- **Types of computers**
  a. PC : Price and performance
  b. PMD : Energy efficiency and real-time
  c. Servers: Availability, Throughout, Scalability.
  d. WSC(used for SaaS): Availability, Price
  Subclasses:
  Super Computers: floating-point performance, fast internal networks.
  e. Embedded computers: price, robustness

- **CA:** The science and art of designing, selecting and interconnecting hardware components and design the hard/soft interface to create a computational system that meets specific criteria(performance/power/cost/etc..)

- **Problem solution stack**
  1. Problem
  2. Algorithm
  3. Data structures
  4. User program
  5. System program
  6. ISA
  7. Microarchitecture
  8. Circuits
  9. Electrons
- **Architecture:**
  Programmer's view of computer(instructions and operand locations)

- **Microarchitecture:**
  How to implement an architecture in hardware.

- **Von-Neumann Architecture: (Most used today in x86, ARM, MIPS, SPARC, ALPHA, POWER, ...)**
  1- Stored program concept:
  Instructions (Program code) and data are stored in the same memory unit
  2- Sequential instruction processing:
  Instructions (fetched - decoded - executed) sequentially (except for jumps) one per time, program counter (PC) decides next instruction to do.

- **How the CPU distinguish instructions and data? Control signals**

- **Control Unit (CU): Core of sequencing of operations**
  Fetches, Decodes and Coordinates execution for instructions.

- **Arithmetic-Logic Unit (ALU): Core of computation (works with binary numbers)**
  Performs Arithmetic, logic and shifting operations.

- **Registers: Places for data needed for execution**
  - Small in size and capacity, fast to access and expensive, thus, it's on top of the memory hierarchy
  - MIPS has 32 registers, each of 32-bit capacity
  - Can be constructed from several 1-bit flip-flops.
  - Can be grouped together --> Register files
  - Easy for compilers to use (That's why today's architectures are register-Set based)

- **Bus:**

  Definitions:
  - Bus is Interconnection between different components/Common electrical pathway between multiple devices.
  - Data sender is called master/active device (Memory can't be a master)
  - Data receiver is called slave/passive device

  Types:

On-chip bus	Off-chip bus
Connects components on the same chip. AXI: connects CPU with GPU/L2 cache	Connects components in different chips PCI-Express: connects components outside the CPU SPI/IIC/UART: connects devices like sensors and serial-terminals

	Parallel Buses	Serial Buses
Data lines	Several	Only one
Price	More expensive	Less expensive
Throughput	Higher	Lower
Examples	ISA/EISA/PCI	USB/IEEE 1394

	Synchronous Buses	Asynchronous Buses
Have a clock	Yes (5-100) Hz	No
Bus Cycles	All of the same length	Can be of any length
Implementation	Easy	Harder

In Synchronous bus, the system works at the speed of the slowest device, however, most buses today are synchronous

Bus Protocol:

Set of rules governing how the bus works
Makes it possible for boards designed by third parties to be attached

- **CPU chip control pins:**
    1. Address and data pins: Can share the same physical pin
    2. Interrupts: Intel chip has 2 levels, Motorola has 8
    3. Bus arbitration: Intel has Bus Request/Grant, Motorola has Bus Request/Grant/Acknowledge
    4. Bus control: to indicate if it's memory or I/O access
    5. Cache control: to allow CPU to communicate with a cache chip
    6. Status: for indicating the current status of the CPU
    7. Miscellaneous: ex. the reset pin

- **Cache:**
    - Component that stores data so that future requests for that data can be served faster
    - Can store the result of an earlier computation OR a copy of data stored elsewhere
    - Modern processors usually have multiple caches for data, instructions, Translation Lookaside Buffer
        - TLB: memory cache that is used to reduce the time taken to access a user memory location.
    - Caches are usually placed physically close to the processor
- **Cache levels**
    - L1 caches: for each thread
        - Hyper-threads usually have separate L1 cache
        - Hyper-threading is a process where a CPU splits each of its physical cores into virtual cores, which are known as threads.
    - L2 caches: for each core
    - L3 caches: shared with all components of the CPU
- **Cache types:**
    - Fully Associative cache
    - Direct-mapped cache
- **How cache works:**
    - The processor is asking memory for data
    - Cache is checked
    - In case of hit -> load data from the cache
    - In case of miss
        - load data from memory to cache (free space in cache if necessary)
        - load data from cache to processor

- **Main memory:**
    - Direct addressed, fast access, volatile (loses data when there is no power)
    - Support only two operations (Read/Write) - (Load/Store)
    - Byte addressed
    - Words are aligned [word has size 4x -bytes]

- **Disk storage:**
    - Sequential in accessing nature (while randomly accessible), slower access, non-volatile, large capacity

- **I/O Devices:**
  - Wide range of speed/requirements/least amount of research

- **Clock**
  - Device in processor that does ticks, measures time | Free-running signal with a fixed cycle time.
  - It has: rising edge, falling edge, cycles of constant time(Clock period/frequency/rate/ticks)
  - Determines the sequence of events/Synchronizes circuit's operation
  - Presence of a global clock means the system is synchronized.
- **Clock Generator:**
  - Produces clock signals
- **Clock Signal:**
  - Signal that oscillates between high and low state
  - All circuit actions happen only on the rising or the falling edge of the clock
- **Edge-triggered methodology**
  - A clocking scheme in which all state changes occur on a clock edge.
- **Critical path:**
  - The longest path in a circuit and limits the clock speed.

- **Five components of any computer:**
  1. CU (computer brain)
  2. Datapath (computer brawn)
     - Datapath is a collection of useful components inside a CPU Ex. (ALU - Multipliers - ...), registers and buses
  3. Main Memory
  4. Input Devices
  5. Output Devices
     - Disk can be both I/O device
- **Technologies used in computer:**
  - Vacuum tubes
  - Transistors(on/off switches controlled by electrical signal)
  - Integrated Circuits (IC's)
    i. A tiny circuit made up of resistors, diodes and transistors/Device containing dozens to millions of transistors)
  - VLSI (Very Large Scale IC) (Tens of thousands to millions of transistors)
  - ULSI (U: Ultra)

- **Instruction Set Architecture (ISA): The interface between hardware and low-level software**
  - Advantage: different implementations of the same architecture
  - Disadvantage: sometimes prevents using new innovations

- **Organization of computers today:**
  - Computational devices: CPU/DSP/GPU
  - Memory hierarchy, I/O devices, Bus

- **Nine Great ideas in Computer Architecture:**
  - **Moore's law:**
    1. IC resources (number of transistors on a chip) doubles every 18-24 months
  - **Abstraction**
    1. Hiding low-level details to offer simpler model at higher levels
  - **Performance via parallelism**
    1. Executing operations in parallel to achieve performance
  - **Performance via prediction**
    1. guess and start working rather than wait until you know for sure, assuming that the cost of recovering from a misprediction is not too expensive.
  - **Performance via pipelining**
    1. Connecting a set of data processing elements is such a way that the output for each one is the input of the next one.
  - **Common case fast**
    1. Emphasize on developing the performance in particular cases which are usually wanted.
  - **Memory hierarchy**
    1. Putting the fastest, smallest, most expensive piece of memory at the top of the hierarchy and the slowest, biggest, least expensive at the bottom. ex: register > cache > RAM > Disk
  - **Dependability via Redundancy**
    1. Adding some unnecessary redundant code that can take over when a failure occurs and to help detect failures.
  - **Finite State Machines (FSM's):**
    - i. Machines that can be at one of finite states at a time
    - ii. It is defined by Initial state and transition function
    - iii. F: I*S -> O*S (I: set of inputs, O: set of outputs, S: set of states)
    - iv. A block of combinational logic + a register to hold the current state
    - v. **Examples:** Traffic lights, vending machines, elevators, ..
- **How to define performance? GQM principle:**
  - Goal: why you measure?
  - Question: what do you need to determine?
  - Metrics: set suitable measurement device
  - **What performance metrics of a processor do you know?**
    - Time/Latency, Throughput, Power Consumption
- **Response/Elapsed/Execution/Wall clock time: time needed for everything.**
- **CPU (execution) time: (Ignores Memory/Disk/I/O access time)**
  - **CPU user time:**
    1. Time spent in program (determined by the number of lines of machine code)
  - **System CPU time**
    1. Time spent in OS doing tasks on behalf of the program
- **Is more always better?**
  - Replacing the processor with a faster version increases both latency and throughput
  - Adding more cores to a parallel system makes computer faster
  - Adding more processors to a system that uses multiple processors for separate tasks increases throughput but not response time
  - Two slow processors vs one fast one are not comparable
  - High clock speed (more than 3.5 GHz) is not practical due to heat dissipation problems.
  - To compare two processors, benchmarks are used.

- **Types of benchmarks:**
  - Real program
  - Component Benchmark / Microbenchmark
  - Kernel
  - Synthetic Benchmark
  - I/O benchmarks
  - Database benchmarks
  - Parallel benchmarks

- **To increase computer performance:**
  - Add more parallelism to microarchitecture
  - Add hardware support for multithreading
  - Use multi-core clusters
  - Use specialized hardware, optimized for certain tasks (like GPU for 3D graphics)
- **Calculating Performance:**
  - **Abbreviations:**
    - i. Performance (P)
    - ii. Execution time (ET)
    - iii. Number of clock Cycles (NC)
    - iv. clock Cycle Time (CT)
    - v. Average number of clock Cycles Per Instruction (CPI)
    - vi. Instruction Count (IC)
    - vii. Clock Rate (CR)
  - **Formulas**
    - i. P = 1/ET
    - ii. Px/Py = n => x is n times faster than y
    - iii. ET = NC * CT = NC / CR = IC * CPI * CT = (IC * CPI)/CR
      1. changing CT often changes NC
      2. seconds/program = cycle/program * seconds/cycle
- **Take into account when measuring performance:**
  - Technologies used in processor, organization and software

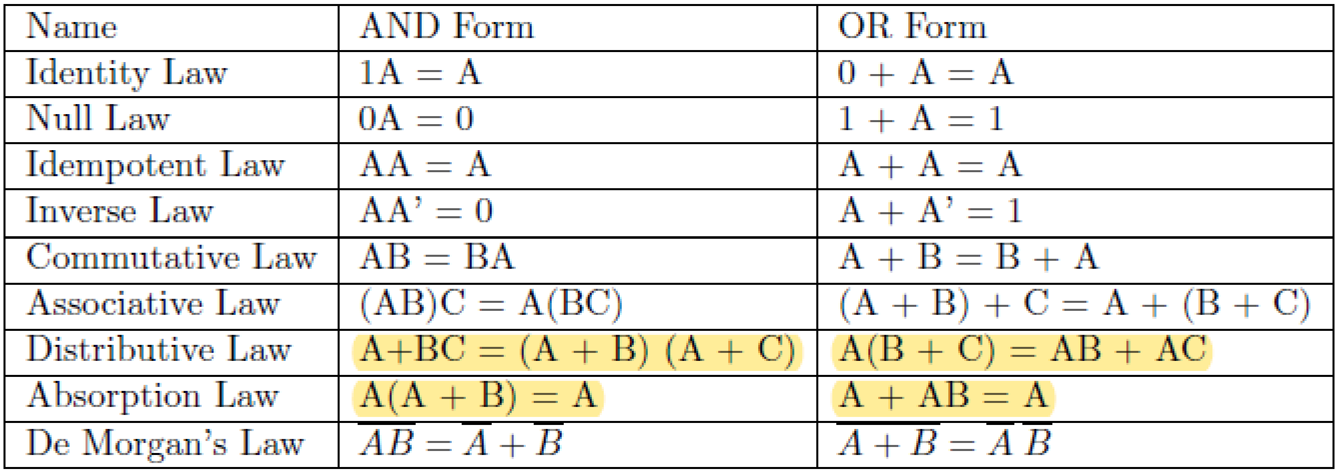
- **Amdahl's law**
  - Execution time after improvement = Execution time unaffected + (Execution time affected / Amount of improvement)
- **What makes number of cycles different from number of instructions?**
  - Multiplication takes more time than addition
  - Floating-point operations take longer than integer ones
  - Accessing memory takes more time than accessing registers
  - In pipeline processors, multiple instructions may be executed at the same time.

- **Number: Abstract entity**
- **Numeral: String of symbols that represent a number in a given system**
  - The same number can be represented by different numerals in different systems
- **Positional Numeric System:**
  - Value of digit doesn't depend on its index (ex. Roman Numeric System)
- **Non-Positional Numeric System:**
  - Value of digit depends on its index
  - Each digit represents the coefficient of (base^index)
  - Adding the values of the digits together yields the number in decimal
  - Digits after the decimal point have a negative index
- **How to convert from decimal to any base?**
  - Divide by base, get quotient and remainder, divide the quotient by base again until you get zero
  - list the remainders from right to left.
- **Two complement and biased/excess [-2^(n-1), 2^(n-1)-1]**
- **One complement and Sign&Magnitude [-2^(n-1)+1, 2^(n-1)-1]**
- **Negative values representation in:**
  - **Sign and magnitude:** make the leftmost bit 1 then convert the number itself.
  - **One's complement:** convert the number itself, negate all
  - **Two's complement:** convert the number itself, negate all, THEN ADD +1
    1. WHEN CONVERTING FROM DECIMAL ADD LEADING ZEROS BEFORE NEGATING.
  - **Excess:**
    1. if the magic number is given add it to your target number then convert to binary
    2. else add (the nearest power of two - 1) to your target number then convert to binary
- Standard IEEE 754
  - Simple precision (32bit):
    1. Sign (1 bit) | Exponent (8 bits) | Mantissa (23 bits)
  - Double precision (32bit):
    1. Sign (1 bit) | Exponent (11 bits) | Mantissa (52 bits)

UNSIGNED	SIGNED
0000 0000 0000 0000 0000 0000 0000 0000 = 0 0000 0000 0000 0000 0000 0000 0000 0001 = 1 0000 0000 0000 0000 0000 0000 0000 0010 = 2 1111 1111 1111 1111 1111 1111 1111 1111 = 2^32- 1	0000 0000 0000 0000 0000 0000 0000 0010 = 2 0000 0000 0000 0000 0000 0000 0000 0001 = 1 0000 0000 0000 0000 0000 0000 0000 0000 = 0 1111 1111 1111 1111 1111 1111 1111 1111 = -1 1111 1111 1111 1111 1111 1111 1111 1110 = -2 1111 1111 1111 1111 1111 1111 1111 1101 = -3

- **Ternary Systems:**
  - Setun: The first ternary computer (MSU)
  - Standard (unbalanced) system: Which uses the values 0, 1 and 2
  - Balanced system: Which uses the values -1, 0, 1
  - Why it's not used
    1. It is harder to build components that have 3 states.
    2. If you use 3 states you need to be compatible to binary.

- **Unum: A floating-point number format, alternative to IEEE 754, implemented in Julia PL.**
- **Saturation Arithmetic:**
  - Limiting the result of operations to some interval max and min where all values cannot exceed them.
  - Used in many problems in DSP (ex. adjusting the volume level)
- **Boolean Algebra laws**



Name	AND Form	OR Form
Identity Law	$1A = A$	$0 + A = A$
Null Law	$0A = 0$	$1 + A = 1$
Idempotent Law	$AA = A$	$A + A = A$
Inverse Law	$AA' = 0$	$A + A' = 1$
Commutative Law	$AB = BA$	$A + B = B + A$
Associative Law	$(AB)C = A(BC)$	$(A + B) + C = A + (B + C)$
Distributive Law	$A+BC = (A + B)\ (A + C)$	$A(B + C) = AB + AC$
Absorption Law	$A(A + B) = A$	$A + AB = A$
De Morgan's Law	$\overline{AB} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}\,\overline{B}$

- **Truth tables:**
  - Representation of logic blocks by listing all the values of the inputs and showing the resulting outputs
- **Combinational logic:**
  - logic that has no memory elements
- **Sequential logic:**
  - logic that has memory present
- **Gate:**
  - Hardware unit that receives Boolean inputs and produces one output/implements a basic logic function
- **Decoder:**
  - A logic block that has an n-bit input and 2^n outputs where only one output is asserted for each input combination.
- **Multiplexor (mux/selector):**
  - Has 2^n data bits and n control bits and one output, Selects one of its inputs as an output.
- **Programmable Logic Array:**
  - Used to implement any combinational logic in hardware as sum of products
  - A set of inputs and complements for them and two stages of logic (AND gates then OR gates)
  - Not used today because it's not fast enough, consumes more voltage and have high gate count
- **Ripple carry adder:**
  - A circuit that produces the arithmetic sum of two binary numbers.
  - Constructed with full adders connected in cascade.
  - The output is known only after the carry of the previous stage is generated (DELAY)
- **Carry-Look ahead Adder:**
  - It solves the problem of ripple carry adder by calculating the carry signals in advance
- **A memory element that contains a value is the "state" of the element.**

- **Latch:**
  - A circuit used to store information.
  - ALL Latches are Asynchronous(Level sensitive)
  - Built from logic gates
  - SR latch: Doesn't have a clock/Has invalid state (S=1, Q=1)
  - D latch: Has a clock/Avoids invalid state

- **Flip-flop**
  - Synchronous version of latches.
  - ALL flip-flops are Synchronous(edge-sensitive)
  - Built from latches
  - D-flip-flop(edge-triggered): Two cascaded d-latches(master, slave)
  - Enabled flip-flop: limits the power consumption
  - Settable/Resettable flip-flop are the same thing (different only in design)

- **Register file:**
  - Array of processor registers, used to store data
  - Can be implemented using:
    1. Decoder for read/write ports
    2. Array of registers made of D-flip-flops
  - To read from Register file supply the register number as input and get its data as output
  - To write a register in register file we need register number, data to enter and the clock

- **Small memories are built from registers and register files**
- **Larger memories are built from DRAM's and SRAM's**

	DRAM	**SRAM**
Built from	Transistor and capacitor	Transistors and flip-flops
Used for	Main memory	Cache
Leakage problem	Yes	No
Price	Less expensive	More expensive
Speed	Lower	Higher
Power consumption	Higher	Lower
Volatility	Volatile	Volatile

- **Capacitor Leakage:**
  - Losing the capacitor charge when they have devices nearby that draw a little current
- **Byte Alterable:**
  - stored information can be changed without a separate erase step.

- **Field Programmable Devices (FPD):**
  - IC containing combinational logic (and memory devices) that can be programmed.
    1. Programmable Logic Devices (PLD): combinational.
    2. Field Programmable Gate Arrays: provides both combinational and sequential logic.

- **MIPS Instruction Types and format**
    - R type: Arithmetic/logic/shift instructions
        - i. Format: op(6), rs(5), rt(5), rd(5), shamt(5), func(6)
        - ii. All of them have 0 as opcode
    - I type: instructions involving immediate values
        - i. Format: op(6), rs(5), rt(5), imm(16)
    - J type: jumps(unconditional) and branches(conditional)
        - i. Format: op(6), addr(26)

    - op: operation code
    - rs, rt: register sources
    - rd: register destination
    - shamt: shift amount
    - func: function code
    - imm : immediate values or their address
    - addr: memory address for jump destination

- **MIPS Registers: (32)**
    - $zero : constant
    - $at : assembly temporary
    - $k(2) : reserved for OS kernel
    - $fp : frame pointer
    - $sp : stack pointer
    - $gp : global pointer
    - $ra : return address (for jal instruction)
    - $v(2) : function return value
    - $a(4) : function arguments (if it has more arguments, they are stored in the stack)
    - $t(10) : temporaries
    - $s(8) : saved temporaries (are saved through function calls)
- **Instructions: http://www.mrc.uidaho.edu/mrc/people/jff/digital/MIPSir.html**
    - Most important ones
        - i. Arithmetic : add , addi , sub , addu , addiu , subu , sll
        - ii. Memory : lw , sw, lb, sb
        - iii. Decision : beq , bne , slti
        - iv. Unconditional Branches (Jumps): j, jal , jr
- **MIPS has 2 coprocessors**
    - Coprocessor 0 for traps and memory registers (it has EPC)
    - Coprocessor 1 for Floating-point registers ($f0-$f31)
- **MIPS has alignment restriction:**
    - Words must start at addresses that are multiples of 4
    - It leads to faster data transfer

- **Instruction set: Set of instruction that the computer understands and obeys. Classified by:**
    - Number of addresses
    - Number of Operands per instructions
    - Operand location
    - Operand internal storage
    - Operand Type/Size
    - Operations

- **Types of Operand internal storages in the CPU**
  - Stack-based architecture
    1. Stack of variables - Operations only on the top two elements
    2. Simple model of expression evaluation
    3. Stack can't be randomly accessed
  - Accumulator-based architecture
    1. 1 register, two instruction types: Data transfer/ALU
    2. Short instructions
    3. Memory traffic is high
  - Register-Set based architecture (Most used today):
    1. Has general-purpose registers (GPR)
    2. Allow fast access to temporary variables
    3. Permits clever compiler optimizations
    4. Reduce traffic to memory
    5. longer instructions
    6. ex. MIPS
- **Types of ISA's by complexity:**

	CISC	**RISC**
# Instructions	More (Specialized)	Less(Frequent)
# Addressing modes	More	Less
Complexity lies in	Microprogram	Compiler
Pipelining is	Hard	Easy
Emphasize on	Hardware	Software

- **Design Principles**
  - Simplicity favors regularity
    1. Regularity -> simpler implementation
    2. Simplicity -> higher performance at lower cost
  - Smaller is faster (bits/gates/transistors)
  - Make the common case fast (avoid memory access)
  - Good design demands good compromises (Different format, complicate decoding)

- **LSB (Least-significant bit) - on the right**
- **MSB (Most-significant bit) - on the left**
- **Little and Big endian**
  - Two ways of storing multibyte data-types
  - Example Storing 0x01234567 using:
    1. Little: LSB is stored first 76543210
    2. Big : MSB is stored first 01234567
       - Assuming memory address increases from left to right
- **Stack frame is used to**
  - Store function calls
  - Store caller and callee save register
  - Hold local variables and extra arguments or return values

- **Calling Conventions:**
  - Standardized method for functions to be implemented and called by the machine.

- **Register Conventions:**
  - A set of generally accepted rules as to which registers will be unchanged after a procedure call (jal) and which may be changed.

- **Spilling registers:**
  - keep important data in registers by storing them in memory before function call and restoring them back afterward
- **Function Execution steps:**
  - Place parameters in registers/memory
  - Transfer control to function
  - Acquire storage for local variables used in a function
  - Perform function operations
  - Place result in register/memory for the caller
  - Return to the place of call
- **Macros:**
  - Like functions but operate differently
  - It does not require the protocols and execution overhead of functions.
  - Example:

    ```
    macro done
    li $v0 ,10
    syscall
    .end_macro
    ```
  - Usage: just write "done"
- **Exception:**
  - A change in execution caused by a condition that occurs within the processor. It is logical issue not related to the hardware.
- **Interrupt:**
  - A change in execution caused by an external event
- **Trap:**
  - A user-requested exception
- **Mask:**
  - Data that is used for bitwise operations, particularly in a bit field.
  - Using mask you can set bits on/off/invert them in a single bitwise operation
- **MIMD:**
  - Having a number of processors that function asynchronously and independently, but in parallel.
- **SIMD:**
  - All parallel units share the same instruction on different data elements.
- **Types of processors**
  - **Single-cycle:**
    i. Executes each instruction in one cycle.
  - **Multicycle:**
    i. Allows faster operations to take less time than slower ones, so overall performance can be increased
  - **Pipelined:**
    i. Maximize the usage of the available resources by overlapping the execution of several tasks.
    ii. Each instruction is broken up into series of steps, multiple instructions execute at once
    iii. Makes latency slightly worse

        iv. Improves throughput
        v. Pipeline diagram:
            1. # Pipeline cycles = number of column
            2. Pipeline depth = number of stages
            3. Pipeline is always either filling/full/emptying
        vi. Pipeline provide the ability to handle exceptions, save state and restart without affecting program execution

- **Register Transfer Language (RTL):**
    - intermediate representation that is very close to assembly language.
    - used to describe data flow at the register-transfer level of an architecture.
- Control Signal
    - A signal used for multiplexor selection or for directing the operation of a functional unit
- Data signal
    - Contains information that is operated on by a functional unit.

- **Hazards: Instructions interfering with each other in a pipelined processor**
- **Structural Hazards:**
    - Description: When two instructions need the same hardware component at the same time
    - Solutions
        i. Stalling: Delay the second operation by one cycle
        ii. Separate memories for instructions and data "Harvard"
- **Data Hazards:**
    - Description: Attempting to use data before it's ready
    - Solutions: Forwarding/Stalling/Compiler Scheduling
        i. Forwarding: doing the read in the second half of the clock cycle and the write in the first half
        ii. Stalling: adding extra cycle in pipeline to wait until results are ready to avoid hazards
        iii. Compiler Scheduling: Allowing the compiler to rearrange the code smartly to avoid hazards
    - Types: RAW, WAW, WAR
- **Control Hazard:**
    - Description: Attempting to make branching decisions before branch condition is evaluated
    - Solutions: Stalls/Branch predictions(Most used today)
        i. Static Branch Prediction: Assume there will be no branch
        ii. Dynamic Branch Prediction: Statistics are kept at runtime to determine the likelihood of a branch being taken.

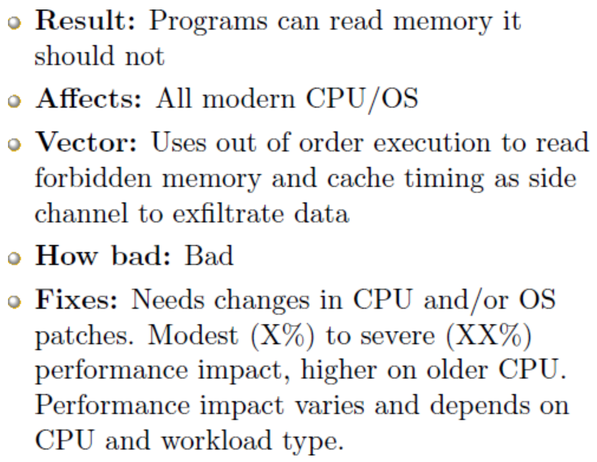
- **Out-of-order execution:** Execute instructions based on “data flow" rather than program order
- **A superscalar processor** can fetch, decode, execute and write back 2 instructions in parallel

- **Vulnerability: A weakness in a system.**
    - Allow programs to steal data which is currently processed on the computer.
- **Exploit: An attack that leverages vulnerability.**
- **A cold boot attack is a type of side channel attack in which an attacker with physical access to a computer performs a memory dump of a computer's random access memory by performing a hard reset of the target machine.**

- **Some Security Vulnerabilities that allow attacker to access sensitive data**
  - **Foreshadow: Speculative execution attack**
  - **Meltdown: (On intel chips)**
    1. Largely begin fixed with software patches
    2. Code explanation:
       1. Inaccessible kernel address is moved to a register, which raises an exception, program is moving to EPC but other instructions are already executed as of out-of-order execution, leaking the content of kernel address through the indirect memory access.
  - **Spectre**

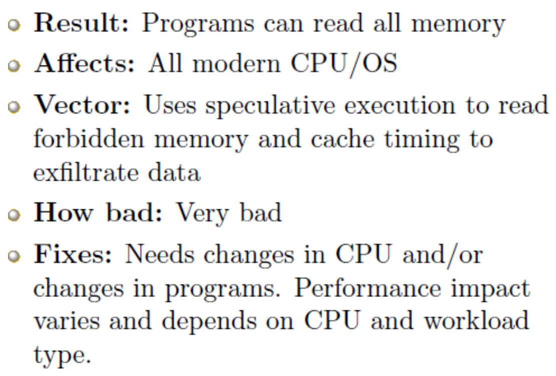
    i. Harder to exploit but can't be avoid avoided

MELTDOWN:

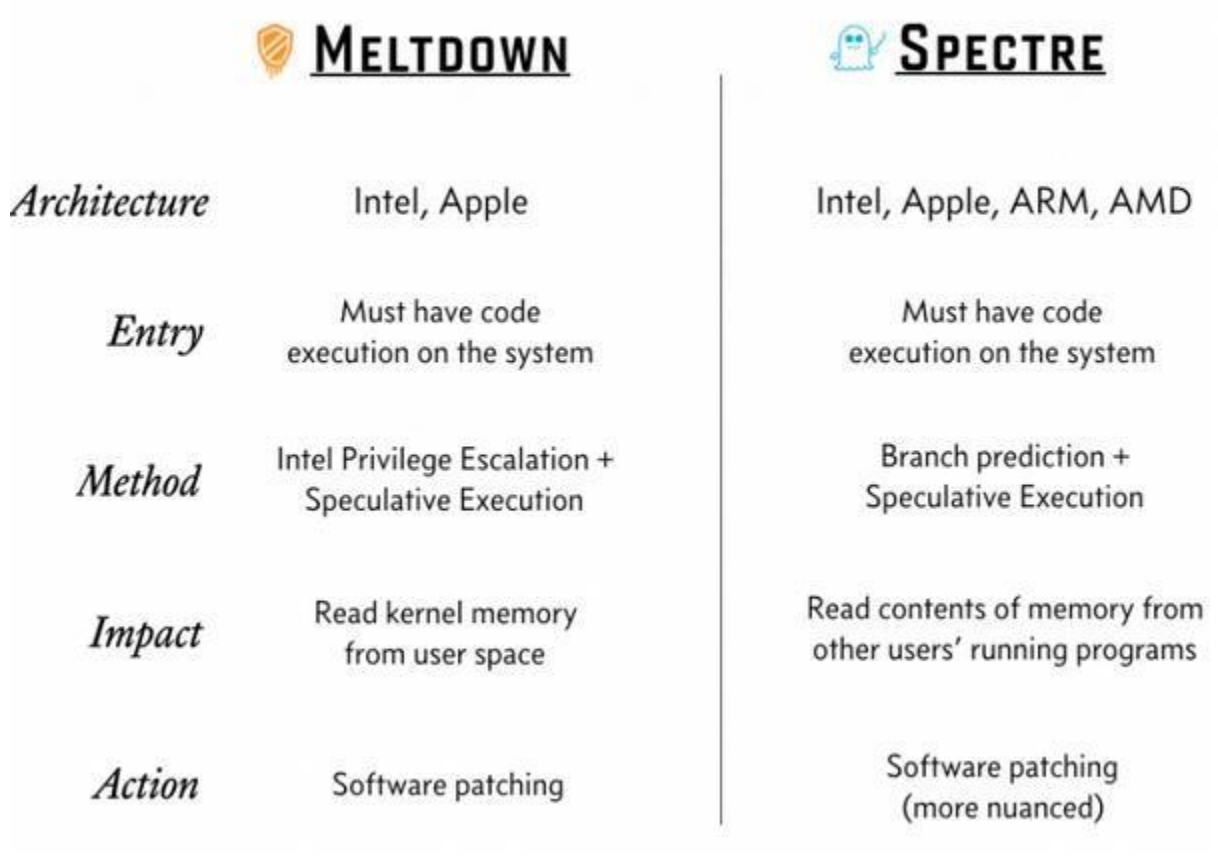


- **Result:** Programs can read memory it should not
- **Affects:** All modern CPU/OS
- **Vector:** Uses out of order execution to read forbidden memory and cache timing as side channel to exfiltrate data
- **How bad:** Bad
- **Fixes:** Needs changes in CPU and/or OS patches. Modest (X%) to severe (XX%) performance impact, higher on older CPU. Performance impact varies and depends on CPU and workload type.

SPECTRE:



- **Result:** Programs can read all memory
- **Affects:** All modern CPU/OS
- **Vector:** Uses speculative execution to read forbidden memory and cache timing to exfiltrate data
- **How bad:** Very bad
- **Fixes:** Needs changes in CPU and/or changes in programs. Performance impact varies and depends on CPU and workload type.

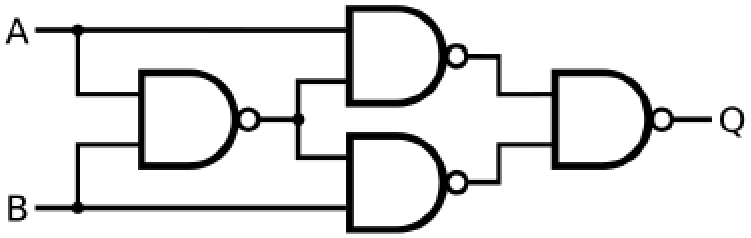


	MELTDOWN	SPECTRE
Architecture	Intel, Apple	Intel, Apple, ARM, AMD
Entry	Must have code execution on the system	Must have code execution on the system
Method	Intel Privilege Escalation + Speculative Execution	Branch prediction + Speculative Execution
Impact	Read kernel memory from user space	Read contents of memory from other users' running programs
Action	Software patching	Software patching (more nuanced)

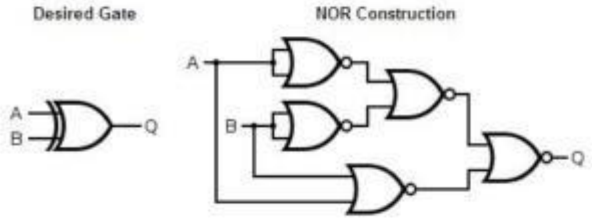
- **FLOPS: Floating-Point Operations Per Second - Measurement unit for algorithms**
- **Memory Bandwidth: rate of (R/W) operations per seconds - byte/second**
- **GPU Bandwidth: rate of GPU operations - GB/second -- FASTER**
- **SIMT: Single Instruction Multiple Threads - NVIDIA way in parallelism**
- **Elements of short vectors processed in parallel**
- **In SIMT but not in SIMD:**
  - Multiple Register-sets
  - Multiple Address
  - Multiple flow paths
- **SIMD: Single instruction Multiple Data**
  - Instructions of several threads run in parallel
- **SMT - Simultaneous Multi-Threading**
  - Somewhere in between
- **Kernel Grid (contains) Thread Blocks (contains) Thread**
- **GPU Unit (contains) Streaming Multiprocessor (contains) Cores**
  - Threads may switch/exchange data/results
- **Multicore Processor:**
  - A single computing component having many cores
  - Core is a processing unit
  - Each core can have many HARDWARE THREADS
  - Logical cores: #Physical cores * #Hardware threads for each core
  - It can run multiple instructions in parallel
  - You only get the advantage of it when running many processes
- **Hardware threads:**
  - Multicore Processor have many cores, each core can have many hardware threads
  - Can run many software threads
- **Software threads:**
  - A running program is a process, each process can run one or more SOFTWARE threads
- **Latency:**
  - Time delay between the moment something is initiated, and the moment one of its effects begins or becomes detectable
- **Throughput:**
  - The amount of work done in a given amount of time
- **CPU: low latency, low throughput**
- **GPU: high latency, high throughput**
- **Locality: Programs access a small proportion of their address space at any time**
- **Temporal locality: Items accessed recently are likely to be accessed again soon**
- **Spatial locality: Items near those accessed recently are likely to be accessed soon**

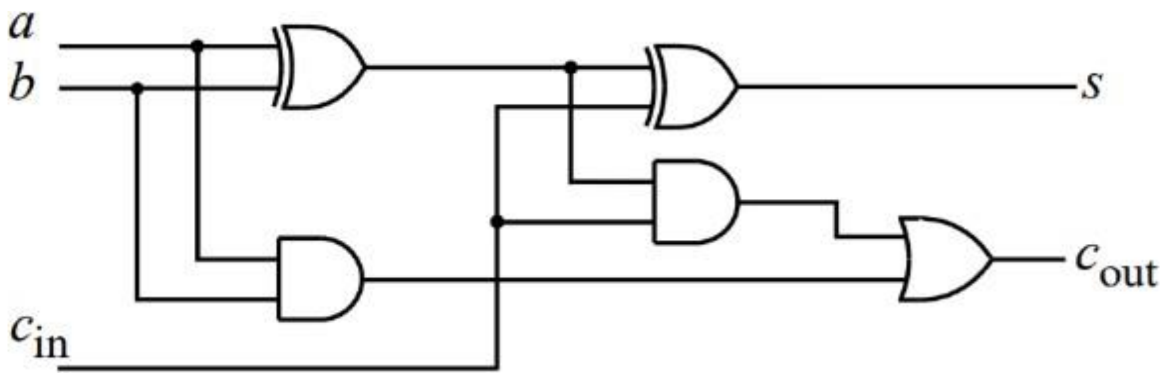
Useful Diagrams

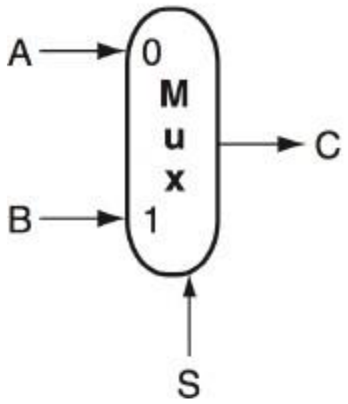
- Building of XOR gate with universal NAND gate

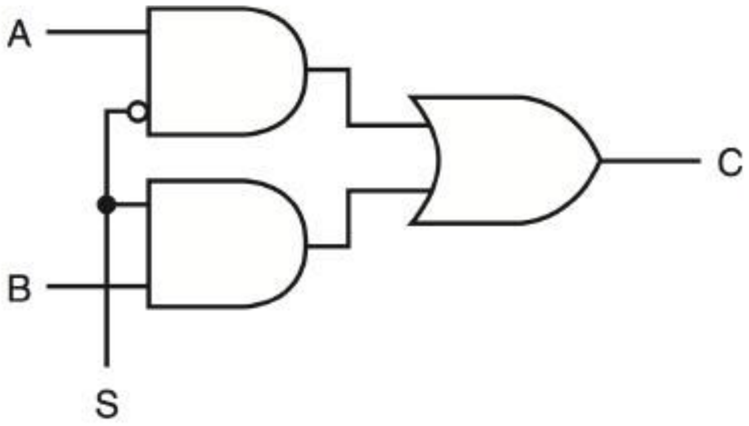


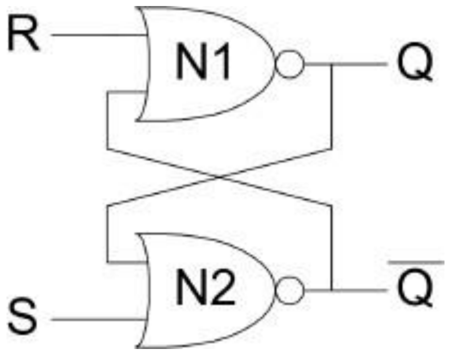
- Building of XOR gate with universal NOR gate

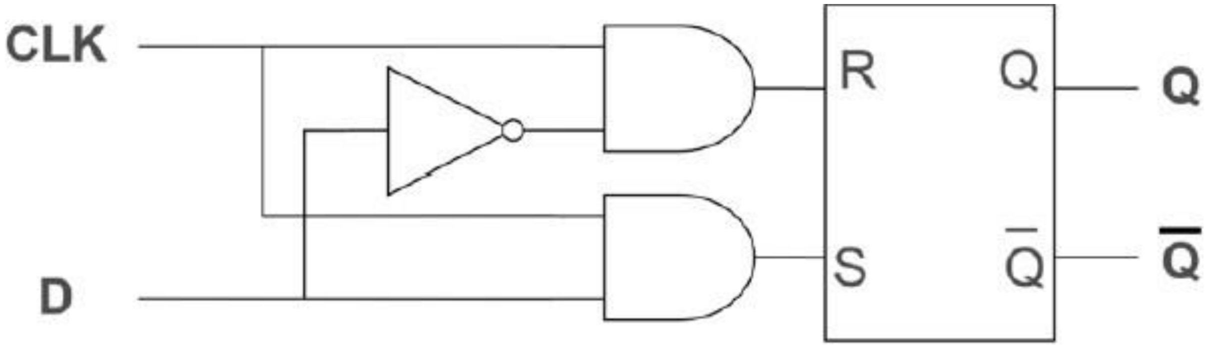


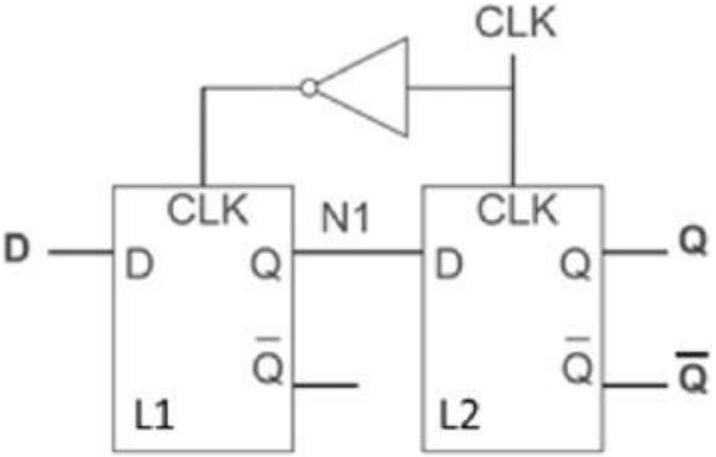


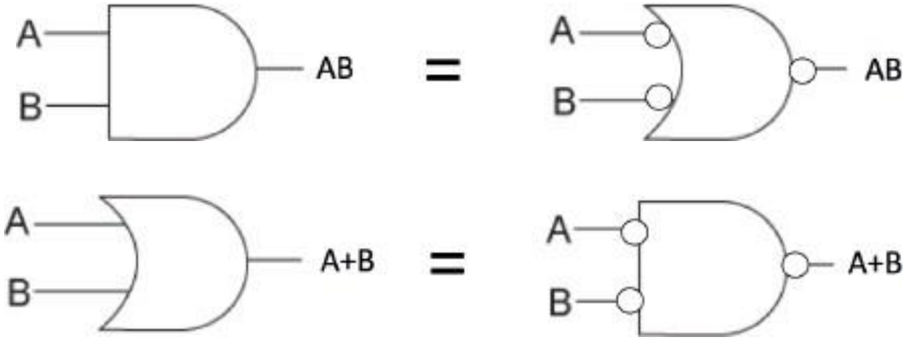

A
0
M
u
x
C
B
1
S

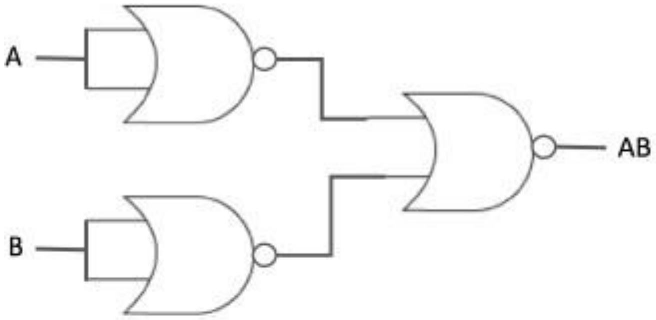

A
C
B
S


R
N1
Q
S
N2
$\overline{Q}$

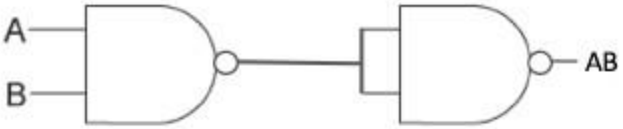

CLK
R
Q
Q
D
S
$\overline{Q}$
$\overline{Q}$


CLK
D
CLK
D
Q
$\overline{Q}$
L1
N1
CLK
D
Q
$\overline{Q}$
L2
Q
$\overline{Q}$

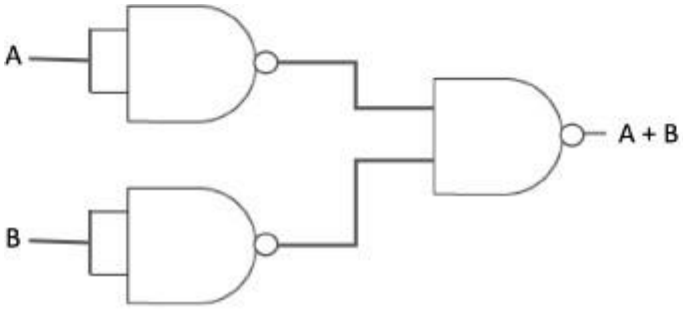

A
B
AB
=
A
B
AB
A
B
A+B
=
A
B
A+B



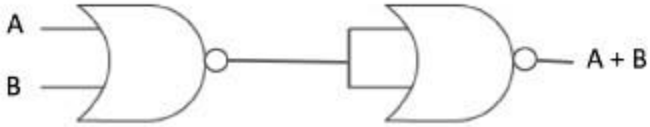
AND gate with NOR gate



AND gate with NAND gate



OR gate with NAND gate



OR gate with NOR gate