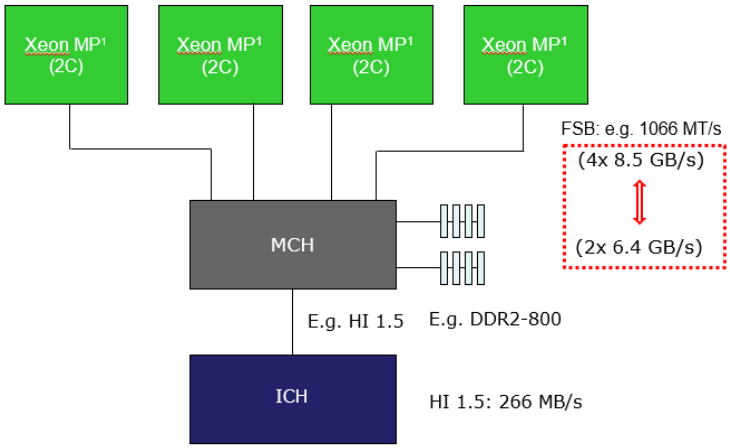


Figure: A hypothetical 4S server platform built up of Core 2 processors

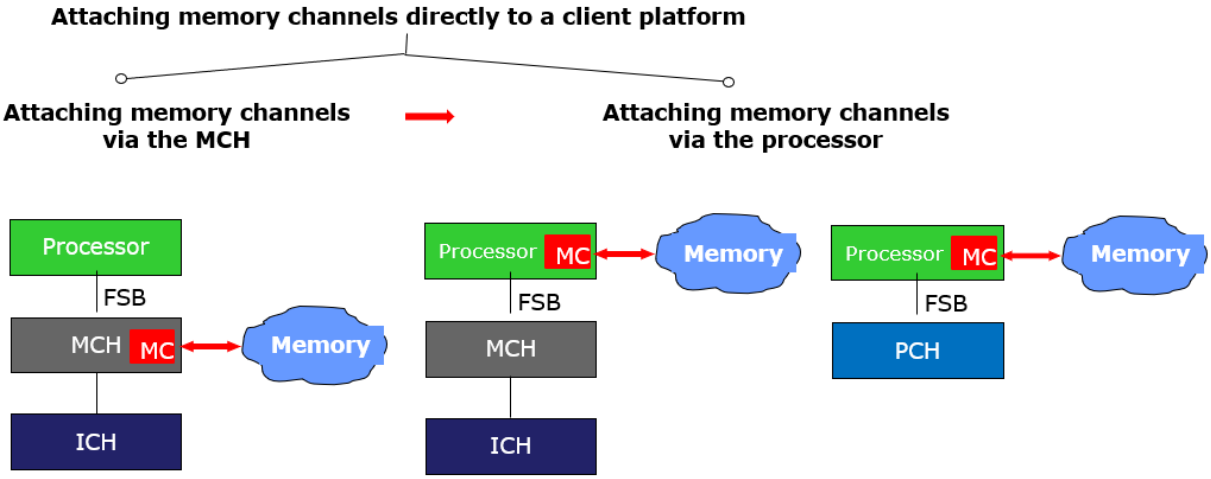
Obviously, in the considered 4-socket server platform 4 processors share the available memory bandwidth and so each processor would have on the average only ¼ of the original bandwidth. One possible way to avoid a memory bottleneck in servers is to attach memory distributed via the processors rather than centrally via the MCH.

Principle of accessing memory when memory is attached to the processors

When the memory is attached to the processors of a multiprocessor server

- each processor has its own memory controller integrated on its die,
- the entire memory address space is subdivided among all processors,
- each memory controller serves directly its own address space but relaying access requests to other parts of the address space through an available bus system to the related processor and memory controller,
- the remote memory controller will then perform the required access and forward read data to the requesting processor.

A further benefit of attaching memory channels directly to the processors is obviously the lower access time for the cores, as indicated in the Figure below.



Due to its benefits of connecting memory channels directly to the processors, i.e. integrating the memory controller onto the processor die, this design option became a major trend for attaching memory to a platform in the first half of the 2000's.

Constraints relating the max. number of memory channels while connecting DIMMs directly to the MCH or the processor(s)

If standard DIMM channels are directly connected to the MCH or the processor (via the MC) each memory channel requires a large number of copper traces on the motherboard that interconnect the DIMM socket either with the MCH socket or the processor socket, since these interconnects are parallel, 8-byte wide links. The exact number of copper traces needed depends on the memory type used.

The interconnects between the MCH and the DIMM sockets are implemented by dense copper trails on the motherboard. For direct connected memory channels electrical constraints can seriously limit the maximum number of attachable memory channels. The limitation arises basically out of the fact that the width and the distance of the copper trails from each other determine the electrical parameters and thus the temporal behavior of the interconnect, first of all its maximum transfer rate.

What happens if the number of direct attached memory channels would be doubled?

More memory channels would require more copper traces to attach to the socket, this could only be implemented with smaller and denser copper traces, then however R and C and thus τ of the copper traces would be increased.

A given memory speed grade constrains the number and width of the copper trails that can be connected to an MCH or processor socket width a given width and thus it limits the number of memory channels that may directly be connected to an MCH or a processor. In this respect the physical dimensions of the MCH or processor socket are of paramount importance since larger sockets allow obviously, more copper trails to connect whereas smaller sockets less copper trails and thus more or less memory channels.

Evolution of cache architectures

uuu. 2, 3 and 4-level cache architectures,

vvv. **per-core or shared caches,**

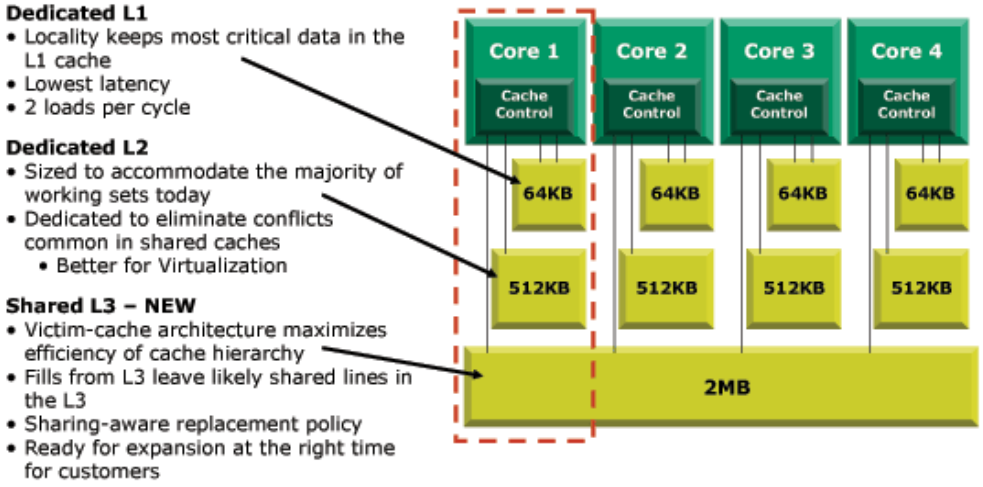
www. **unified or split caches**

A CPU cache is a hardware cache used by the central processing unit (CPU) of a computer to reduce the average cost (time or energy) to access data from the main memory. A cache is a smaller, faster memory, located closer to a processor core, which stores copies of the data from frequently used main memory locations. Most CPUs have different independent caches, including instruction and data caches, where the data cache is usually organized as a hierarchy of more cache levels (L1, L2, L3, L4, etc.).

All modern (fast) CPUs (with few specialized exceptions) have multiple levels of CPU caches. The first CPUs that used a cache had only one level of cache; unlike later level 1 caches, it was not split into L1d (for data) and L1i (for instructions). Almost all current CPUs with caches have a split L1 cache. They also have L2 caches and, for larger processors, L3 caches as well. The L2 cache is usually not split and acts as a common repository for the already split L1 cache. Every core of a multi-core processor has a dedicated L1 cache and is usually not shared between the cores. The L2 cache, and higher-level caches, may be shared between the cores. L4 cache is currently uncommon, and is generally on (a form of) dynamic random-access memory (DRAM), rather than on static random-access memory (SRAM), on a separate die or chip (exceptionally, the form, eDRAM is used for all levels of cache, down to L1). That was also the case historically with L1, while bigger chips have allowed integration of it and generally all cache levels, with the possible exception of the last level. Each extra level of cache tends to be bigger and optimized differently.

L1 is the fastest and smallest, L2 has slightly more latency but is larger, and L3 holds data that is shared among all cores in the processor (and is even larger and even slower).

Level 4 on-package cache was introduced by Intel starting with their Haswell microarchitecture. The level 4 cache uses, embedded DRAM (eDRAM), on the same package, as the Intel's integrated GPU. This cache allows for memory to be shared dynamically between the on-die GPU and CPU, and serves as a victim cache to the CPU's L3 cache.



A split cache is a cache that consists of two physically separate parts, where one part, called the instruction cache, is dedicated for holding instructions and the other, called the data cache, is dedicated for holding data (i.e., instruction memory operands). Both of the instruction cache and data cache are logically considered to be a single cache, described as a split cache, because both are hardware-managed caches for the same physical address space at the same level of the memory hierarchy. Instruction fetch requests are handled only by the instruction cache

and memory operand read and write requests are handled only by the data cache. A cache that is not split is called a unified cache.

The Harvard vs. von Neumann architecture distinction originally applies to main memory. However, most modern computer systems implement the modified Harvard architecture whereby the L1 cache implements the Harvard architecture and the rest of the memory hierarchy implements the von Neumann architecture. Therefore, in modern systems, the Harvard vs. von Neumann distinction mostly applies to the L1 cache design. That's why the split cache design is also called the Harvard cache design and the unified cache design is also called von Neumann.

The split design enables us to place the instruction cache close to the instruction fetch unit and the data cache close to the memory unit, thereby simultaneously reducing the latencies of both. This is the primary advantage of the split design over the unified design. Another advantage of the split design is that it allows instruction and data accesses to occur in parallel without contention. Essentially, a split cache can have double the bandwidth of a unified cache. This improves performance in pipelined processors because instruction and data accesses can occur in the same cycle in different stages of the pipeline.

Methods for power management

xxx. **design space,**

yyy. **dynamic and static leakage,**

zzz. clock gating,

aaaa. **power gating,**

bbbb. **DVFS, AVFS,**

cccc. **P-state management,**

dddd. **Utilizing the power headroom of processor packages to raise performance**

Power consumption of processors

Dynamic power consumption (Dynamic dissipation)

Power consumption of active transistors.
Caused by loading the dynamic capacitor (C_{dyn}) of the processor and discharging it per clock cycle.

Dynamic power consumption: D_s

D_d = const. * $C_{dyn} * V_{CC}^2 * f_c$

with V_{cc}: Core voltage
f_c: Clock frequency

Static power consumption (Static dissipation)

Power consumption of idle transistors.
Caused by the leakage current (I_{leak}) flowing through the closed transistors.

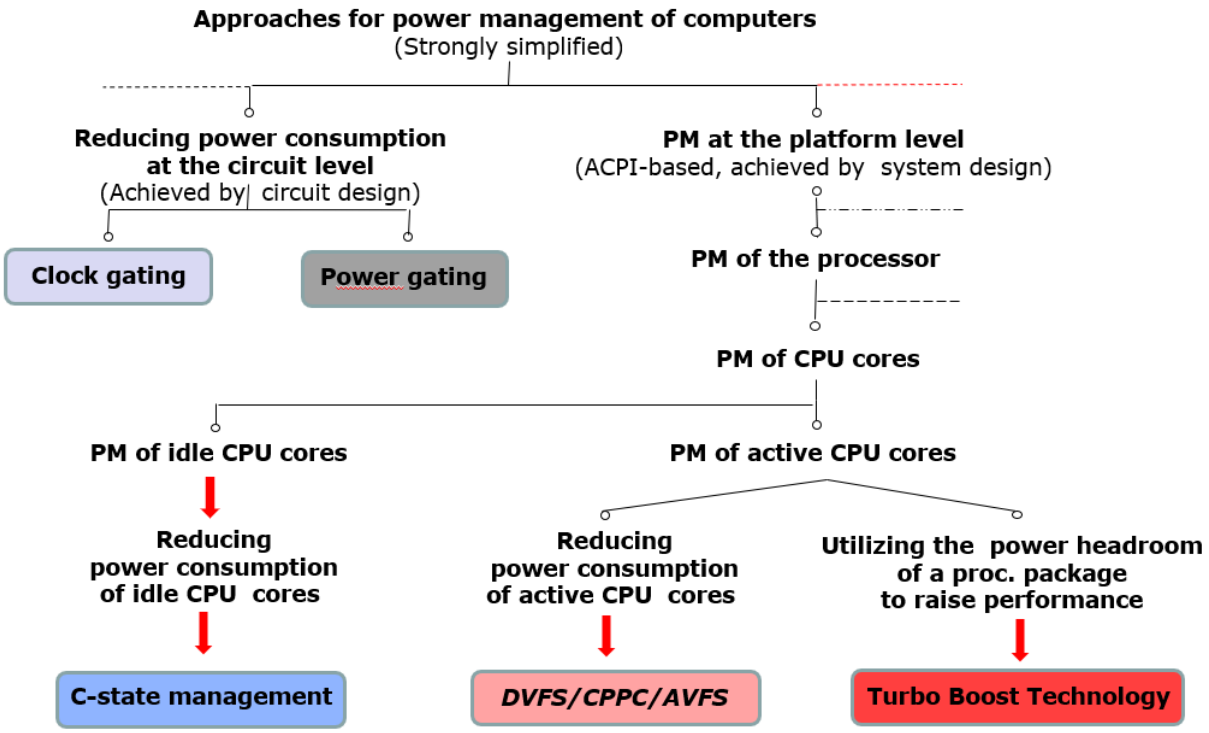
Static power consumption: D_s

$D_s = V_{CC} * I_{leak}$

Total power consumption = Dynamic power consumption + Static power consumption =

= $D_d + D_s$ = const.* $C_{dyn} * f_c * V_{CC}^2 + V_{CC} * I_{leak}$

TDP (Thermal Design Power) is the power consumption of a processor (package) specified in datasheets, given usually in W. It can be interpreted as the maximum power consumed by the processor while running realistic, power intensive applications. It serves as a reference value for designing the cooling system of the platform.



Clock gating is a technique used to reduce dynamic power by disabling clocking to currently not needed circuits. Clocking is disabled by inserting AND gates (i.e. controlled on/off switches) that are called clock gaters into the clock distribution network, as indicated below.

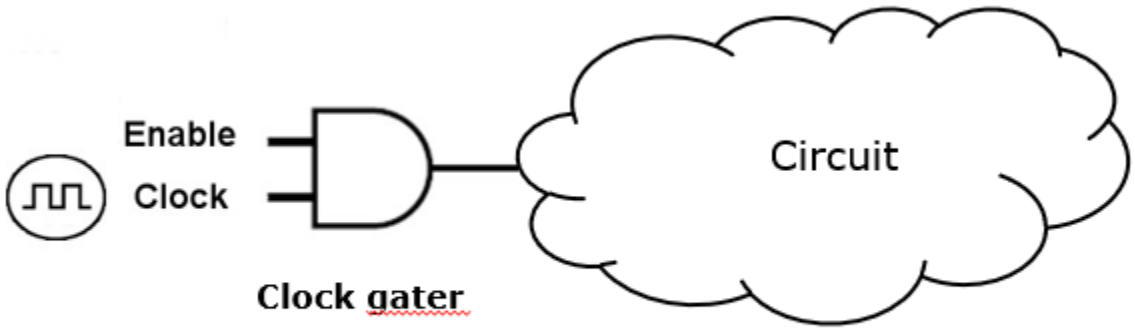
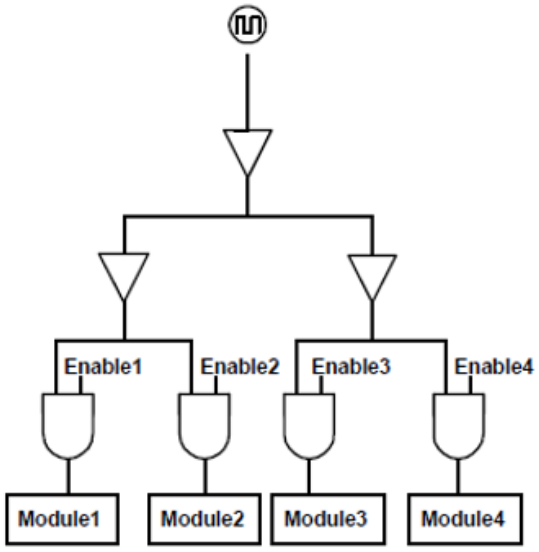
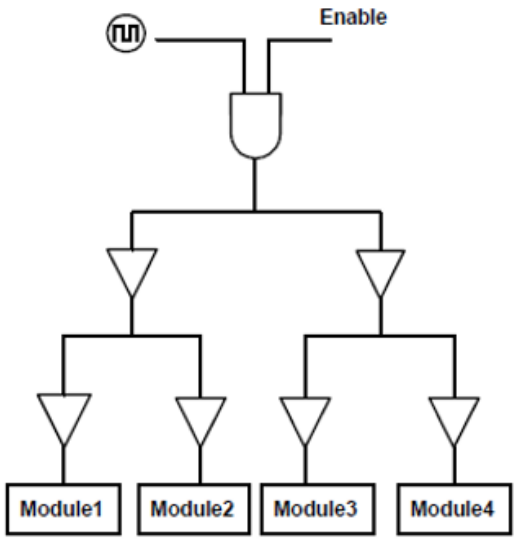


Figure: Principle of clock gating

Clock gating can be applied either in a fine-grained or coarse-grained fashion, as the next Figure shows.



Fine-grained clock gating

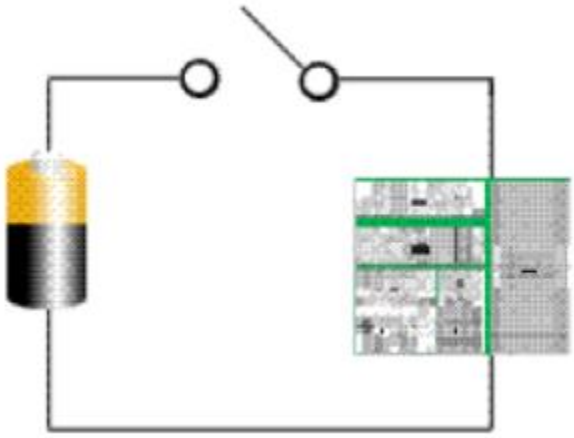


Coarse-grained clock gating

Fine-grained clock gating allows to save power in different parts of circuits whereas coarse-grained clock gating is used to disable clocking of currently idle units or parts of them (e.g. an FP unit as long as only FX instructions are processed.

Clock gating was introduced in the second half of the 1990s, designated at that time as conditional clocking. Soon clock gating became widely used, e.g. in Intel's Pentium 4 (2000) or Pentium M (Banias) (2003). Recently, clock-gating is a pervasive technique used in processors.

Power gating switches off idle units on the die (like a core) from the power supply by means of power transistors (sleep transistors) to eliminate both dynamic and static power consumption caused by leakage currents.



Clock gating eliminates only the dynamic part of power consumption whereas power gating eliminates both the dynamic and static part of power consumption of a unit.

In a multicore processor power gating allows to switch off individual, not used cores.

Obviously, before a core can be powered off

- its (write back) caches need to be flushed (i.e. modified cache lines need to be written back into the next level cache if available else into the main memory)

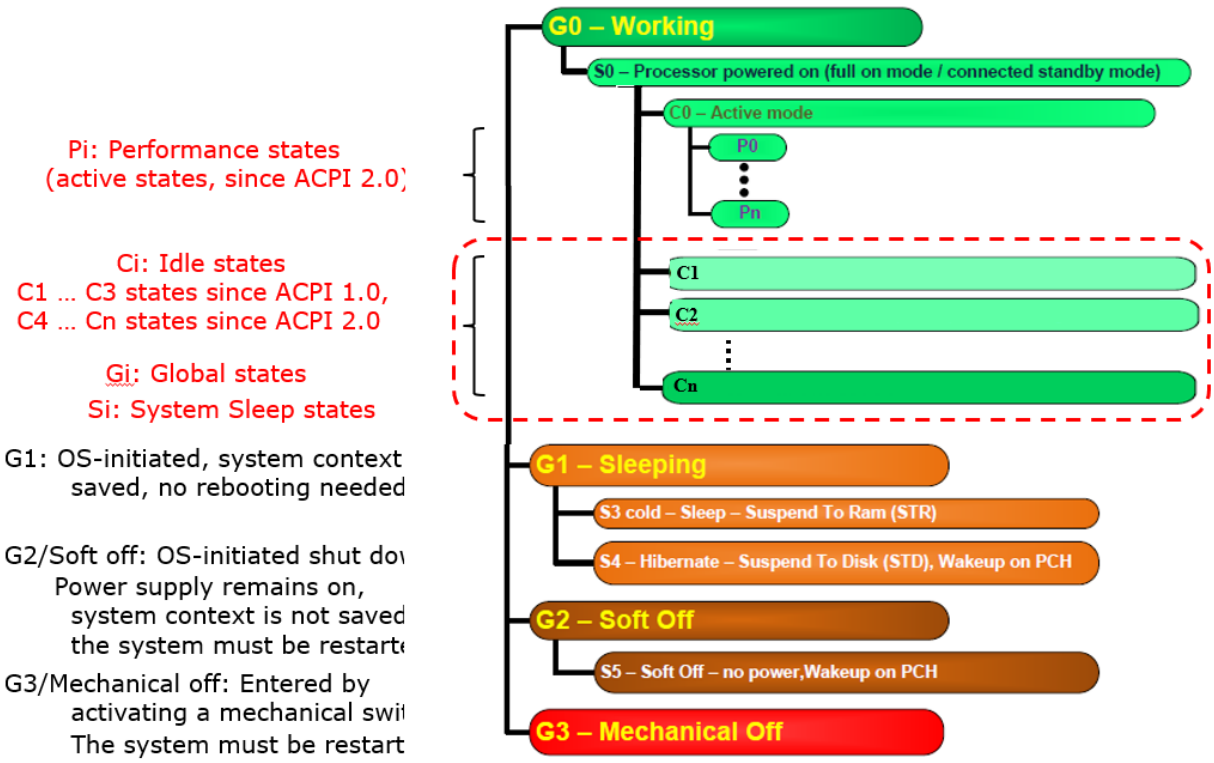
and

- its execution state needs to be preserved (either in the main memory, in the next level cache (if existent) or in an SRAM that has an independent power supply).

Power gating as a precondition for implementing the Turbo Boost technology

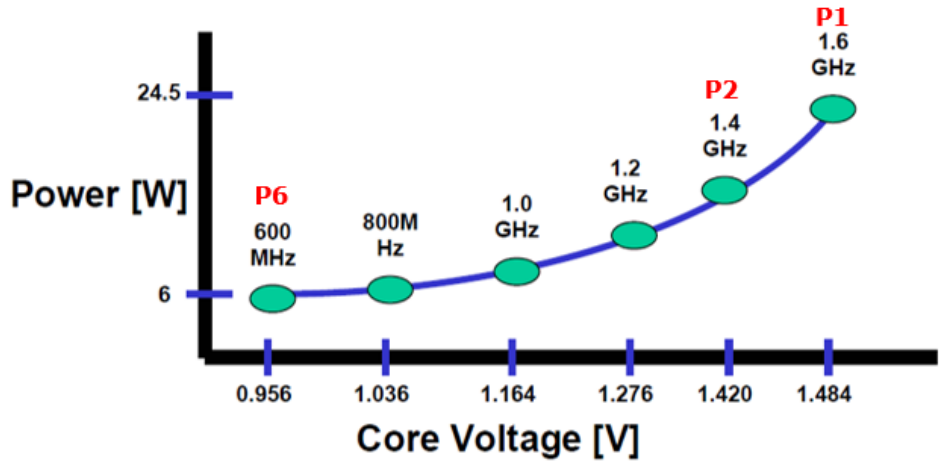
The Turbo Boost technology needs a high enough power headroom to be efficient. Power gating eliminates both static and dynamic dissipation of idle cores, in this way it largely increases the power headroom of lightly loaded or idle processors. Thus, power gating can be considered as a precondition for the implementation of Turbo Boost technology since clock gating alone does not provide enough headroom for a noteworthy increase of the clock frequency.

Integrated voltage regulators were introduced in Intel's Haswell (2013) and Broadwell (2014) lines (under the name of FIVR (Fully Integrated Voltage Regulator). They allow to switch off units individually so they make power gating superfluous. As integrated voltage regulators caused increased dissipation and thus reduced clock frequency, in their subsequent Skylake line Intel omitted integrated voltage regulators and reintroduced power gating. Then in 2019 in their Ice Lake line Intel brought back integrated voltage regulators.



Higher numbered Ci states provide higher power savings along with longer enter and exit latencies. In idle states the processor is powered on but does not execute instructions. This allows to reduce power consumption by various techniques, such as clock gating, power gating, reducing or switching off the supply voltage of actually idle cores or caches etc. Idle states are characterized by two key parameters: the power reduction they provide and the time needed to enter and exit to and from an idle state. E.g. the C6 state has a significantly lower power consumption (0.3 W) than the C4 state (1.7 W) providing headroom for introducing a forerunner of the Turbo Boost technology. C-state management can be controlled by the South Bridge or by the Power Control Unit (more advanced).

Dynamic Voltage and Frequency Scaling: the idea of DVFS is to scale down the clock frequency (fc) as far as possible, that is as far as the run time of the actual workload does not lengthen noticeably and lower core voltage (Vcc) to the reduced fc value needed. This principle will be applied in subsequent time windows of, say 100 ms to reduce dynamic power consumption. DVFS is implemented based on P-states (Performance states). P-states are possible operating points (Vcc and fc pairs) of a core while running under DVFS.



Example: Operating points of Intel's Pentium M processor (~2003)

Now it is the task of the OS to determine how far fc and Vcc can be reduced without causing a noticeable lengthening of the run time of the actual workload. To do this the OS will assess the utilization rate of the core in subsequent time windows (of. e.g. 100 ms) while running the actual workload and will choose the P-states that is needed to perform this workload without lengthening the run time. E.g. if a core runs recently at 2 GHz but it is utilized only by 50 %, each second clock cycle is wasted and the clock frequency can be reduced to 1 GHz without worsening the user experience. Then the OS selects the optimal working point from a given list of possible working points (fc, Vcc values) by looking for the P-state with a core frequency of at least 1 GHz.

Subsequently, the OS will inform the processor about the P-state requested and typically, the PCU (Power Control Unit) of the CPU will accomplish the required P-state transition.

There are two improvements of DVFS:

- Collaborative Processor Performance Control

  It passes over the control of DVFS from the OS to the PCU, to get a faster and finer frequency and voltage scaling.

- AVFS (Adaptive Voltage and Frequency Scaling)

  It allows a more efficient voltage and frequency scaling by replacing the

worst case design approach by an adaptive approach.

In DVFS, the voltage levels of the targeted power domains are scaled in fixed discrete voltage steps. Frequency-based voltage tables typically determine the voltage levels. It is an open-loop system with large margins built in, and therefore the power reduction is not optimal. On the other hand, AVFS deploys closed-loop voltage scaling and is compensated for variations in e.g. temperature using dedicated circuitry (typically analog in nature) that constantly monitors performance and provides active feedback. Although the control is more complex, the payoff in terms of power reduction is higher.

Intel introduced the **Turbo Boost technology** in 2008, designated also as the 1. gen. Turbo Boost technology. Also AMD implemented the Turbo Boost technology, called the Turbo Core technology, nevertheless with a delay of at least two years.

If a processor dissipates less than its TDP value, a power headroom arises. Turbo Boost technology converts power headroom to higher performance by raising the clock frequency of the active cores.

In case when

- the actual workload requests the highest performance state (P0),
- there is a power headroom, that is the actual power consumption is less than the TDP value,
- the actual current is less than a given limit and
- also the die temperature is below a given limit

the PCU converts available power headroom into higher performance by raising clock rate and core voltage of the active cores up to a given limit.

If the discussed conditions for activating the Turbo Boost technology are met, the PCU automatically steps up core frequency in a closed loop by one bin (133.33 MHz for the Nehalem family) in each step as long as it reaches the max. boost ratio.

2nd generation: The concept utilizes the real temperature response of the processor to power changes (due to the "thermal mass of the "cooler) and raises processor performance in a time burst until the package temperature slowly rises to its allowed limit.

The Turbo Boost 2.0 technology raises the base frequency of all active cores in case of a light workload as far as the TDP allows it. By contrast the **Turbo Boost 3.0** technology aims at increasing the performance of single threaded applications. To achieve this, during processor testing Intel determines the max. clock speeds of all cores and arranges the cores into a list according to their max clock speed. For single threaded workloads the fastest core will be activated.

The roadmap of AMD Zen release

eeee. Zen, Zen+ architecture,

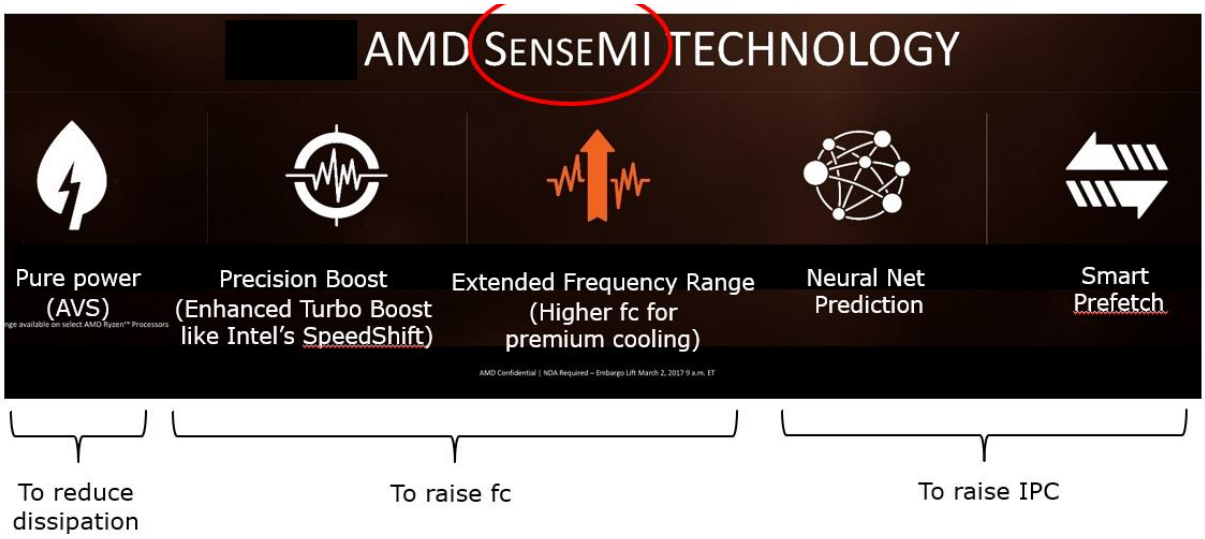
ffff. core complexes,

gggg. Zeppelin, Threadripper, and Epyc lines,

hhhh. infinity fabric connections

AMD's effort to design the Zen core: Up to 300 engineers were working on the Zen core spending over two million working hours. Aim: optimizing the power/frequency curve.

Innovations introduced by the Zen family



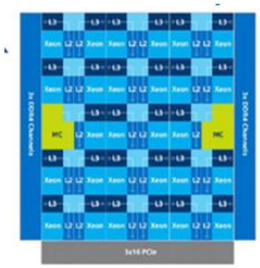
Design paradigms for segmenting multicore processors

Monolithic implementation

All cores are implemented on the same die

Example

Intel's Skylake-SP (2017) *[2]*

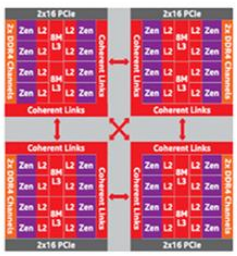


Up to 28 cores

Multi-Chip-Module (MCM)

Cores are implemented on a number of dies, they are properly interconnected and mounted into the same package

AMD's Epyc (2017) *[2], [3]*



Up to 4x8 cores

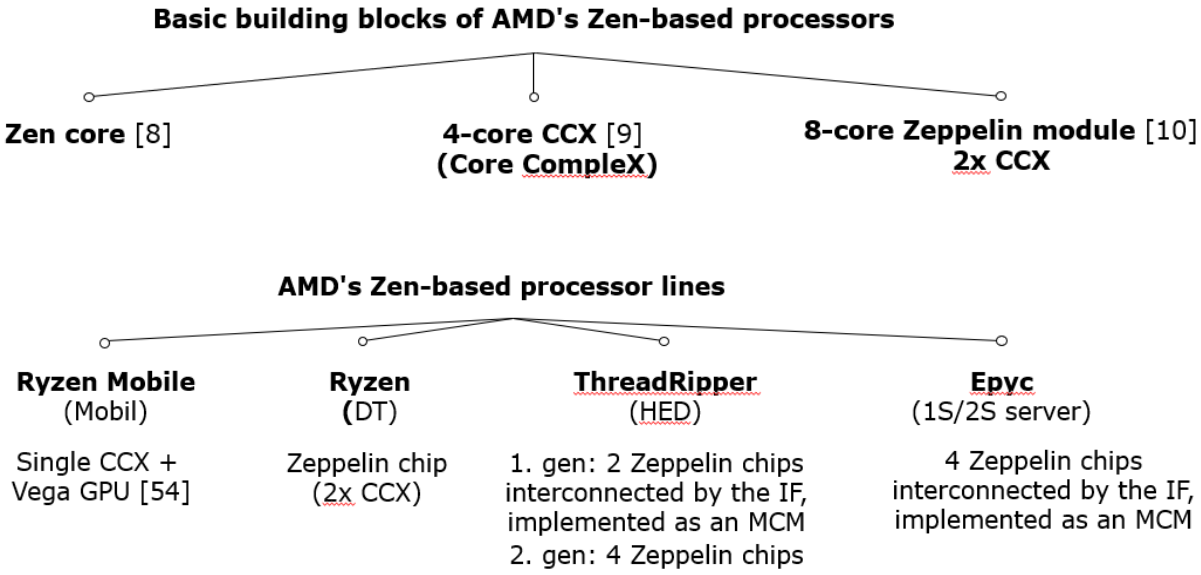
Pros and cons of a modular processor design

Pros

- It is more economical to manufacture large core count, and therefore large dies if segmented into a number of smaller dies
- Memory channels and I/O become linearly scaled with the die count (by proper design)
- It becomes possible to design processors for different market segments by including different numbers of the same die

Cons

- High die-to-die transfer latencies that degrade performance



With Zen AMD departed from the Bulldozer design style cramming two integer parts and an FP part into a module and declaring the module as being dual cores and followed the regular way of building up cores.

Innovations

a) Neural net improved branch predictor (Part of SenseMI)

b) Smart prefetch (Part of SenseMI) - Anticipates the location of future data accesses and prefetches the data into the local cache

c) Large Micro-Op cache (2K instructions)

Enhancements

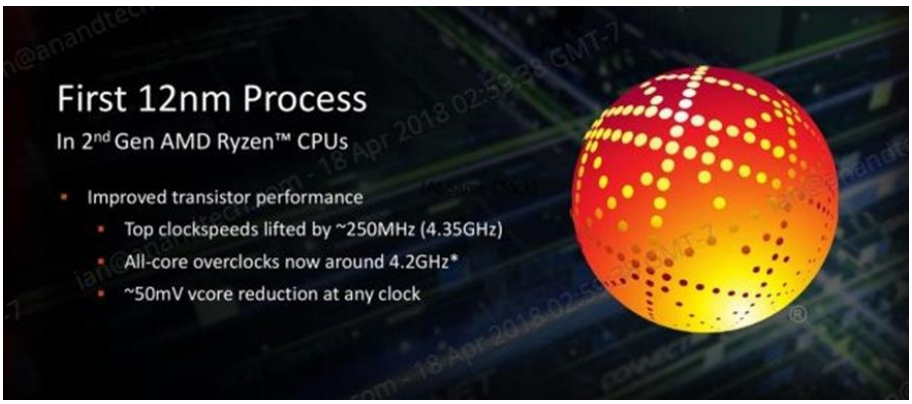
d) Wider μop dispatch

- Six-issue FX execution (up from 4)
- Quad-issue FP execution (up from 3)

e) Further improvements of the microarchitecture

Power management: see next topic

Zen+ core



While employing the 12LP technology AMD implemented the refreshed Ryzen DT line on the same die size (213 mm2) and made use of the same number of transistors (4.8 billion) as in their original design. As a consequence, with reducing the feature size the dark silicon (unused silicon) area has grown. Since dark silicon acts as a thermal buffer between high-consumption parts of the die it improves the thermal behavior of the die.

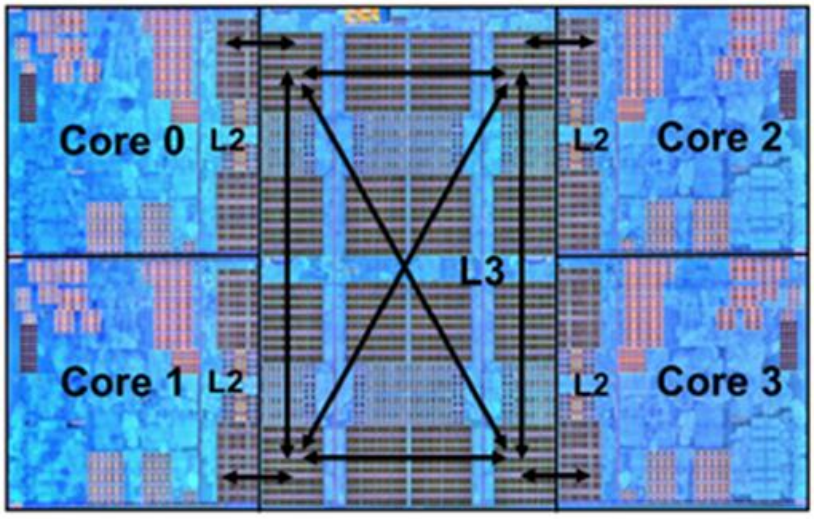
The new Ryzen 2000 series processors draw about 11 % less power than the Ryzen 1000 series models at the same clock frequency. It results about 15 % more performance at the same power.

The Zen2 core optimized energy consumption even further with its 7nm technology.

Core complexes

The 4-bit CCX (CPU-Complex) is the basic building block of Zen-based processor lines.

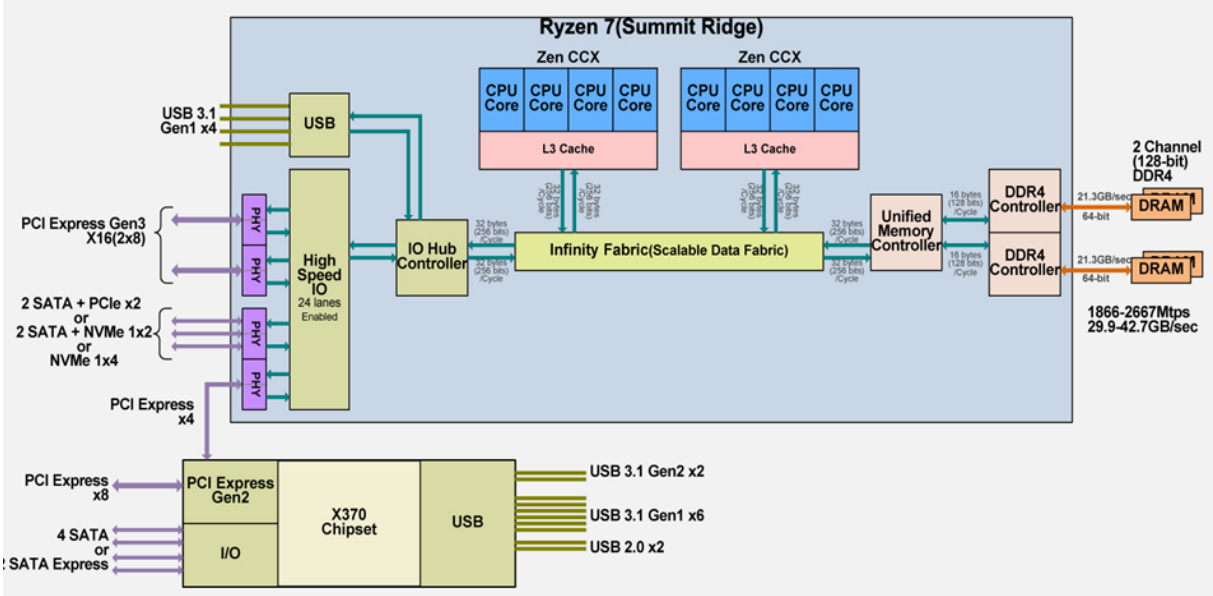
Each core has a private 512 kB L2 cache whereas all 4 cores share an 8 MB L3 cache that is split into 4 slices (see Figure) by low-order address interleave. It includes 1.4 billion transistors implemented on a die area of 44 mm2.



The L3 cache is mostly exclusive to the L2 cache, it is actually a victim cache for the L2 cache. Due to the internal crossbar interconnection (see previous Figure), each core can access each L3 cache segment with the same average latency.

Zeppelin

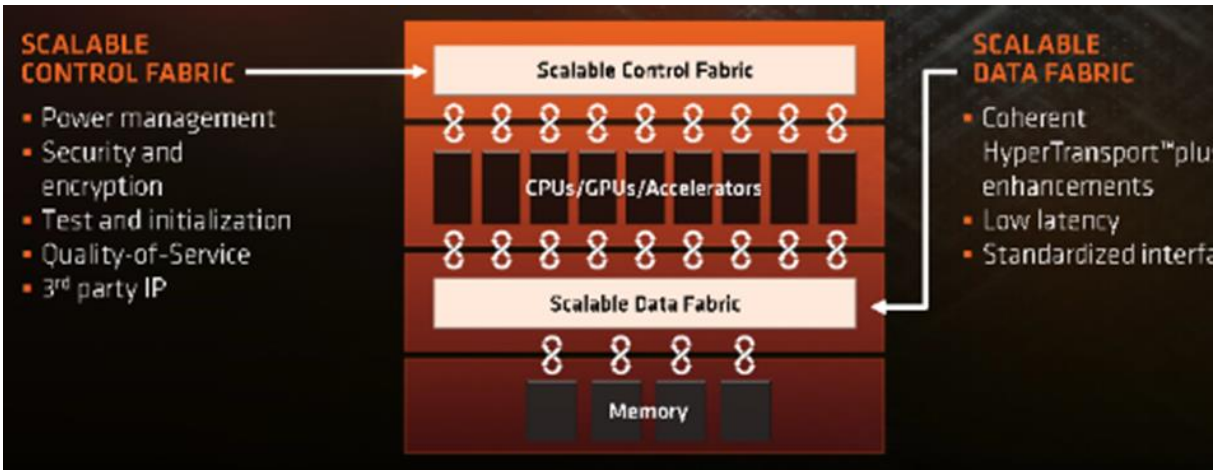
It is based on two CCX blocks connected together by the Infinity Fabric, as seen below.



AMD designed the Zeppelin module for the server market, but they utilize it also as a modular building block for their consumer (Ryzen) and HED (Threadripper) lines.

Infinity Fabric

The IF is AMD's recent interconnection fabric used for linking system units in a cache coherent way and provides also system wide control functions, such as power management or system security. It is the evolution of the HyperTransport bus, it is similar to Intel's QPI or UPI interconnection technology or ARM's CCN (Cache Coherent Network). The Infinity Fabric consists of two key parts: a Scalable Control Fabric, and a Scalable Data Fabric, as seen in the next Figure.



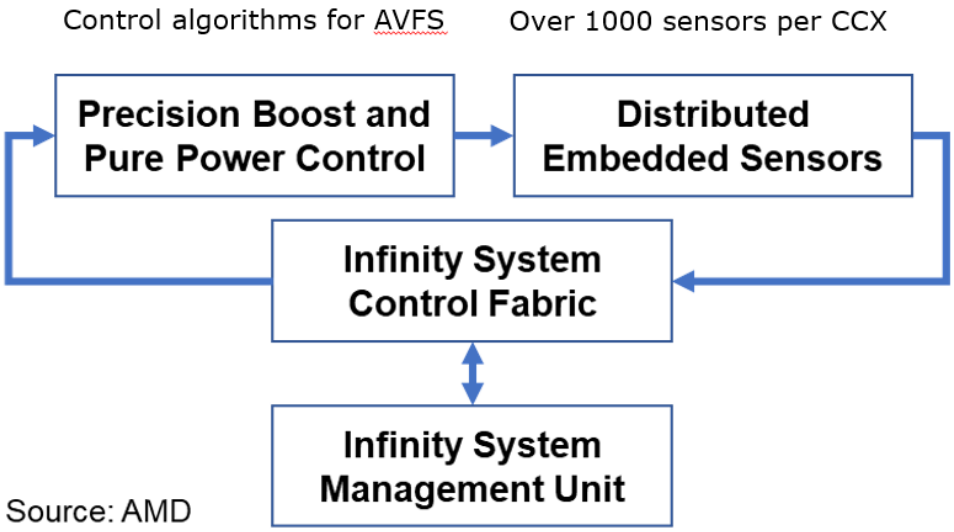
The Scalable Control Fabric is responsible for the system wide

- Power management
- Security and Encryption
- Test and initialization

- Quality of Service

The Scalable Control Fabric incorporates central control elements, and remote embedded elements dispersed in all blocks of the SoC.

The remote elements are mainly sensors monitoring temperature, speed and voltage, their data is fed into the central control element and allows to optimize power management in a closed control loop.



The system may be customized by choosing the proper balance between performance and power consumption

The Scalable Data Fabric provides a high performance cache coherent data pathway for interconnecting all system blocks. It is an evolution of the HyperTransport bus.

It interconnects

- CCX building blocks within a die
- different dies within a socket
- both sockets of a 2S server or even further sockets.

ThreadRipper HED line

AMD launched until now two generations of ThreadRipper processors:

- the 1. generation in 2017 and
- the 2. generation in 2018.

.

1st generation

HED (or HEDT) meaning High-End Desktop.

2 Zeppelin chips interconnected by the IF, implemented as an MCM.

AMD reuses the packaging of the EPYC line including the cooling solution:

- they put 4 chips into the package whereas two chips are actually dummies,
- they use the TR4 socket with 4094 contacts.

Advanced technologies implemented in the ThreadRipper line

1) Pure Power technology (actually AVS)

2) Precision Boost technology (actually Turbo Boost)

3) XFR (eXtreme Frequency Range) technology (actually configurable TDP)

See topic 35.

Creator mode - Game mode of the ThreadRipper line

- The Creator mode is the natural configuration of ThreadRipper processors with all cores enabled.
- In Gaming mode only one die is enabled, so only half of the cores are active. This can be beneficial while running older games that are not prepared to use more than 8 cores, in this case long Die-to-Die latencies could significantly reduce performance.

2nd generation

Launched in 2018. It is based on the Zen+ μarchitecture, implemented on 12nm technology.

- the low core count models have two active dies with 8 or 6 cores each, with both dies having direct access to memory and I/O, similar to the 1. gen. models, whereas
- the high core count models incorporate four active dies with 8 or 6 cores each.
- In the latter case
    - two dies have direct access to memory and I/O whereas
    - the other two dies have no direct access to memory or I/O, only via the direct-access dies, obviously with higher latency.
- The new high core count models have more compute resources and the same memory and I/O resources, as the low core count models, obviously they target users with compute bound applications, like simulations or physics calculations.
- The above asymmetric kind of resource usage can be designated as being bi-modal.

Implications of the bi-modal layout of the high-core ThreadRipper models for the scheduler

- For obvious reasons the scheduler will assign tasks first to cores that are directly attached to the memory and I/O before loading other cores. Nevertheless, to prevent overheating, the scheduler will likely avoid

loading all cores of the direct connected dies before allocating tasks to not directly connected dies, e.g. after loading 12 or 14 cores out of 16 it begins already loading cores of not directly connected dies.

Enhancements of the 2. gen. ThreadRipper line vs. the 1. gen. line

a) Higher core counts (24 and 32 core models) vs. up to 16-core models

b) Precision Boost 2 (decrease of clock frequencies for higher number of active cores in Turbo mode)

c) Precision Boost Overdrive (overriding factory settings)

d) StoreMe (unified storage)

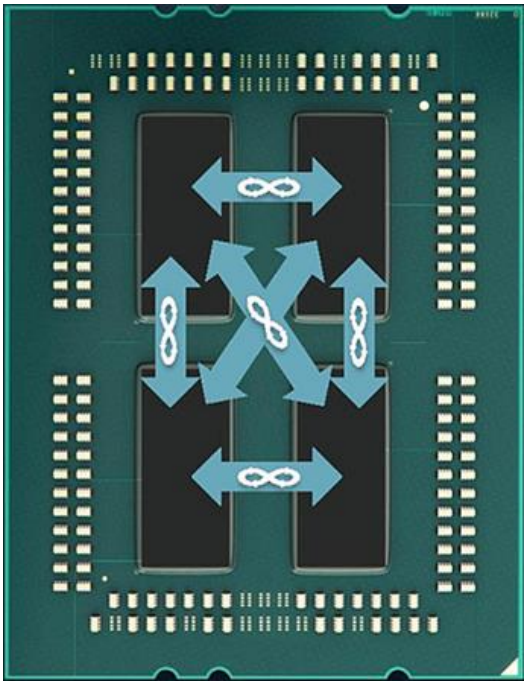
AMD unveiled until now two generations of **Epyc** processor lines:

- the 1. generation, called Naples line, launched in 2017 and
- the 2. generation, called Rome line, introduced in 2018.

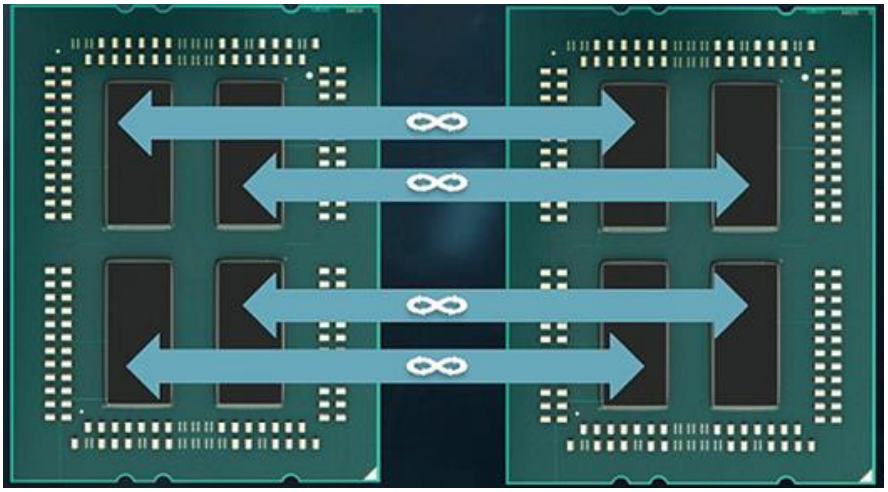
1st generation

Launched in 2017 based on the Zen core, manufactured on 14 nm technology. It covers 1S and 2S servers. The reason why AMD focuses on up to 2S servers is that these servers cover recently over 90 % of the server market.

A 1S EPYC server processor is built up of 4 Zeppelin dies interconnected by a crossbar Infinity Fabric switch and implemented as an MCM (Multi-chip_Module), as shown in the Figure below.



A 2S EPYC server processor is built up 2 EPYC processors that are interconnected by the Infinity Fabric (IF), as shown in the Figure.



An Epic processor has 8 x 16 serialization-deserialization circuits (called SERDES) that can be used either for implementing the interconnection lanes between two sockets of a 2S server processor or PCIe 3.0 lanes.

Additional features of EPYC-7000-series processors

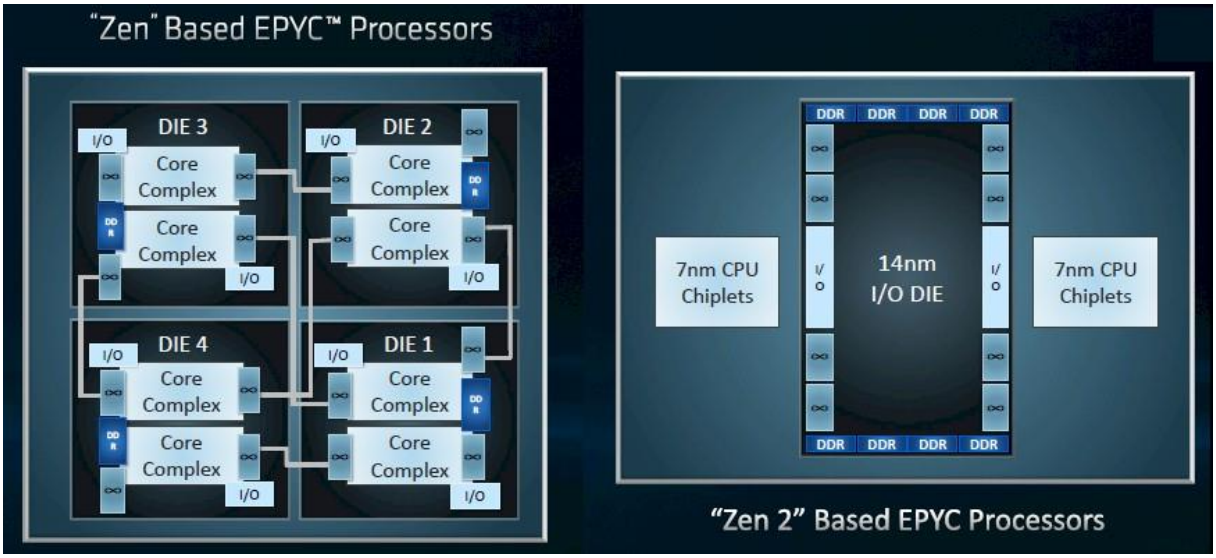
a) Workload aware power management - recognizes workloads which are utilizing all the cores for not so latency sensitive tasks and lowers the core frequencies

b) Per core frequency and voltage scaling

c) Integrated secure processor - dedicated security subsystem, provides cryptographic functionality

2nd generation

Introduced in 2018. It is based on the Zen 2 core and is fabricated on 7 nm technology (TSMC). It covers 1S and 2S servers. Novel layout based on an I/O die (manufactured on 14 nm) and 8 core chiplet, each with 8 Zen 2 cores.



Key features of the 2. gen. Rome Epyc server line

- Eight channel DDR4 memory controller that provides equal access latency to all chiplets.
- Up to 4 TB DDR4 memory.
- 128 PCIe 4.0 lanes per socket, which is the first PCIe 4.0 implementation in processors.
- Greatly improved Infinity Fabric speed.
- Ability to connect GPUs via the I/O chip and Infinity Fabric protocol without using PCIe lanes.
- Socket compatibility with Naples processors.

The Ryzen desktop line

iiii. pure power technology,

jjjj. precision boost,

kkkk. XFR,

llll. closed loop control range "triangle",

mmmm. StoreMI technology

The Ryzen desktop line is called also the 1. gen. Ryzen desktop line. Designated also as the Summit Ridge line.

Launched in 2017 (Ryzen 7)

2017 (Ryzen 5)

2017 (Ryzen 3)

Ryzen processors include a single Zeppelin die, called also the Ryzen die, which is built up basically of two CCX complexes interconnected by the IF (Infinity Fabric). Models of the Ryzen desktop line are unlocked. They have AM4 socket.

Until introducing the Ryzen desktop line desktops provided up to 4 cores. The Ryzen desktop line provides up to 8 cores. Subsequently, also Intel raised the max. core count to 6 in their 7. gen. Kaby Lake Refresh desktop line in Q3 2017 to cope with AMD's Ryzen line for MT workloads.

Pure power technology

It is basically an AVS (Adaptive Voltage Scaling) technology. AVS is a kind of AVFS (Adaptive Voltage and Frequency Scaling) technology. AVFS is an enhancement of the DVFS (Dynamic Voltage and Frequency Scaling).

Key points of the implementation of AMD's Pure Power

a) Real time monitoring of temperature, speed and voltage by sensors,

b) Adaptive closed-loop control

c) Per-core scaling of core frequency and voltage

a) **Real time monitoring of temperature, speed and voltage by sensors**

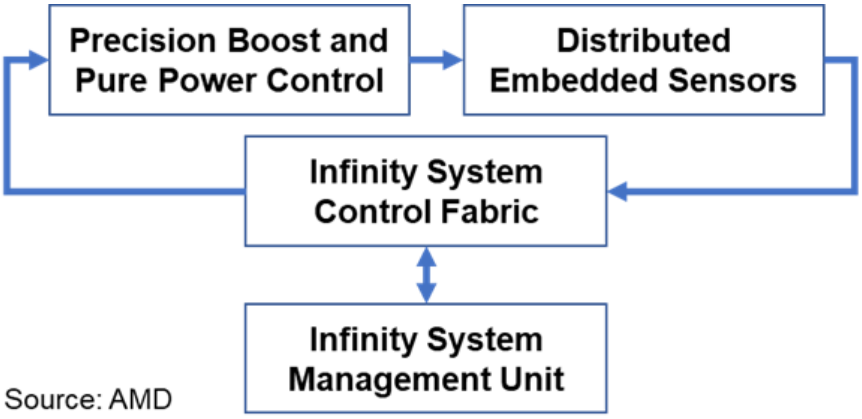
1000+ embedded sensors per CCX monitor temperature, speed and voltage per ms.

Critical path monitors overlook processor speed (clock frequency), they allow to reduce core voltage in a controlled way as far as possible whereby a given clock frequency remains maintained.

Droops are rapid spikes in the voltage supply, they need to be addressed to avoid erroneous operations (not detailed here).

b) **Adaptive closed-loop control**

There is closed loop control for implementing AVS.



An Infinity System Management Unit (based on the SMU microcontroller) is used for system settings according to user requirements. The SMU (System Management Unit) microcontroller allows to choose the proper balance of performance and power consumption.

c) **Per-core scaling of core frequency and voltage**

Each core has

- a digital Low Drop-Out (LDO) Voltage Regulator (VR) and
- a digital frequency synthesizer (DFS)

to independently vary core frequency and voltage for reducing power consumption.

The voltage regulators operate also as power gates.

Two stage voltage regulation (similar to Intel's two stage FIVR implementation):

- 1. stage: on the mainboard, it reduces voltage from 12 V to 0.9 V,
- 2. stage: separate on-die VRs for each core.

The **2nd generation Ryzen** desktop line was code-named as the Pinnacle Ridge line.

Launched in 2018 as the Ryzen 7 2700X/2700 and

Ryzen 5 2600X/2600 models.

The line is manufactured on the 12 nm technology (by GlobalFoundries). These processors are unlocked. They have AM4 socket. The line is accompanied by new motherboards (400-series) but they remained compatible with the previous 300-series motherboards. The processors come bundled with AMD's Wraith Spire cooler.

Enhancement vs the previous Ryzen line:

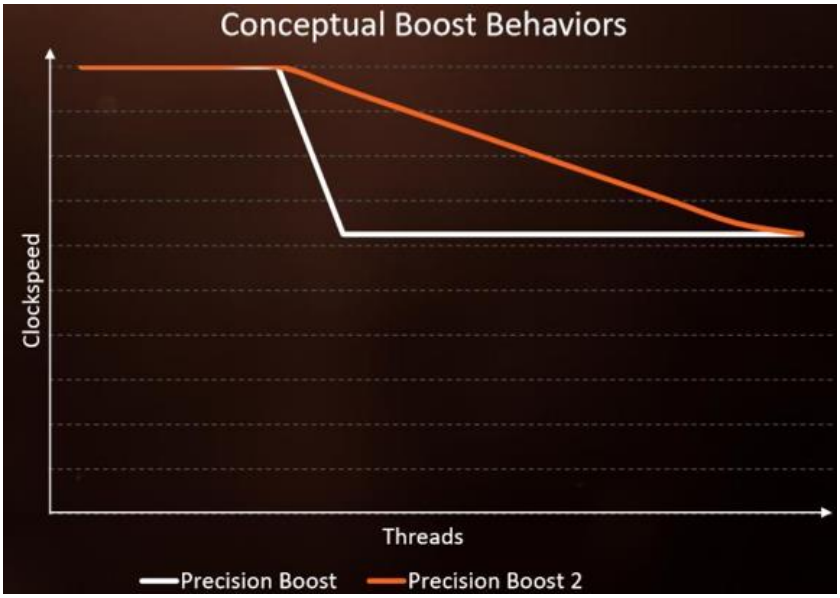
a) Precision Boost 2 technology

b) Precision Boost Overdrive (PBO)

c) XFR2 (Extended Frequency Range) technology

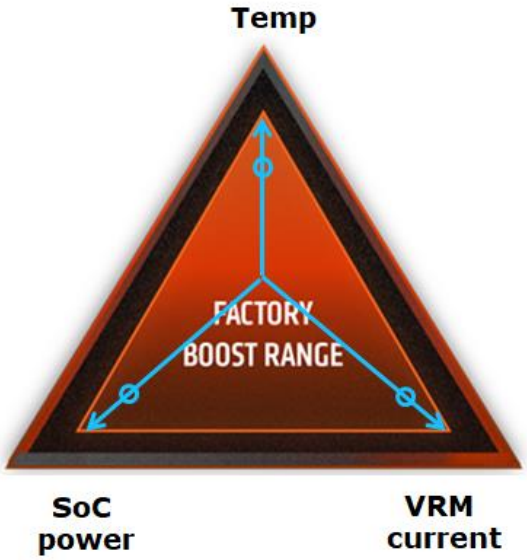
d) StoreMI technology

a) **Precision Boost 2 technology**



Precision Boost 2 technology raises the clock frequency for any active core count as much as given limits (defined by the factory boost settings) allow it. The limiting factors are

- SoC power ("PPT (Total Platform Limit"): the maximum amount of power the CPU can draw before boost levels off (measured in watts)
- VRM current ("TDC Limit"): the maximum amount of current the motherboard is allowed to deliver to the CPU after warming up to a steady-state temperature before boost levels off
- EDC current Limit - Electrical Design Current, maximum peak current that can be delivered by the motherboard' voltage regulator (VRM) for a short time, as a spike
- Temp (°C): measured in degrees Celsius, the maximum temperature the CPU die can reach before boost levels off.

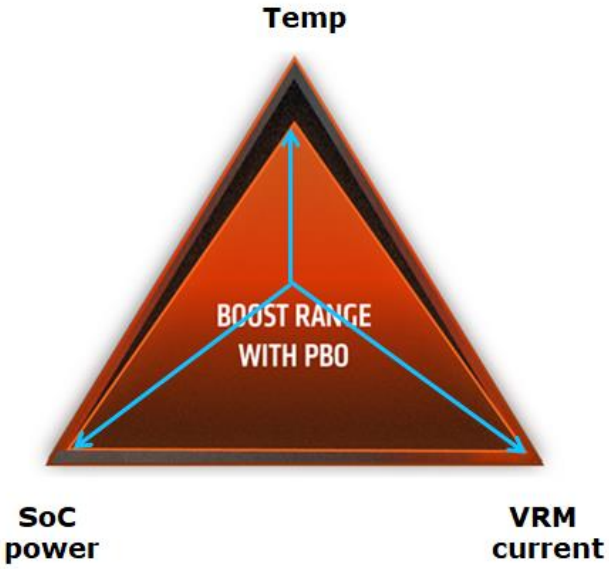


The inner triangle illustrates the factory boost range settings, the blue circles represent actual operating points of the key limiting factors. Clock frequency and core voltage of all active cores will be increased in 25 MHz steps as long as one of the limiting factors reach the factory setting.

b) **Precision Boost Overdrive (PBO)**

It is a new feature of 2. gen. Ryzen DT and 2. gen. ThreadRipper processors, aiming at increasing multithreaded performance. It requires however the use of specific chipsets and the Ryzen Master overdrive utility release 1.3 or higher. Unlike traditional overlocking while PBO is activated both Precision Boost 2 and XFR2 (eXtended Frequency Range) remains enabled.

When PBO is activated by means of the Ryzen Master overclocking utility the factory settings that determine the operation limits of Precision Boost 2 will be overwritten and the processor will operate outside the factory settings.



Note that the temperature limit with PBO remains unchanged.

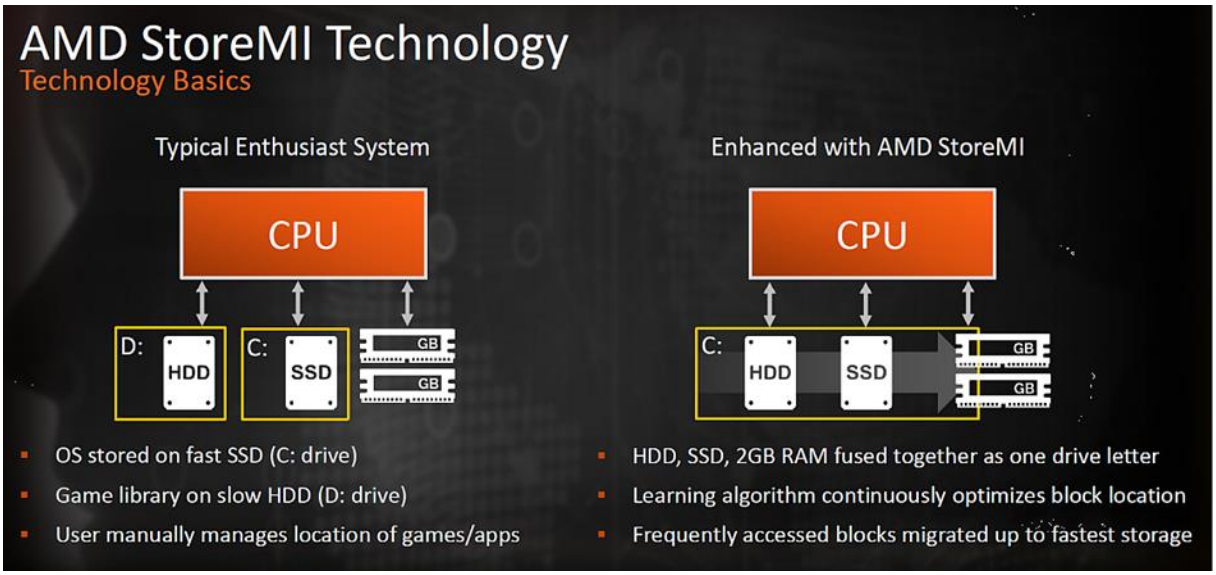
Enabling the PBO mode invalidates AMD’s warranty.

c) **XFR2 (Extended Frequency Range) technology**

In the previous generation certain processor models could raise clock frequency over the turbo boost frequency by 50 to 200 MHz if a premium cooling system was used. XFR2 allows a higher clock frequency boost assuming a premium cooling system and a CPU temperature below 60 C°.

d) **StoreMI technology**

The StoreMI technology combines fast SSD (up to 256 GB NVMe or SATA) and large capacity HDD storage into a single drive (as indicated below) and automatically moves the data accessed the most to the SSD.



The unified storage has the responsiveness of SSD, and capacity of low priced mechanical HDD. In addition up to 2GB of DRAM can be included to have a last-level cache for ultra-fast data access.

The StoreMI technology requires for 2. gen. Ryzen 2000-series DT processors and AMD 400-series or for 2. gen. ThreadRipper processors an AMD X399 chipset. The storage hierarchy is controlled by the AMD's StoreMI software utility that initially was sold for $20 but later became free of charge.

Along with the 7th generation Core line, called the Kaby Lake line, Intel introduced in 2017 their Octane memory technology and enhanced it in their subsequent 8th generation Coffee Lake line in 2018.

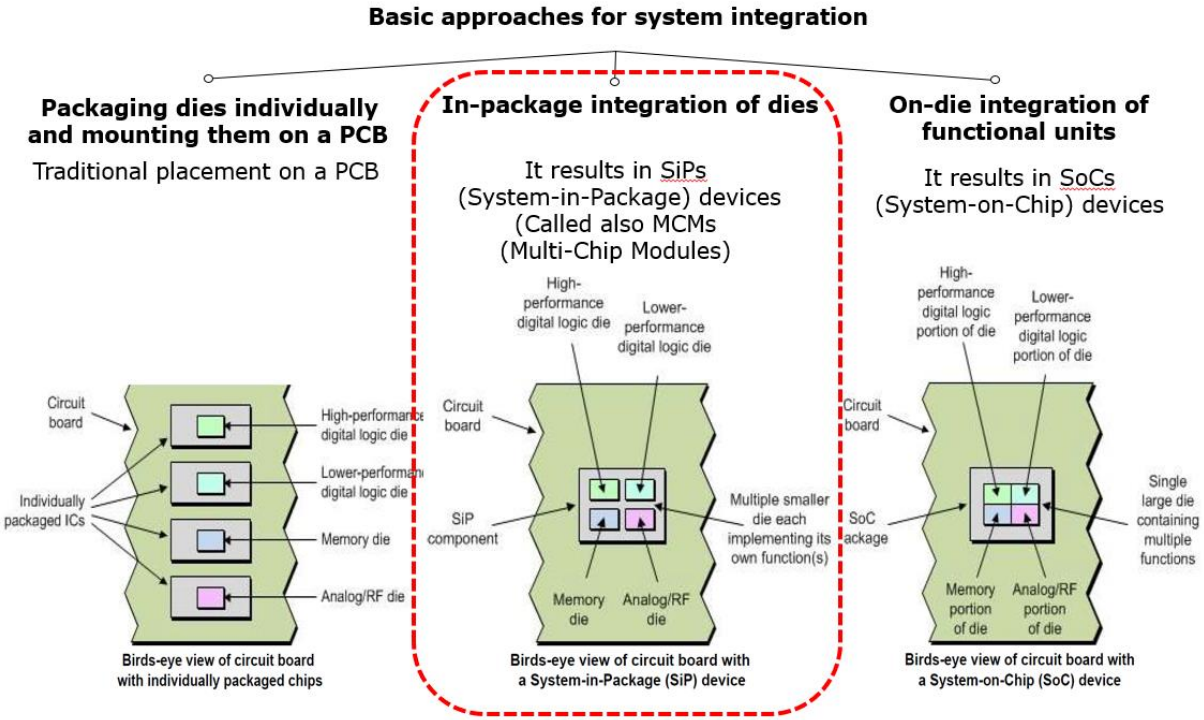
Evolution of in package integration

nnnn. **Multi-Chip Modules,**

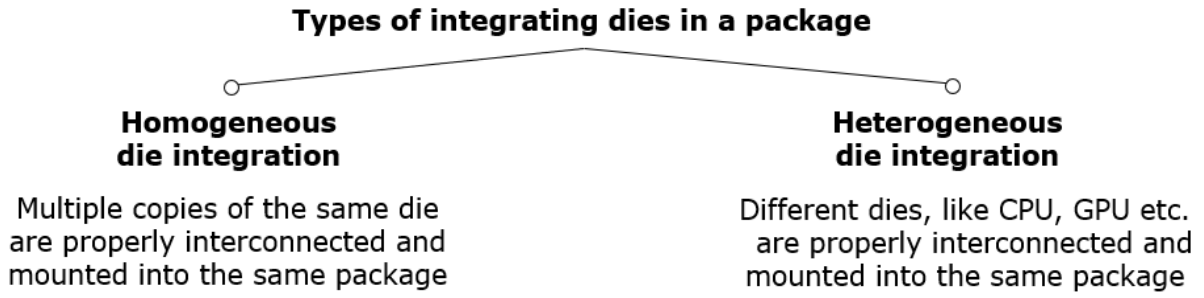
oooo. **homogeneous and heterogeneous die integration,**

pppp. **horizontal and vertical die placements,**

qqqq. **limitation: power consumption of communication fabrics**



PCB: Printed Circuit Board



There may be two main reasons why a complex IC should be implemented by multiple copies of the same dies, rather than by a single monolithic die:

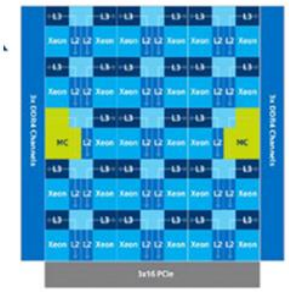
a) To shorten time-to-market by interconnecting already existing circuits rather than redesign the complex IC,

b) To reduce manufacturing costs by implementing multiple lower cost dies.

Example for a): Intel introduced quad-core processors by employing existing dual-core designs in their dual chip Xeon 5300 (Clovertown) DP processor (2006) by interconnecting already existing dual core processor designs rather than redesigning the complex IC.

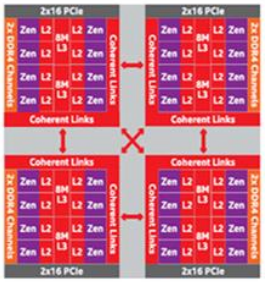
Example for b): AMD preferred to implement their 32-core EPYC server processor by four, less complex 8-core dies rather than implementing it as a 32-core monolithic die. By contrast, Intel implemented their 28-core Skylake-SP server processor as a monolithic die.

Intel's Skylake-SP (2017) *[78]*



Up to 28 cores

AMD's Epyc (2017) *[78]*

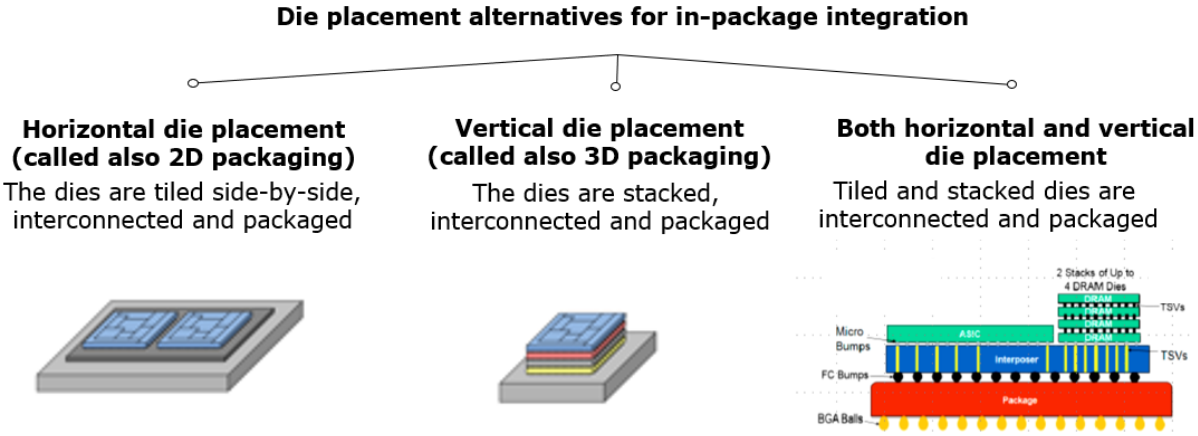


Up to 4x8 cores

Again, there may be two main reasons why a complex IC should be implemented on a number of different dies, rather than on a single monolithic die as follows:

a) To overcome transistor count limitations in case when a large transistor count can not yet be implemented cost effectively on a single die,

b) To overcome technological limitations in case when different IC technologies should be implemented on the same die such as “traditional IC technology” and e-DRAM technology.



Horizontal die placement in in-package integration is also called as 2D packaging. The dies are tiled side-by-side, interconnected and packaged.

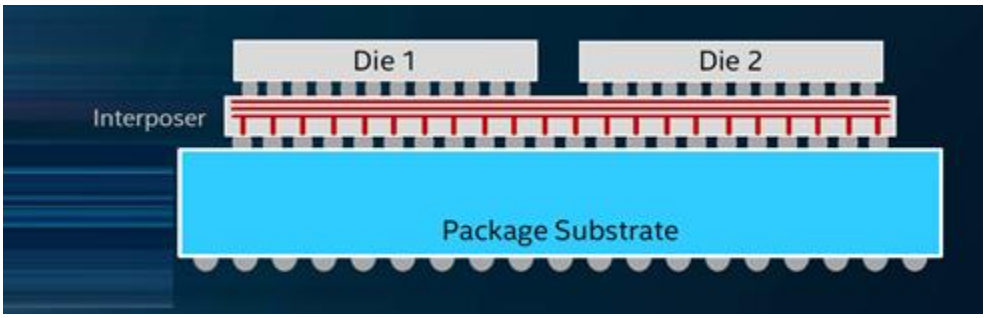
Main alternatives for horizontal die placement in in-package integration

Traditional packaging	2.5D packaging	EMIB technology (Embedded Multi-Die Interconnect Bridges)
Interconnections are made by wires through wire bonding or by conducting traces formed on a ceramic, silicon, or organic substrate made e.g. by thin film, thick film or PCB technologies	Interconnections are provided by multi-layered conducting traces in a silicon or organic interposer, serving as a redistribution layer and by TSVs (Through-Silicon-Vias)	Interconnections are provided by multi-layered conducting traces in a silicon interposer serving as a redistribution layer and by silicon bridges

a) Interconnections in **traditional packaging** may be implemented by a number of technologies, like

- by wires and using wire bonding or
- by conducting traces formed e.g. by means of
    - thin film technology
    - thick film technology
    - PCB technology

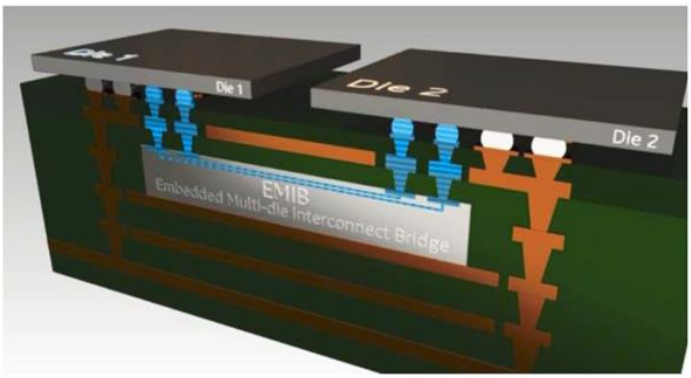
b) In case of **2.5D packaging** a thin interposer, made of silicon or an organic material is placed between the dies and the package substrate that serves as a redistribution layer (RDL).



Further on in the 2.5D technology short, vertical conducting paths are used through the interposer, called TSVs (Through-Silicon-Vias) to interconnect the dies and the redistribution layer.

Unfortunately, 2D packaging technology is expensive due to the high costs of the TSV technology and the silicon interposer.

c) **EMIB** makes use of short silicon bridges between the dies that are embedded into the package substrate.



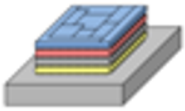
The silicon bridges provide a multi layer connection between adjacent dies.

Vertical die placement in in-package integration is also called 3D packaging. The dies are stacked, interconnected and packaged.

Main alternatives for vertical die placements in in-package integration

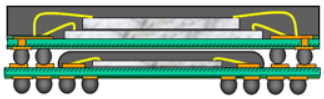
3D-stacked packaging by using TSVs (Through Silicon Vias)

The dies are 3D stacked, interconnected through TSVs and enclosed into a single module



PoP packaging (Package-on-Package)

BGA (Ball-Grid-Array) packages are interconnected and enclosed into a single module



a) 3D-stacked packaging does not use interposer layers but vias through active dies.

b) In the PoP technology individual BGA (Ball-Grid-Array) packages are interconnected and enclosed into a single module, in fact into a package.

Benefits of PoPs

- discrete BGA packages can separately be tested,
- only good parts will be used in the final assembly.

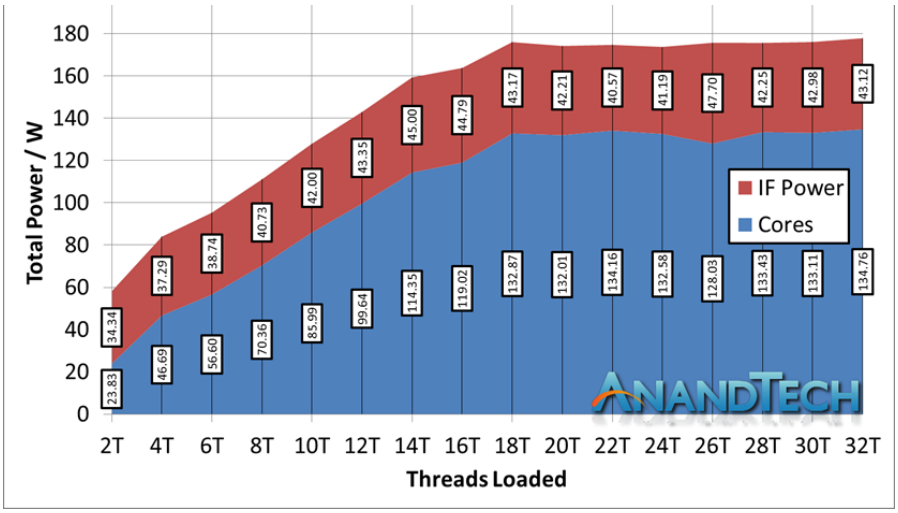
PoP packaging is widely used in smartphones and tablets,

- to efficiently implement logic and memory BGA packages in the same module, like in in Apple's devices, or
- to raise memory size of a package by enclosing two or more memory packages in a single module.

Example: Apple's iPhone incorporates a PoP (fabricated by Samsung) that includes the application processor and 2 memory chips. First PoPs emerged in cell phones around 2004. First generation PoP technology typically integrates the baseband or application processor with one or two memory dies.

Die-to-die interconnects may have a notable power consumption.

The IF of the 16-core (2 active dies) AMD TR 2950X consumes about 25 % of the total power.



The IF of the 32-core (4 active dies) TR 2990WX consumes already about 35 % of the total power due to the much more complex die-to-die interconnects.

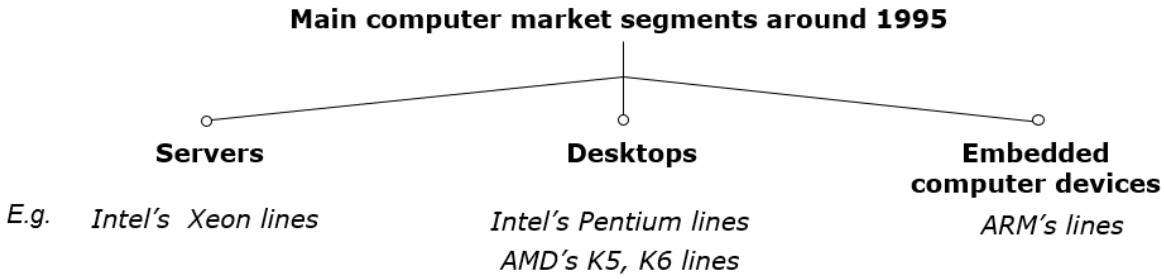
The IF of the 32-core (4 active dies) EPIC 7601 consumes already up to 50 % of the total power due to the much more complex die-to-die interconnect.

Note that the indicated power consumption includes both the die-to-die and the on-die power consumption. It concludes that with increasing core count - beyond providing enough memory bandwidth - the implementation of low power, scalable and high performance interconnects will be a great challenge for processor designers.

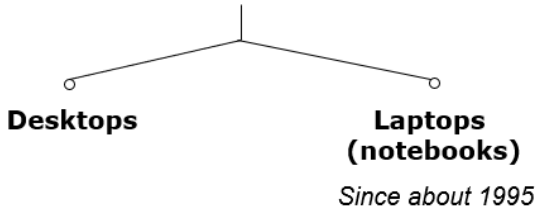
The mobile revolution

rrrr.	**short history of mobile phones and tablets,**
ssss.	**typical use cases,**
tttt.	**requirements /low power, connectivity, etc.,**
uuuu.	**the CPUs of Intel and AMD for mobile platforms**

The traditional computer market around 1995



Around 1995: spreading of multimedia, graphics and internet and emergence of laptops (aka notebooks) (portable computers).



Emergence of smartphones

Forerunners of smartphones emerged already at the beginning of the 2000's. The Nokia 7650 became the first widely available phone with camera and color screen but supported no video. It was the first Nokia phone running under the Symbian OS.

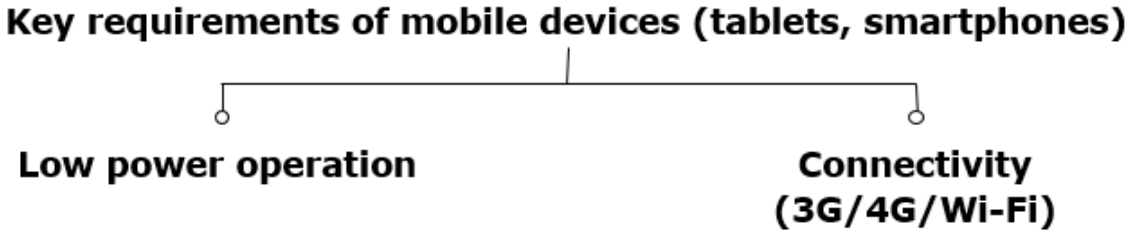
The emergence of smartphones is often contributed to the BlackBerry Pearl 8100 line of the Canadian firm RIM (Research in Motion). This phone – shipped in 2006 - supported beyond a camera also video and became very popular in the US. It was run under the BlackBerry OS.

In 2007 Apple's iPhone gave a strong momentum for rapid spreading of smartphones. It run under the iPhone OS (renamed later to iOS in 2010).

Google's Android was unveiled also in 2007 with first Android-powered phones sold in 2008.

Emergence of tablets

In 1972 at the ACM National Conference a Personal Computer for Children was introduced, called the Dynabook. From 2009 Android-based tablets arrived the market from many vendors. In 2010 Apple's iPad was introduced with 9.7 " screen, touch screen and Wi-Fi or additionally wireless 3G broadband internet connection (mobile internet connection), operating under iOS.



Low power consumption is expressed

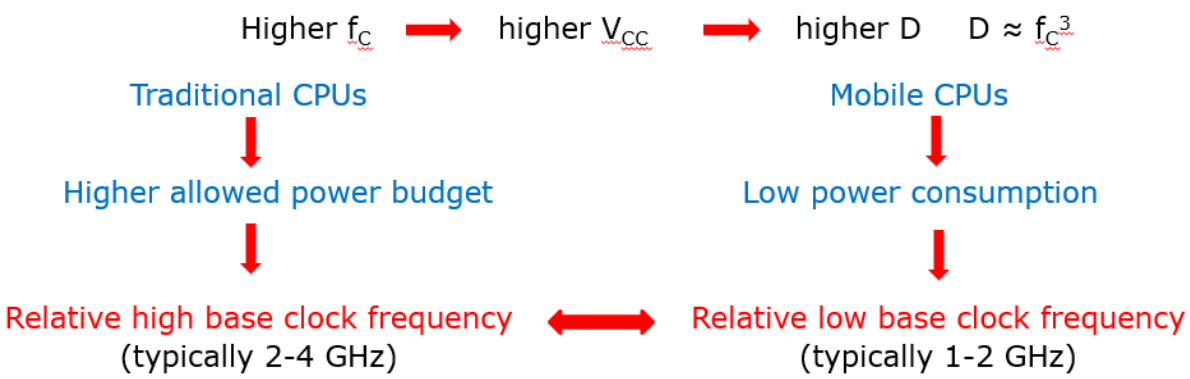
- either by specifying the power consumption, e.g. the TDP value of the processor in Watt,
- or in in length of the operating hours of the device under given conditions.

Contrasting the design paradigms of traditional and mobile processors

Traditional processors		Tablets and smartphones
High performance/power (Power efficiency) (e.g. GFLOPS/Watt)	⟷	Low power consumption (Watt) (Number of operating hours)

Intel's and AMD's traditional CPUs are designed for high performance/power, but mobile devices require low power consumption, consequently Intel's and AMD's traditional microarchitectures are not suited for mobile devices. To avoid shrinking market shares on the global processor market and benefit from the rapidly increasing mobile market Intel and AMD needed processors that are competitive with ARM based designs. Thus Intel and AMD were forced

- to introduce new, narrow (i.e. 2-wide), low-power microarchitectures for their CPUs and
- clock them at a relatively low rate.



To achieve low power consumption first application processors used in tablets and smartphones were "thin" designs with a typical decode width of 2. Later, advancements in IC technology and power management allowed to widen the microarchitecture of mobiles without noticeably raising their power consumption.

Apple's A10 (used in the iPhone 7 (2016) introduced the widest (7-wide) microarchitecture, and Apple's subsequent processors up to the latest A13 (Bionic 2019) chip (used in the iPhone 11 and iPhone 11 Pro) have a 7-wide front-end. The A13 is the most powerful mobile processor to date.

Intel introduced the Atom processors to penetrate into the market. In 2015 they achieved the 3. place in the worldwide market share in tablet application processor revenue.

Tablet application processors worldwide market share 2015 (revenue) [57]	
Apple (USA)	31 %
Qualcomm (USA)	16 %
Intel (USA)	14 %

Due to their high losses Intel first stopped paying subsidies and then in 2016 announced their withdrawal from the mobile market. AMD also withdrew from the market at approximately the same time.

When it comes to AMD, their Cat line (Bobcat, Jaguar, Puma) was aimed at the smartphone market. They were also unsuccessful in gaining a significant market share, but the Jaguar and Puma architectures from AMD are used in game consoles nowadays(Playstation, XBOX).

Connectivity may include (depending on the model features)

- LAN connectivity
- Wi-Fi connectivity (coined after Hi-Fi)
- mobile broadband connectivity (recently 3G/4G).

Integrating the modem onto the application processor die results in less costs and shorter time to market. Qualcomm pioneered this move designing integrated parts already about 1996.

The ARM CPU families

vvvv. **The business model of ARM,**

wwww. **ARM's product lines,**

xxxx. **the ARM Cortex-A family,**

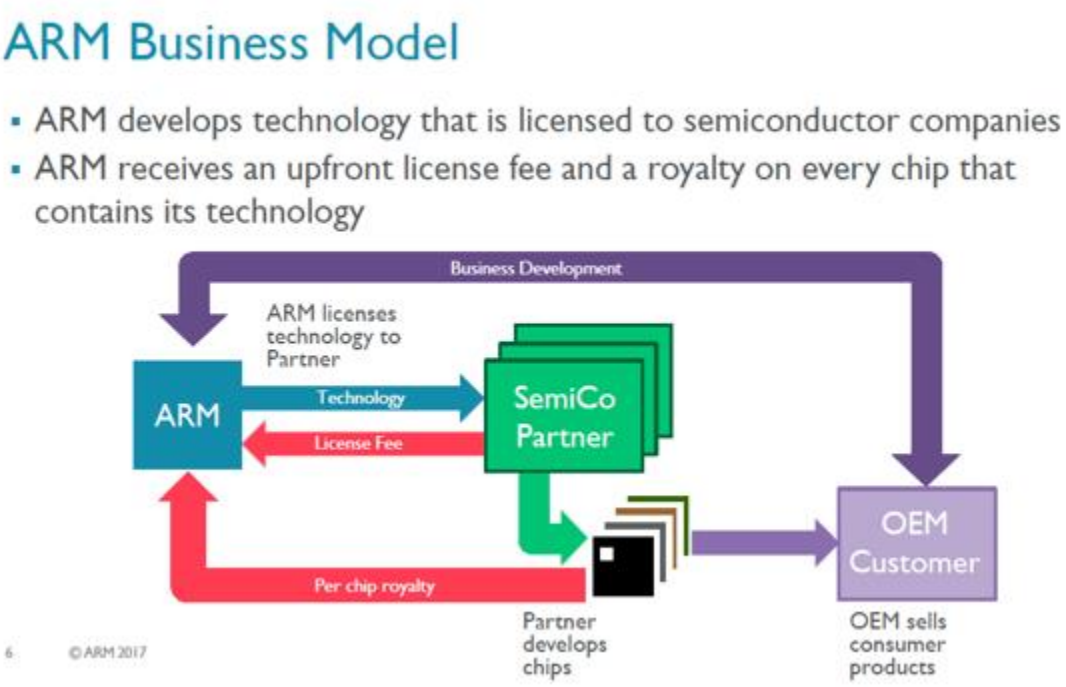
yyyy. **ARM ISA**

ARM (ARM Holdings plc) is a British multinational semiconductor company with its head office in Cambridge, acquired by Softbank (Japan) in 2016. ARM does not manufacture processor hardware. Instead, ARM creates microprocessor designs that are licensed to their customers, who integrate them into System-on-Chip (SoC) devices.

ARM

- designs
    - ARM processors for the embedded, mobile and server market,
    - mobile GPUs (termed as Mali GPUs) as well as
    - design tools (development studios etc.),
- and licenses
    - their IP (Intellectual Property) including their ISA but does not fabricate semiconductors.

To guarantee interoperability and to provide a common programmers model between different implementations, ARM defines architecture specifications that define how ARM products must operate. Additionally, some partners license the right to implement their own ARM processors conforming to the architecture specifications.



ARM's processor series

Processors implementing the ARMv1 – ARMv2 ISA	**Processors implementing the ARMv3 – v6 ISA**	**Processors implementing the ARMv7 – ARMv8 ISA**
(Earliest ARM processors) (~ 1985-1990)	*(Early ARM processors)* (~ 1991-2004)	*(ARM's Cortex processors)* (Since 2004/2012)
ARM1-ARM3	*ARM6xx-ARM11xx*	*Cortex lines*
26-bit address bus 32-bit data buses	32-bit address bus 32-bit data buses	ARMv7: 32-bit ARMv8: 64-bit with AArch64 and AArch32 modes

Processors based on the ARM ISA versions ARMv1 and ARMv2 (like ARM1, ARM2 or ARM3) implement only a 26-bit address bus and a 32-bit data bus. They are called 26-bit architectures. By contrast, processors based on the ARMv3 or higher ISA version support already a 32-bit address space and are known as 32-bit architectures up to the ARMv7 ISA.

In the ARMv8 ISA the AArch64 mode supports already 64-bit addressing.

2003: First public disclosure of the ARM11 MPCore.

2004: ARM11 MPCore (multi core ARM11 with 1-4 cores) available for licensing with an evaluation system for early software development.

It is basically an up to four way (4-socket) cache coherent symmetric multicore processor

The ARM11 MPCore incorporates the IEM (Intelligent Energy Manager): It dynamically predicts the required performance and lowers the voltage and the frequency accordingly.

Processors implementing the ARMv7 or ARMv8 ISA are designated Cortex family processors. The Cortex-family along with its first member, the Cortex-M3 processor was announced in 2004, without disclosing the ARMv7 ISA. AMD revealed the technical specification of the ARMv7 ISA finally in 2005.

Along with revealing the technical specifications for the ARMv7 ISA in 3/2005 ARM introduced three family profiles to optimize its Cortex family processors for specific market segments:

The Cortex-A (application) profile

- It aims at application processors for complex OS and user applications, like processors in smartphones, tablets, netbooks, eBook readers etc.
- The Cortex-A family supports a Virtual Memory System Architecture based on a Memory Management Unit (MMU).
- It supports three instruction sets:
    - the A32 (AArch32),
    - the A64 (AArch64) since ARMv8 (2012) and
    - the T32 (Thumb32)

The Cortex-R (Real-time) profile

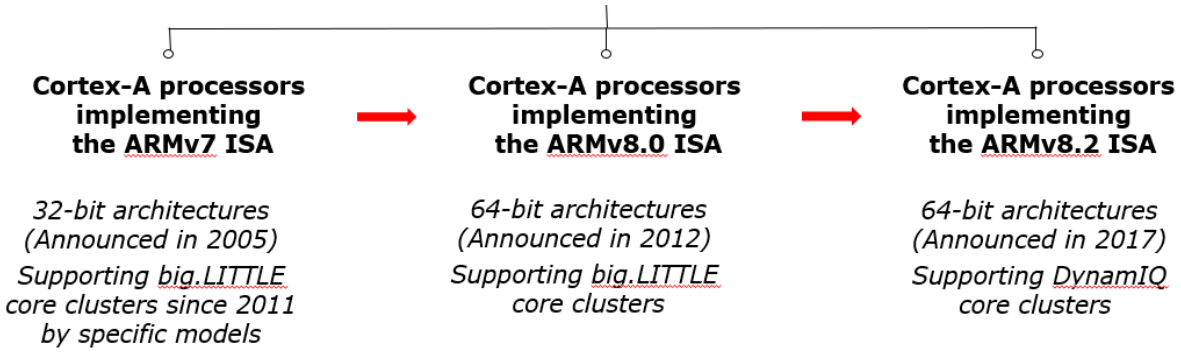
It marks embedded processors for real time applications, like mass storage or printer controllers. It implements a Protected Memory System Architecture (PMSA) based on a Memory Protection Unit (MPU). Supports the A32 and T32 instruction sets.

The Cortex-M (Microcontroller) profile

- Processors of the M profile are optimized for deeply embedded processors aimed at microcontroller and cost sensitive applications, like automotive body electronics, or smart sensors.
- Supports writing interrupt handlers in high-level languages.
- It implements a variant of the Protected Memory System Architecture (PMSA).

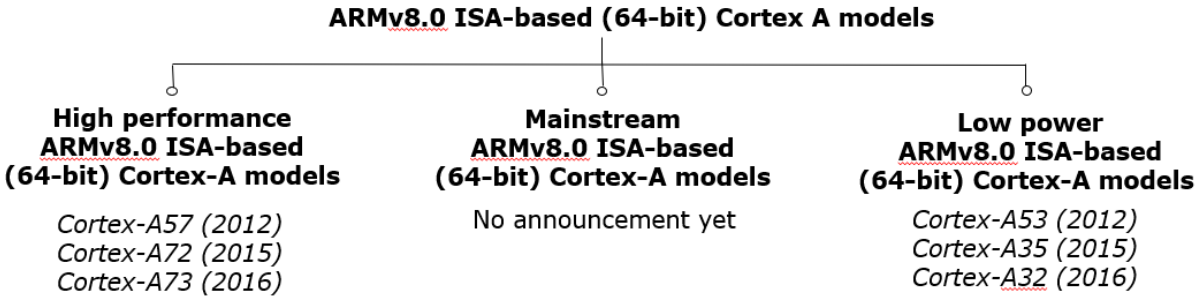
- It supports a variant of the T32 instruction set.

+1: In 2007 ARM introduced the **SecurCore profile**. It is optimized for smart card and secure applications.



Key features of ARMv7 - ARMv8.0 ISA based Cortex-A models

- Word length of the models
    - ARM's 32 bit Cortex-A models (Cortex-A models implementing the ARMv7 ISA)
    - ARM's 64 bit Cortex-A models (Cortex-A models implementing the ARMv8 ISA)
- Inclusion of L2 cache
    - No L2 cache
    - Optional L2 cache
    - Mandatory L2 cache
- Multiprocessor capability
    - Single processor designs
    - Dual designs with two options - single processor and multiprocessor option
    - A-priory multiprocessor option
- Support for big.LITTLE configurations
    - No support for big.LITTLE configurations
        - First 32-bit (ARMv7) models, such as the Cortex-A8 (2005), Cortex-A9 (2007), Cortex-A5 (2009)
    - Support for big.LITTLE configurations
        - Advanced 32-bit (ARMv7) models, such as the Cortex-A15 (2010, bL support: 2011), Cortex-A7 (2011), Cortex-A17 (2014) and all 64-bit (ARMv8) models



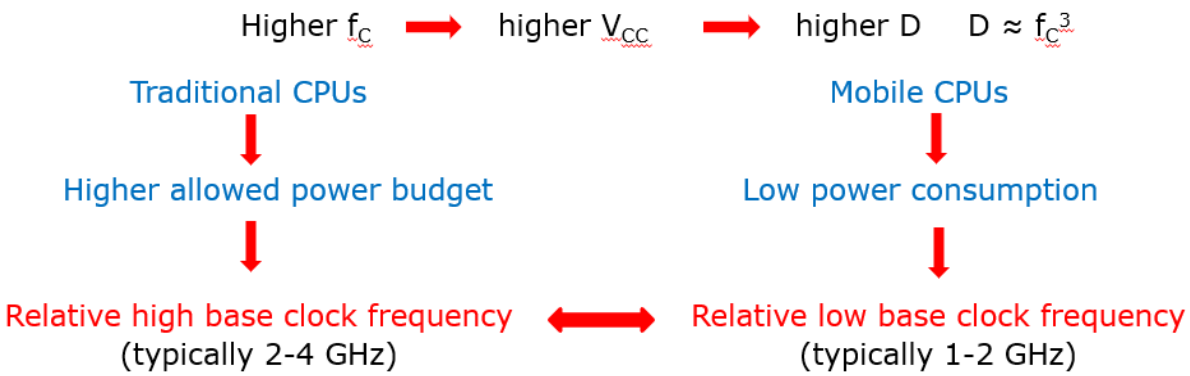
The big.LITTLE architecture of ARM

zzzz.	**Main problems of mobiles: power consumption vs computational performance,**
aaaaa.	**power gating and clock gating,**
bbbbb.	**DFVS,**
ccccc.	**the big.LITTLE architecture,**
ddddd.	**big.LITTLE schedulers**

Power gating switches off idle units on the die (like a core) from the power supply by means of power transistors (sleep transistors) to eliminate both dynamic and static power consumption caused by leakage currents.

Clock gating is a technique used to reduce dynamic power by disabling clocking to currently not needed circuits. Clocking is disabled by inserting AND gates (i.e. controlled on/off switches) that are called clock gaters into the clock distribution network, as indicated below.

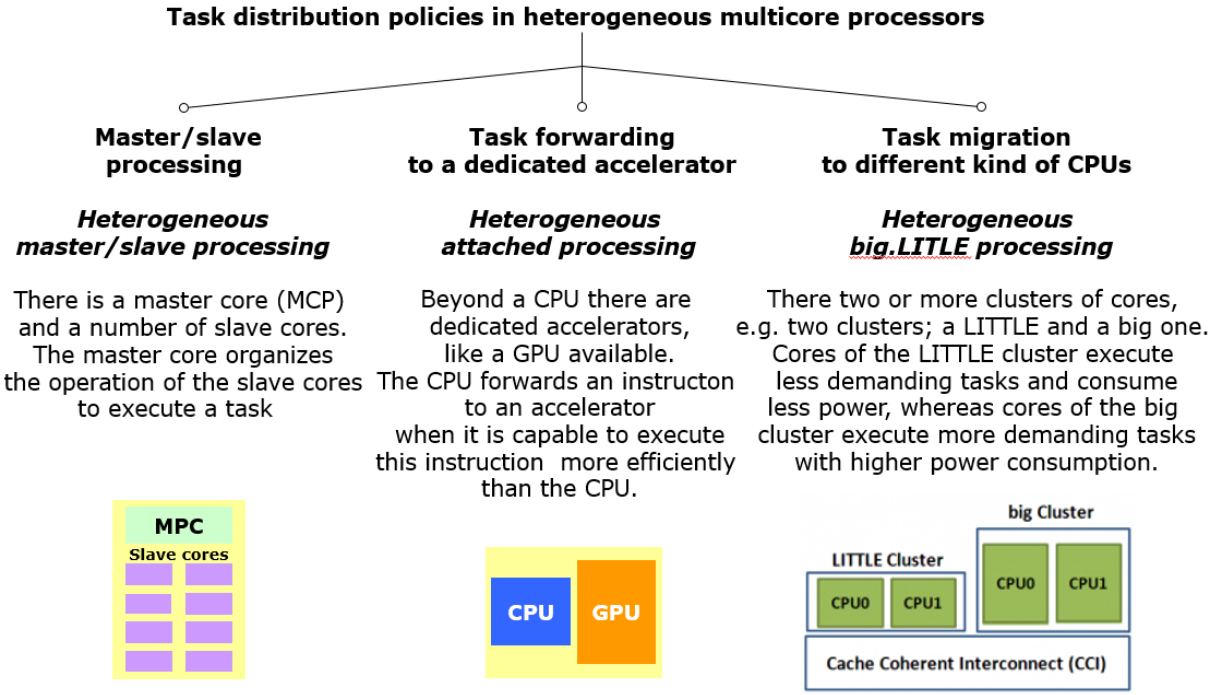
Contrasting the design paradigms of traditional and mobile processors



big.LITTLE technology aims at power reduction. The technology is based on the recognition that smartphones and tablets have dynamically changing usage patterns, i.e. tasks with different processing intensities alternate, such as high intensity tasks, like games and low intensity tasks, like audio, e-mail etc.

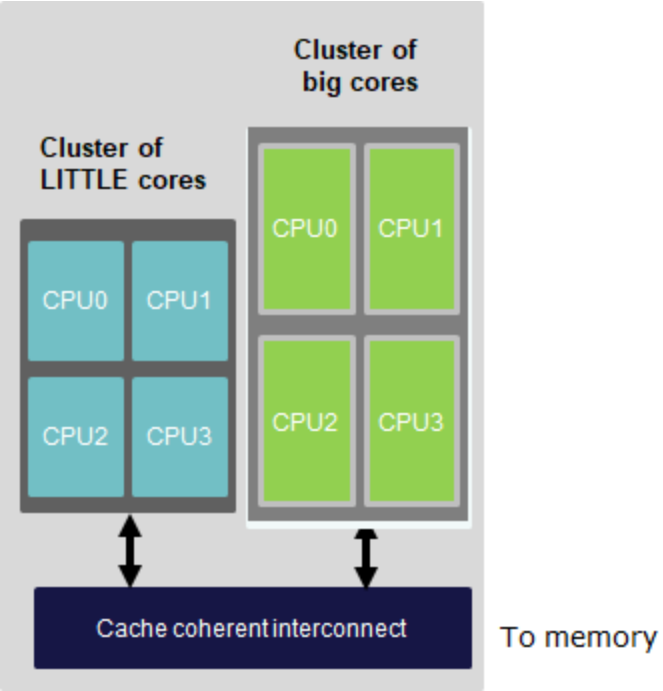
The idea of the big.LITTLE technology is that power consumption of the chip could be reduced when

- high intensity tasks would run on suitable powerful but power hungry
- whereas low intensity tasks on less powerful and less power hungry cores.



Let's have two or more clusters of architecturally identical cores in a processor. As an example let's take two clusters;

- a cluster of low performance/low power cores, termed as the LITTLE cores and
- a cluster of higher performance higher power cores, termed as the big cores, as seen in the Figure below.



Let's interconnect these clusters by a cache coherent interconnect to have a multicore processor, as indicated in the Figure.

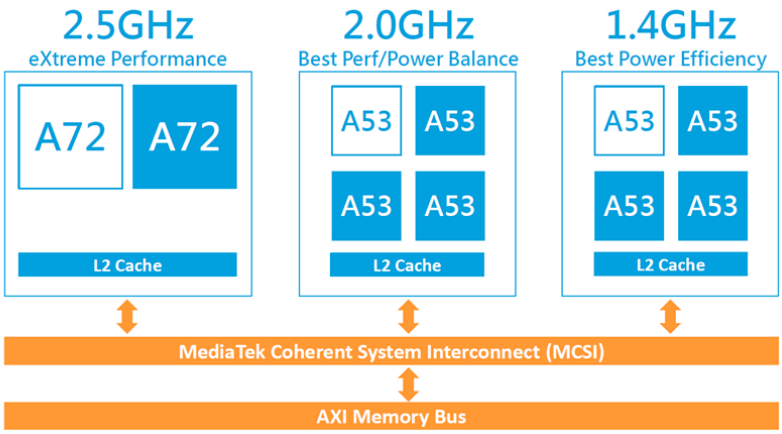
Principle of operation

The scheduler activates only the cores of the low power, low performance „LITTLE" cluster as long as this cluster can tackle the actual workload. If however the workload exceeds the performance capability of the LITTLE core cluster, an appropriate switching routine activates the cluster of high performance high power "big" cores, performs a cluster switch by migrating the actual workload to the cores of the big cluster and switches off the cores of the LITTLE cluster. In this way, at any given time only cores of one cluster can be active, this kind of the big.LITTLE technology is called **Exclusive cluster switching**. When any time the actual workload becomes less than a given lower limit of the big core cluster, the scheduler activates again the cluster of the LITTLE cores.

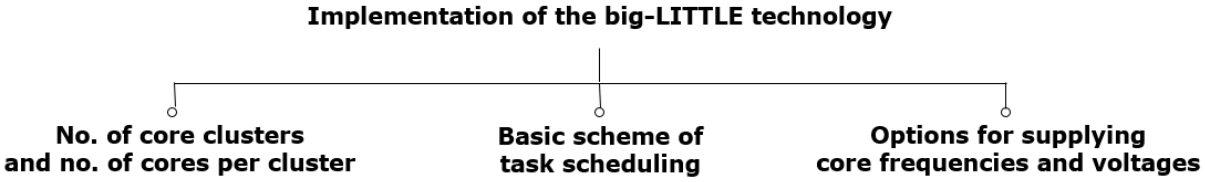
big and LITTLE cores of a big.LITTLE system need to satisfy a few requirements for a seamless operation, such as

- the big and LITTLE CPU cores must be architecturally identical, i.e. they must have the same ISA, i.e. run the same instructions and support the same ISA extensions, like virtualization, address space etc. and
- both core clusters must be fully cache coherent to support task migration.

First three cluster (10-core) implementation (2016): MediaTek Helio X20 (6797). In this case, there are two A53 clusters with different maximum clock frequencies and different performance-power characteristics.



The big.LITTLE technology needs appropriate software support, e.g. to monitor the workload, task scheduling, perform task migrations etc.

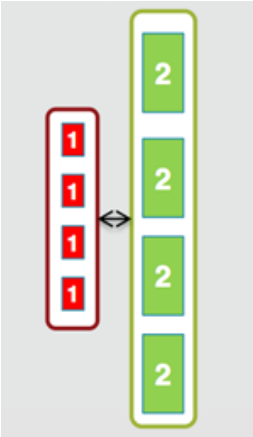


No. of core clusters and no. of cores per cluster

- Dual core clusters used (E.g. 4+4 cores, Samsung Exynos Octa was the world's first octa core mobile processor, launched in some Galaxy S4 models.)
- Three core clusters used (E.g. 2+4+4 cores, Helio X20 was the world's first three cluster 10-core big.LITTLE implementation. All cores in the clusters run at the same clock frequency. Announced in 2015, launched in HTC One A9 in 2015.

Basic scheme of task scheduling

- Migrating tasks between big.LITTLE clusters or between cores of big.LITTLE core pairs
    - Cluster migration - Clusters are sets of cores. Tasks will be migrated between clusters of cores.

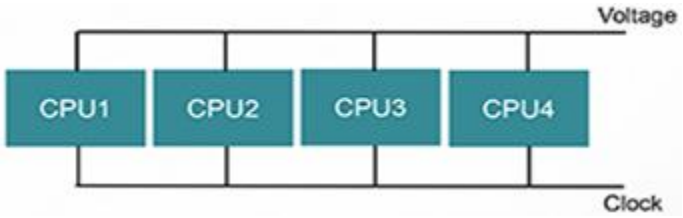


- Core migration - Tasks are migrated between individual cores, e.g. between LITTLE and big cores of core pairs of LITTLE and big cores.

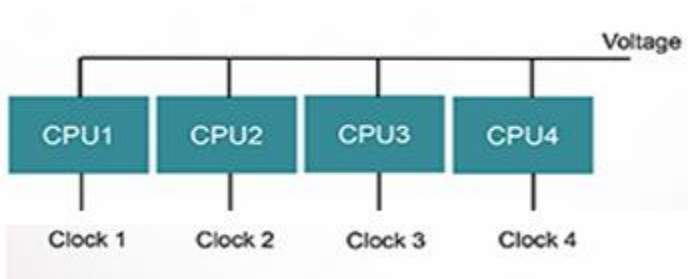
- Exclusive or inclusive big.LITTLE execution
  - Exclusive use of big and LITTLE clusters or cores - At any time either the big or the LITTLE execution resources are in use as indicated next for both the cluster and core migration model
  - Inclusive use of big and LITTLE clusters or cores - For high workloads both the big and the LITTLE execution resources are in use as indicated next for both the cluster and core migration model

Options for supplying core frequencies and voltages in clusters

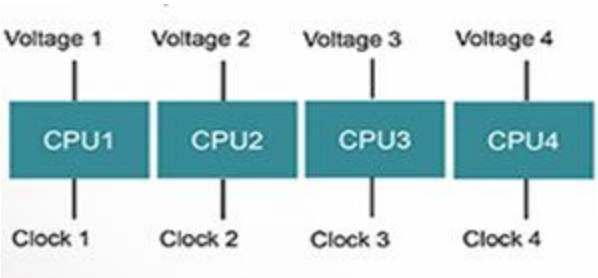
- Common core frequency - common core voltage



- Per core frequency - common core voltage



- Per core frequency - per core voltage



The cluster switch approach was the default scheduler in Android 4.2.2.

EAS (Energy Aware Scheduling)

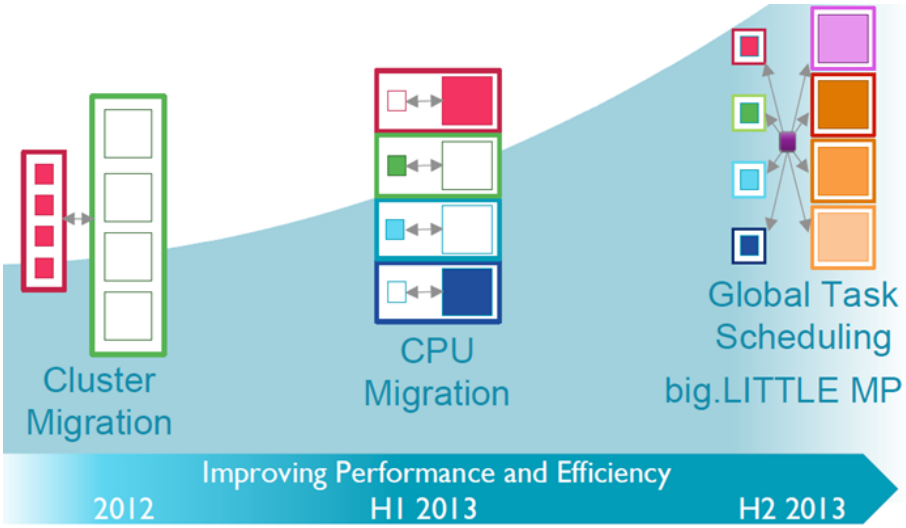
Ideally, each cluster will operate at its own separate independent frequency and voltage. By lowering the voltage and frequency, there is a substantial power saving. This allows the per-cluster power/performance to be accurately controlled, and tailored to the workload being executed.

IKS

Linaro developed a model for task scheduling on big.LITTLE SOCs, called IKS (In Kernel Switcher) and designed an appropriate Linux kernel patch for an experimental system. IKS builds core pairs from the cores of the big and LITTLE core clusters, e.g. from Cortex-A15 and Cortex-A7 cores, and treats each core pair, consisting of a big and a LITTLE core, as a single virtual core.

The Linux kernel schedules then the tasks to the virtual cores. For each core pair the cpufreq driver of the Linux kernel controls whether the LITTLE core (for low power) or the big core (for maximum performance) should be activated. An important feature of IKS is that it relies on existing Linux kernel scheduling mechanisms and thus is easy to implement.

Global task scheduling (GTS) or big.LITTLE MP in ARM's terminology, can be considered as the final step of the evolution of the big.LITTLE technology, as indicated below.



OS (e.g. a modified Linux scheduler) tracks the average load of each task. The OS scheduler has all cores of both clusters or of all three clusters at its disposal and can schedule tasks to any core at any time. There are many options for the layout of the scheduling policy.

Key benefits of GTS over IKS

- Finer grained scheduling of workloads.
- Ability to easily support non-symmetric configurations, such as (2+4) ones. Higher peak performance through the ability to use all cores simultaneously.
- Higher performance/Watt vs. IKS.
- ARM reported about 10 % higher performance/Watt figures over IKS on a range of benchmarks.

With Symmetric Multi-Processing (SMP), the Completely Fair Scheduler (CFS) of the Linux kernel implements currently the most common scheduling algorithm, it distributes the workload equally among CPU cores. In case of Heterogeneous Multi-Processing, however, employing CFS can cause performance degradation, since tasks do not efficiently match to CPU core capabilities. MediaTek's CorePilot, on the other hand, is based on a true heterogeneous compute model by using a scheduling algorithm that assigns tasks to two different schedulers, according to their priority — the Heterogeneous Multi-Processing (HMP) scheduler and the Real-Time (RT) scheduler.

Lately, not just CPU cores and clusters are used by schedulers but also the GPU.

Qualcomm's Power aware scheduler (2014)

Enhancement of the ARM/Linaro GTS. Qualcomm's Power aware scheduler covers three tasks, as follows:

- load tracking: Tracking CPU demand is critical for an efficient scheduling.
- power module: This model provides the interrelationship between the core frequency and the execution efficiency in terms of mW/MIPS.
- hmp scheduler: Based on the available information on computing demand, energy impact on core performance etc. the hmc scheduler allocates tasks to the cores and migrates the tasks between cores if necessary.

Qualcomm's Symphony System Manager (SSM)

SSM achieves this by scheduling tasks to the right computing resources, such as (big or LITTLE Kryo cores, GPU or accelerators) by taking into account the required load and needed energy consumption to perform it. For example, when a user is taking a picture, SSM powers up the right components (CPU, Spectra ISP, GPU and memory system) and directs them to run at the needed frequency as long as required. In this way SSM chooses the most efficient and effective combination of cores and accelerators to perform a task as quickly as possible, with the least power consumption.