

COA Theory — Learn Fast, Explain in Viva

How to use this: Search the topic name (Ctrl+F, or Find in your phone reader), read only that topic, then say its "Say this in viva" line out loud. If you can say it without looking, you know it. Every topic is written in simple words on purpose.

1. Computer Organisation vs Computer Architecture

In one line: Architecture is what the programmer sees. Organisation is how the machine is actually built.

Understand it like this: Architecture is the menu of a restaurant — the dishes you can order. Organisation is the kitchen — how the food is actually cooked. Same menu, different kitchens.

Key points:

- Architecture: instruction set, registers the programmer can use, addressing modes, data types.
- Organisation: how the ALU is designed, bus widths, control signals, how memory is connected.
- Two computers can have the same architecture but different organisation (example: same instruction set, one faster than the other).

Say this in viva: "Architecture is the programmer's view of the computer, like its instruction set and registers. Organisation is the hardware implementation of that view, like the ALU design and bus structure."

2. Functional Units of a Computer

In one line: Every computer has five parts: input, memory, ALU, control unit, output.

Understand it like this: Input brings raw material in, memory stores it, ALU does the work, control unit is the manager giving orders, output delivers the finished product.

Key points:

- Input unit: keyboard, mouse — converts outside information into binary.
- Memory unit: stores programs and data. Primary memory (RAM) is fast and small; secondary memory (disk) is slow and big.
- ALU (Arithmetic Logic Unit): does addition, subtraction, AND, OR, shifts, comparisons.
- Control unit: fetches instructions from memory, decodes them, and sends control signals to all other units. It does not process data itself.
- Output unit: monitor, printer — converts binary back into human form.
- Bus: the set of wires that carry data, addresses, and control signals between units. Three types: data bus, address bus, control bus.

Say this in viva: "A computer has input, memory, ALU, control unit and output, connected by buses. The control unit fetches and decodes instructions and directs the other units."

3. Number Systems: Binary, Hex, BCD, ASCII

In one line: Computers only understand 0 and 1, so every other system is just a shorthand for binary.

Key points:

- Binary: base 2, digits 0 and 1. One binary digit is one bit. 8 bits make one byte.
- Hexadecimal: base 16, digits 0–9 and A–F. One hex digit equals exactly 4 bits, so hex is a short way to write binary. Example: 1010 1111 in binary = AF in hex.
- BCD (Binary Coded Decimal): each decimal digit is written in 4 bits. Example: 25 in BCD is 0010 0101. Used in calculators and digital displays.
- ASCII: a 7-bit code for characters. Example: 'A' is 65, '0' is 48 (30H in hex). It lets computers store text as numbers.

Say this in viva: "Binary is base 2 and is what hardware uses. Hex is a short form of binary, one hex digit for 4 bits. BCD stores each decimal digit in 4 bits, and ASCII is the code that stores characters as numbers."

4. Complements: 1's Complement and 2's Complement

In one line: Complements let the computer do subtraction using only addition.

Key points:

- 1's complement: flip every bit. 1 becomes 0, 0 becomes 1. Example: 0101 becomes 1010.
- 2's complement: 1's complement plus 1. Example: 0101 → 1010 → 1011.
- Negative numbers are stored in 2's complement form. That is how the computer knows a number is negative.
- Subtraction trick: $A - B = A + (2\text{'s complement of } B)$. The computer never really subtracts — it adds the complement.
- Example: $7 - 5$ becomes $7 + (-5 \text{ in } 2\text{'s complement}) = 2$.

Say this in viva: "1's complement flips all bits, 2's complement flips all bits and adds 1. Negative numbers are stored in 2's complement, and subtraction is done by adding the 2's complement of the number."

5. Signed Numbers and Overflow

In one line: The leftmost bit is the sign bit: 0 means positive, 1 means negative.

Key points:

- In an 8-bit number, the range in 2's complement is -128 to $+127$.
- Carry flag (CF) is for unsigned numbers: it becomes 1 when the result does not fit, like a carry out of the last bit.
- Overflow flag (OF) is for signed numbers: it becomes 1 when adding two positives gives a negative, or two negatives give a positive. That result is wrong.
- CF and OF are different flags for different jobs. Do not mix them up.

Say this in viva: "The sign bit is the leftmost bit. Carry flag shows overflow for unsigned numbers, and overflow flag shows overflow for signed numbers, when two same-sign numbers give an opposite-sign result."

6. ALU Operations

In one line: The ALU does arithmetic (add, subtract, multiply, divide) and logic (AND, OR, XOR, NOT, shifts).

Key points:

- ADD and SUB work on binary numbers, with carry handled bit by bit from right to left.
- AND gives 1 only when both bits are 1. Used for masking, that is, clearing selected bits.
- OR gives 1 when any bit is 1. Used for setting selected bits.
- XOR gives 1 when bits are different. Used for comparing and for toggling bits.
- NOT flips all bits.
- Shift left multiplies by 2. Shift right divides by 2. Example: 0001 0010 (18) shifted left once is 0010 0100 (36).

Say this in viva: "The ALU performs arithmetic like addition and subtraction, and logic like AND, OR, XOR, NOT and shifts. Shift left multiplies by two and shift right divides by two."

7. Booth's Multiplication Algorithm

In one line: Booth's algorithm multiplies signed binary numbers faster by skipping over long runs of 1s.

Understand it like this: Normal multiplication adds the number again and again for every 1. Booth's method notices that a run like 1111 can be handled as one subtraction and one addition instead of four additions.

Key points:

- It works for both positive and negative numbers in 2's complement.
- Look at two bits at a time: the current bit Q_0 and the previous bit Q_{-1} (which starts as 0).
- 01 means: add the multiplicand to the accumulator, then shift right.
- 10 means: subtract the multiplicand from the accumulator, then shift right.
- 00 or 11 means: only shift right, no add or subtract.
- After all bits are processed, the accumulator and multiplier together hold the answer.

Say this in viva: "Booth's algorithm multiplies signed numbers by checking pairs of bits. 01 adds the multiplicand, 10 subtracts it, 00 and 11 only shift. It is faster because it skips long runs of ones."

8. Division: Restoring vs Non-Restoring

In one line: Both divide by repeated subtraction and shifting. The difference is what happens when a subtraction goes negative.

Key points:

- Restoring division: subtract the divisor. If the result is negative, add the divisor back to restore the old value, set the quotient bit to 0, and shift. Simple, but the restore step wastes time.
- Non-restoring division: subtract the divisor. If the result is negative, do NOT restore. Instead, in the next step, add the divisor rather than subtracting. At the very end, one final correction is done if needed.
- Non-restoring is faster because it never wastes a step restoring.

Say this in viva: "In restoring division, a negative remainder is restored by adding the divisor back. In non-restoring division there is no restore step — the next step simply adds instead of subtracting, so it is faster."

9. The 8086 Processor: Registers

In one line: Registers are tiny, super-fast storage places inside the CPU itself.

Key points:

- General purpose registers: AX, BX, CX, DX. Each is 16-bit and splits into two 8-bit halves, like $AX = AH + AL$.
- AX is the accumulator, used most in arithmetic. BX is the base register, used for addressing memory. CX is the counter, used for loops and repeats. DX is the data register, used in multiplication, division and I/O.
- Pointer and index registers: SP (stack pointer), BP (base pointer), SI (source index), DI (destination index).
- Segment registers: CS (code), DS (data), SS (stack), ES (extra). They hold the starting address of a 64KB block of memory.
- Instruction pointer (IP) holds the offset of the next instruction.

Say this in viva: "8086 has general registers AX, BX, CX and DX, pointer and index registers SP, BP, SI and DI, segment registers CS, DS, SS and ES, and the instruction pointer IP."

10. Memory Segments and Physical Address

In one line: 8086 makes a 20-bit address from a 16-bit segment and a 16-bit offset.

Key points:

- Memory is divided into segments of 64KB each.
- Physical address = (segment $\times$ 16) + offset. In hex, that means: write the segment, add a zero at the end, then add the offset.
- Example: CS = 1234H, IP = 0020H. Physical address = 12340H + 0020H = 12360H.
- CS:IP points to the next instruction. DS:SI or DS:DI points to data. SS:SP points to the top of the stack.

Say this in viva: "The physical address is segment multiplied by 16 plus offset. For example, CS 1234H and IP 0020H give 12360H. This is how 16-bit registers address 1MB of memory."

11. Flag Register

In one line: Flags are single bits that record what happened in the last operation.

Key points:

- CF (Carry Flag): 1 when there is a carry out of the highest bit. Unsigned overflow.
- ZF (Zero Flag): 1 when the result is zero.
- SF (Sign Flag): copies the leftmost bit of the result. 1 means the result is negative.
- OF (Overflow Flag): 1 when a signed result is wrong, as in topic 5.
- PF (Parity Flag): 1 when the result has an even number of 1 bits.

- AF (Auxiliary Carry Flag): carry from bit 3 to bit 4, used in BCD arithmetic.
- DF (Direction Flag): controls string instructions. DF = 0 means forward, DF = 1 means backward.
- IF (Interrupt Flag): 1 means the CPU accepts maskable interrupts.
- TF (Trap Flag): 1 means single-step mode, used in debugging.

Say this in viva: "Flags record the result of the last operation: carry for unsigned overflow, zero for a zero result, sign for a negative result, overflow for a wrong signed result, plus direction and interrupt flags for control."

12. BIU and EU in 8086

In one line: 8086 is split into two workers: one fetches instructions, one executes them, so both happen at the same time.

Key points:

- BIU (Bus Interface Unit): fetches instructions from memory, computes physical addresses, and fills a 6-byte instruction queue. It handles all bus activity.
- EU (Execution Unit): takes instructions from the queue, decodes them, and executes them using the ALU and registers. It never touches the bus directly.
- Because fetching and executing overlap, the CPU wastes less time waiting. This is an early form of pipelining.
- If the EU needs data from memory, the BIU stops fetching briefly and serves that request first.

Say this in viva: "The BIU fetches instructions and manages the bus, while the EU decodes and executes them. They work in parallel, so fetching and execution overlap and the processor is faster."

13. Addressing Modes

In one line: An addressing mode tells the CPU where to find the data for an instruction.

Key points:

- Immediate: the data is in the instruction itself. Example: MOV AX, 5.
- Register: the data is in a register. Example: MOV AX, BX.
- Direct: the instruction gives the memory address. Example: MOV AX, [2000H].
- Register indirect: a register holds the memory address. Example: MOV AX, [BX].
- Indexed: an index register plus a displacement gives the address. Example: MOV AX, [SI + 10H].
- Based indexed: base plus index plus displacement. Example: MOV AX, [BX + SI + 10H].

Say this in viva: "Addressing modes say where the operand is: immediate means in the instruction, register means in a register, direct means the address is given, indirect means a register holds the address, and indexed adds a displacement."

14. Instruction Cycle: Fetch, Decode, Execute

In one line: Every instruction goes through three steps: fetch it, understand it, do it.

Key points:

- Fetch: the address in PC is sent to memory, the instruction comes back into the instruction register, and PC moves to the next instruction.
- Decode: the control unit reads the instruction and works out what operation it is and where its data lives.
- Execute: the operation is performed — the ALU computes, or data moves between registers and memory, and the result is stored.
- Then the cycle repeats for the next instruction. Interrupts are checked between cycles.

Say this in viva: "The instruction cycle has three steps: fetch the instruction using the program counter, decode it in the control unit, and execute it in the ALU or by moving data."

15. RISC vs CISC

In one line: RISC keeps instructions few and simple. CISC allows many complex instructions.

Key points:

- RISC: few instructions, all simple, all the same length, executed in one cycle each. Only load and store touch memory. Many registers. Control is hardwired. Easy to pipeline.
- CISC: many instructions, including complex ones that do several jobs at once. Instructions have different lengths. Memory can be used directly in operations. Control is usually microprogrammed.
- Example of RISC thinking: ARM processors in phones. Example of CISC thinking: classic x86 processors in PCs.
- Trade-off: RISC shifts work to the compiler, which uses more simple instructions. CISC makes assembly shorter but hardware more complex.

Say this in viva: "RISC uses few simple fixed-length instructions and load-store architecture, which pipelines well. CISC uses many complex variable-length instructions that can work directly on memory, at the cost of more complex hardware."

16. Control Unit: Hardwired vs Microprogrammed

In one line: Hardwired control is built from logic gates. Microprogrammed control is built from tiny programs stored in memory.

Key points:

- Hardwired: control signals come from combinational logic circuits. It is fast, but changing the instruction set means rewiring hardware. Suits RISC.
- Microprogrammed: each machine instruction is carried out by a sequence of microinstructions stored in a control memory. Changing behaviour means changing the microprogram, not the hardware. Suits CISC.
- Hardwired is faster. Microprogrammed is more flexible and easier to design for complex instruction sets.

Say this in viva: "Hardwired control generates signals with logic circuits, so it is fast but inflexible. Microprogrammed control uses microinstructions from control memory, so it is slower but flexible and easier to change."

17. Memory Hierarchy and Locality

In one line: Fast memory is small and costly, big memory is slow and cheap — so computers use layers of both.

Understand it like this: Keep the books you use daily on your desk (small, fast to reach), the rest in a cupboard (bigger, slower), and the rarely used ones in a store room (huge, slowest). The computer does the same with registers, cache, RAM and disk.

Key points:

- Order from fastest to slowest: registers, cache, main memory (RAM), secondary memory (disk).
- Locality of reference is why the hierarchy works: programs reuse the same data and nearby data again and again.
- Temporal locality: if you used something just now, you will probably use it again soon.
- Spatial locality: if you used something, you will probably use the things stored next to it soon.
- Cache works because of locality: it keeps copies of recently used memory close to the CPU.

Say this in viva: "Memory is organised in layers from fast and small to slow and large: registers, cache, RAM and disk. It works because of locality — programs reuse recent data and nearby data."

18. Cache Memory and Mapping

In one line: Cache is small, fast memory between the CPU and RAM that holds copies of data the CPU is likely to need.

Key points:

- Hit: the data is found in cache. Fast. Miss: it is not there, so it must be fetched from RAM. Slow.
- Miss penalty: the extra time a miss costs. $\text{Average access time} = \text{hit time} + (\text{miss rate} \times \text{miss penalty})$.
- Direct mapping: each memory block has exactly one fixed place in cache, decided by its index bits. Simple and cheap, but two blocks can fight for the same place.
- Associative mapping: a block can go anywhere in cache. No fighting, but searching the whole cache is costly.
- Set-associative mapping: the middle way. Cache is divided into sets, a block goes to one set but any place inside that set. Most real caches use this.
- Replacement: when a set is full, usually the least recently used block (LRU) is thrown out.

Say this in viva: "Cache holds copies of frequently used data near the CPU. Direct mapping fixes each block to one place, associative mapping allows any place, and set-associative mapping, which real systems use, allows any place within one set."

19. SRAM vs DRAM, RAM vs ROM

In one line: SRAM is fast and costly, DRAM is slow and dense. RAM loses data without power, ROM keeps it.

Key points:

- SRAM (Static RAM): built from transistors, holds data as long as power is on, no refresh needed, fast, costly, used for cache.

- DRAM (Dynamic RAM): built from one transistor plus a capacitor, the charge leaks so it must be refreshed thousands of times per second, slower, cheap, dense, used for main memory.
- RAM: read and write memory, volatile, data is lost when power goes off.
- ROM: read-only memory, non-volatile, keeps data without power. Holds startup programs like the BIOS. Variants: PROM, EPROM (erased by UV light), EEPROM (erased electrically).

Say this in viva: "SRAM is fast, needs no refresh and is used for cache, while DRAM is dense and cheap but needs constant refresh and is used for main memory. RAM is volatile, ROM keeps its data without power."

20. Designing Memory from Chips

In one line: Small one-bit chips are combined in a grid to build a bigger memory.

Understand it like this: Think of a grid with 4 rows and 4 columns: 4 rows are the 4 words, 4 columns are the 4 bits of each word. That is $4 \times 4 = 16$ one-bit chips. A decoder picks one row, and the 4 chips in that row together give the 4-bit word.

Key points:

- A 1×1 chip stores 1 bit at 1 address. A 4×4 memory holds 4 words of 4 bits, so it needs $4 \times 4 = 16$ such chips.
- 4 rows need 2 address lines ($2^2 = 4$); the decoder turns those 2 lines into one selected row.
- The 4 chips in the selected row are read together, giving all 4 bits of the word at once.
- To make more locations instead of wider words, you use a decoder and chip-select lines so only one row is active at a time.

Say this in viva: "4 wide by 4 deep is 16 chips. The decoder picks the row, and the 4 chips give the word."

21. Input/Output: Programmed I/O, Interrupts, DMA

In one line: Three ways for the CPU to talk to devices: keep checking, get interrupted, or let the device work alone.

Key points:

- Programmed I/O: the CPU keeps asking the device "are you ready?" in a loop. Simple, but the CPU wastes all its time waiting.
- Interrupt-driven I/O: the CPU does other work. When the device is ready, it sends an interrupt signal, the CPU pauses, serves the device, then resumes. Much less waste.
- DMA (Direct Memory Access): a DMA controller moves large blocks of data directly between device and memory without the CPU touching every byte. The CPU only starts and ends the transfer. Fastest for bulk data like disk transfers.
- Cycle stealing: DMA briefly borrows the bus for one word at a time, so the CPU slows only slightly.

Say this in viva: "Programmed I/O wastes CPU time polling the device. Interrupt I/O lets the CPU work until the device signals it is ready. DMA moves bulk data directly to memory without the CPU, which is fastest."

22. Interrupts in 8086

In one line: An interrupt is a signal that pauses the current program so something urgent can be handled first.

Key points:

- Hardware interrupts come from devices. Software interrupts come from INT instructions in the program.
- 8086 has NMI (non-maskable, cannot be ignored) and INTR (maskable, ignored when the IF flag is 0).
- Each interrupt type has a number from 0 to 255. Its handler address is stored in the interrupt vector table, which occupies the first 1KB of memory. The address for type n is at $4 \times n$.
- When an interrupt happens, the CPU finishes the current instruction, pushes flags, CS and IP onto the stack, clears IF and TF, and jumps to the handler. IRET at the end returns to the paused program.
- Example: INT 21H is the DOS interrupt used for keyboard input and screen output in lab programs.

Say this in viva: "An interrupt pauses the current program to handle an urgent event. 8086 has maskable INTR and non-maskable NMI, handlers are found through the interrupt vector table, and IRET returns to the paused program."

23. String Instructions: MOVS, CMPS, SCAS

In one line: String instructions move or compare whole blocks of memory without writing a loop yourself.

Key points:

- MOVS moves a byte (MOVSB) or word (MOVSW) from DS:SI to ES:DI, then moves SI and DI forward or backward.
- CMPS compares two strings, byte by byte or word by word, and sets flags like a subtraction.
- SCAS compares AL (or AX) with the byte (or word) at ES:DI, used for searching a character in a string.
- REP repeats the instruction CX times. REPE/REPZ repeats while equal, REPNE/REPNZ repeats while not equal.
- DF (direction flag) decides the direction: CLD clears DF so SI and DI increase (forward), STD sets DF so they decrease (backward).

Say this in viva: "MOVS moves a string from DS:SI to ES:DI, CMPS compares two strings, SCAS searches a string, REP repeats the operation CX times, and the direction flag decides forward or backward movement."

24. Pipelining and Hazards

In one line: Pipelining lets the CPU work on several instructions at once, like an assembly line. Hazards are the jams in that line.

Key points:

- An instruction is split into stages, like fetch, decode, execute, memory access, write-back. While one instruction is executing, the next is being decoded and another fetched.
- Ideal speedup equals the number of stages, but hazards reduce it.
- Structural hazard: two stages need the same hardware at the same time, like one memory for both instructions and data.
- Data hazard: an instruction needs a result that an earlier instruction has not finished yet.
Example: ADD R1, R2, R3 followed by SUB R4, R1, R5.
- Control hazard: a branch changes the flow, so instructions already fetched may be wrong and must be thrown away. Branch prediction tries to guess the right path.

Say this in viva: "Pipelining overlaps instruction stages like an assembly line for speedup. Structural hazards are resource conflicts, data hazards are missing results, and control hazards come from branches."

25. Quick Revision: One-Line Answers

- Organisation vs architecture: architecture is the programmer's view, organisation is the hardware build.
- ALU: does arithmetic and logic.
- 2's complement: flip bits and add 1; used for negative numbers and subtraction.
- Booth's: signed multiplication that skips runs of ones.
- Restoring division restores a negative remainder; non-restoring does not.
- Physical address: $\text{segment} \times 16 + \text{offset}$.
- Cache: direct fixes one place, associative allows any place, set-associative is the practical middle way.
- DMA: device to memory transfer without the CPU touching every byte.
- Interrupt: pause current work, handle the urgent event, return with IRET.
- RISC: few simple instructions. CISC: many complex instructions.
- Hardwired control is fast and fixed; microprogrammed control is flexible and stored.

Built for the COA practical/viva: short topics, simple words, one viva line each. Search any topic name to jump straight to it.