IAE ACADEMY INSTITUTE FOR ENGINEERS

GATE CS

IAE Academy free study material · Updated 29 September 2026

Computer Organization and Architecture Formula Sheet

INSIDE

- 46 formulas in 5 topics
- When to use each one
- Taken from the free topic notes on iaeacademy.in

Online, one page per topic: **iaeacademy.in/study/computer-organization-architecture/formula-sheet**

FREE COUNSELLING

Preparing for GATE CS? Talk to IAE Academy.

Or call **+91 84204 36524**

Every day, 10:30 am to 7:30 pm

Chat on WhatsApp

Free. Not for sale. Pass it on.

Instruction sets and addressing modes

BITS NEEDED TO ENCODE N DISTINCT ITEMS

$$k = \lceil \log_2 n \rceil$$

Smallest k with $2^k \geq n$. Applies to an opcode field (n instructions), a register field (n registers) or an address field (n addressable locations).

EXPANDING OPCODES: W-BIT INSTRUCTIONS, A-BIT ADDRESS FIELDS

$$N_1^{\max} = \left(2^{W-2A} - N_2\right) 2^A, \qquad N_0^{\max} = \left[\left(2^{W-2A} - N_2\right) 2^A - N_1\right] 2^A$$

N_2, N_1, N_0 are the numbers of two-, one- and zero-address instructions. The two-address format has a $(W - 2A)$-bit opcode; each unused two-address code gives 2^A one-address codes, and each unused one-address code gives 2^A zero-address codes. Assumes every instruction is exactly W bits, every address field is A bits, and $W > 2A$.

DISPLACEMENT-TYPE ADDRESSING

$$\mathrm{EA} = (R) + D$$

R is the index register (indexed), the base register (base) or the PC (PC-relative); D is the displacement field. For PC-relative, D is signed.

PC-RELATIVE BRANCH TARGET

$$\mathrm{Target} = PC_{\text{after fetch}} + D$$

$PC_{\text{after fetch}}$ is the address of the instruction AFTER the branch (branch address + branch length), unless a question says the branch's own address is used. D is the signed displacement in bytes, unless the ISA scales it.

RANGE OF AN N-BIT TWO'S-COMPLEMENT DISPLACEMENT

$$-2^{n-1} \leq D \leq 2^{n-1} - 1$$

Range in displacement units relative to the base PC. If the displacement counts instructions of s bytes, multiply by s to get the byte range.

ALU design and the control unit: hardwired and microprogrammed

FULL ADDER EQUATIONS

$$S_i = A_i \oplus B_i \oplus C_i, \qquad C_{i+1} = A_i B_i + B_i C_i + A_i C_i$$

One bit-slice of any adder. Both outputs need all three inputs to be stable.

SUBTRACTION AND OVERFLOW (2'S COMPLEMENT)

$$A - B = A + \overline{B} + 1 \ (\text{use } C_0 = 1), \qquad V = C_n \oplus C_{n-1}$$

An adder/subtractor passes each B_i through an XOR gate with a mode line M and sets $C_0 = M$: $M = 0$ adds and $M = 1$ subtracts. $V = 1$ signals signed (2's complement) overflow. For **unsigned** numbers, $C_n = 1$ after an addition means the result does not fit in n bits. After a subtraction done as $A + \overline{B} + 1$, $C_n = 1$ means $A \geq B$ (no borrow).

RIPPLE-CARRY ADDER WORST-CASE DELAY

$$T_{C_n} = n\,t_c, \qquad T_{S_{n-1}} = (n-1)\,t_c + t_s$$

t_c is one FA's carry delay and t_s is its sum delay, measured from when its carry-in is stable (its A_i, B_i are already stable). If one figure t_{FA} is given for both outputs, both times equal $n\,t_{FA}$. The delay grows linearly with n.

GENERATE, PROPAGATE AND SUM

$$G_i = A_i B_i, \qquad P_i = A_i \oplus B_i, \qquad S_i = P_i \oplus C_i$$

$G_i = 1$: bit i produces a carry whatever C_i is. $P_i = 1$: bit i passes an incoming carry on to C_{i+1}. Some texts use $P_i = A_i + B_i$. The carry equations below still hold with that choice, but the sum can then no longer be formed as $P_i \oplus C_i$.

CARRY RECURRENCE AND ITS LOOK-AHEAD EXPANSION

$$C_{i+1} = G_i + P_i C_i$$
$$C_1 = G_0 + P_0 C_0$$
$$C_2 = G_1 + P_1 G_0 + P_1 P_0 C_0$$
$$C_3 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$$
$$C_4 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_0$$

Each expanded carry is a two-level AND-OR expression in the G's, P's and C_0, so every carry is ready **2 gate delays after the P, G signals**, whatever its position. The cost is fan-in: C_i has $i+1$ product terms and its largest AND gate has $i+1$ inputs.

SINGLE-LEVEL CLA DELAY (CONVENTION ABOVE)

$$T_{\text{CLA}} = \underbrace{1}_{P_i,\,G_i} + \underbrace{2}_{\text{all carries}} + \underbrace{1}_{S_i = P_i \oplus C_i} = 4 \text{ gate delays}$$

This is independent of n only because fan-in is assumed unlimited. If a question counts an XOR as more than one gate delay, substitute its figure in the first and last terms.

BLOCK (GROUP) GENERATE AND PROPAGATE FOR A 4-BIT BLOCK

$$G^* = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0, \qquad P^* = P_3 P_2 P_1 P_0, \qquad C_4 = G^* + P^* C_0$$

Fan-in limits force wide adders to be built from k-bit CLA blocks. Each block's G^*, P^* take 2 gate delays after the bit-level P_i, G_i. A **second-level look-ahead unit** applies the same carry equations to the block signals, giving the carry into every block at once instead of rippling between blocks. With a fixed block size k, the number of look-ahead levels grows as $\log_k n$, so the delay grows as $O(\log n)$.

CONTROL MEMORY SIZE

$$\text{Size (bits)} = W \times w, \qquad w = (\text{control field bits}) + (\text{condition-select bits}) + (\text{next-address bits})$$

W is the number of words (microinstructions) and w is the width of one microinstruction. When a question gives the number of address bits a of the CAR instead, $W = 2^a$. Convert to bytes by dividing by 8, and to KB with K = 2^{10}.

NEXT-ADDRESS AND CONDITION-SELECT FIELD WIDTHS

$$\text{next-address bits} = \lceil \log_2 W \rceil, \qquad \text{condition-select bits} = \lceil \log_2 c \rceil$$

W is the number of control-memory words that must be addressable. c is the number of branch choices the field must select among, counting 'unconditional' (or 'no branch') as a choice when the question includes it.

CONTROL-FIELD WIDTH: HORIZONTAL VS ENCODED

$$w_{\text{horizontal}} = \sum_j k_j, \qquad w_{\text{encoded}} = \sum_j \lceil \log_2 (k_j + 1) \rceil$$

Group j holds k_j mutually exclusive signals. The $+1$ is the code for 'none of this group active'. Drop it (use $\lceil \log_2 k_j \rceil$) only if the question says exactly one signal of the group is always active, or says no-operation need not be encoded.

Memory hierarchy, performance and cache mapping

HIT RATIO

$$h = \frac{N_{hit}}{N_{hit} + N_{miss}}$$

The fraction of memory references satisfied directly by the cache. Miss ratio $= 1 - h$.

AVERAGE ACCESS TIME, HIERARCHICAL (SERIAL) ACCESS

$$T_{avg} = h \cdot T_c + (1 - h) \cdot (T_c + T_m) = T_c + (1 - h) \cdot T_m$$

Use this when main memory is accessed only after the cache lookup has missed, so a miss costs $T_c + T_m$. T_m is the time to get the needed data from main memory. If the question says a whole block is transferred into the cache on a miss and gives the block-transfer time, use that time as the miss cost in place of a single-word T_m.

AVERAGE ACCESS TIME, SIMULTANEOUS ACCESS

$$T_{avg} = h \cdot T_c + (1 - h) \cdot T_m$$

Use this when the cache and main memory are looked up in parallel. On a miss, the cache-check time is hidden inside the longer memory access and is not added again.

AVERAGE ACCESS TIME FROM A STATED MISS PENALTY

$$T_{avg} = T_{hit} + (1 - h) \cdot T_{penalty}$$

The miss penalty is the EXTRA time a miss costs on top of the hit time. With hierarchical access and $T_{penalty} = T_m$, this is the same as the hierarchical formula above.

SIZE OF THE TAG DIRECTORY

$$\text{Tag memory (bits)} = (\text{number of lines}) \times (\text{tag bits} + \text{extra bits per line})$$

Number of lines = cache size ÷ block size, whatever the associativity. The extra bits per line are whatever the question specifies. Typically that is a valid bit, a dirty bit for a write-back cache, and any replacement bits.

AVERAGE MEMORY ACCESS TIME, TWO-LEVEL CACHE

$$AMAT = T_{L1} + MR_{L1}\,(T_{L2} + MR_{L2} \cdot T_{mem})$$

Model: L2 is accessed only on an L1 miss, and main memory only on an L2 miss, with each lower-level time added on top (hierarchical). MR_{L2} here is the LOCAL miss rate of L2: misses in L2 ÷ accesses that reach L2 (the L1 misses). It is not the global miss rate.

GLOBAL MISS RATE OF L2

$$MR_{L2,\,global} = MR_{L1} \times MR_{L2,\,local}$$

The fraction of ALL CPU accesses that miss in both L1 and L2 and go to main memory. If a question gives a global L2 miss rate, divide it by MR_{L1} to get the local rate before using the AMAT formula.

CHIPS NEEDED TO BUILD AN M X W MEMORY FROM M X B CHIPS

$$\text{chips} = \frac{M}{m} \times \frac{w}{b}$$

Here M and m are numbers of words, and w and b are bits per word. Each row places w/b chips side by side to form the full word width. M/m rows are stacked to supply all the words. The memory needs $\log_2 M$ address lines. Each chip takes the low $\log_2 m$ of them, and the remaining $\log_2(M/m)$ high-order bits drive a decoder that selects one row.

TIME TO FETCH AN N-WORD BLOCK FROM LOW-ORDER INTERLEAVED MEMORY WITH M >= N BANKS

$$T_{block} = T_{access} + (n - 1) \cdot T_{transfer}$$

Model: all n banks start their access together. The first word is on the bus after T_{access} (the bank access time, including placing the word on the bus). Each further word, already read in its own bank, needs only one bus transfer time $T_{transfer}$. Questions may define the timing differently, for example with a separate address-send cycle. Use the question's model when it gives one.

TIME TO FETCH AN N-WORD BLOCK, NO INTERLEAVING

$$T_{block} = n \cdot T_{access}$$

Same model with a single bank: each word is a full, separate access. This is the baseline against which the speedup from interleaving is measured.

DISK ACCESS TIME

$$T_{access} = T_{seek} + T_{rotational} + T_{transfer} \ (+ T_{controller} \text{ if given})$$

Seek: moving the head to the correct track. Rotational latency: waiting for the target sector to rotate under the head. Transfer: reading or writing the data once positioned. Add a controller overhead only if the question gives one. Unless stated otherwise, a given seek time is taken as the average seek time.

AVERAGE ROTATIONAL LATENCY

$$T_{rotational} = \frac{1}{2} \cdot \frac{60}{RPM}$$

Half the time of one revolution, in seconds (RPM = revolutions per minute). This is the AVERAGE. The worst case is a full revolution and the best case is zero.

DATA TRANSFER TIME

$$T_{transfer} = \frac{\text{bytes to transfer}}{\text{transfer rate}}, \quad \text{transfer rate} = \text{sectors per track} \times \text{bytes per sector} \times \frac{RPM}{60}$$

One track passes under the head per revolution, so the transfer rate is the track capacity times the revolutions per second. Equivalently, reading one sector takes (revolution time) ÷ (sectors per track).

I/O interface: interrupts and DMA

TOTAL INTERRUPT-SERVICING OVERHEAD FOR N INTERRUPTS

$$T_{\text{ISR,total}} = n\,(t_{cs} + t_{svc} + t_{cr})$$

n = number of interrupts serviced in the period analysed; t_{cs} = context-save time; t_{svc} = the ISR's own execution time, excluding save and restore; t_{cr} = context-restore time. Divide by the total elapsed time to get the fraction of CPU time spent.

FRACTION OF CPU TIME SPENT ON ONE DEVICE'S INTERRUPTS

$$f_{\text{CPU}} = \frac{t_{\text{int}}}{T_{\text{int}}} = r \times t_{\text{int}}$$

t_{int} = total CPU time per interrupt (save + service + restore); T_{int} = average time between interrupts; $r = 1/T_{\text{int}}$ = interrupts per second. If there is one interrupt per word, r is the device's word rate. This holds only while $t_{\text{int}} < T_{\text{int}}$. Otherwise the CPU cannot keep up and data is lost.

BURST (BLOCK) MODE DMA TRANSFER TIME

$$T_{\text{burst}} = N \times t_{\text{mem}}$$

N = number of words in the block; t_{mem} = one memory (bus) cycle time. This holds when:

- each word needs exactly one memory cycle;
- the words are available back to back (for example from the device's buffer);
- the DMAC holds the bus for the whole block;
- bus request/grant overhead is ignored.

During this time the CPU cannot use the bus. It can still carry on with internal operations that need no bus.

CPU TIME LOST TO CYCLE-STEALING DMA OVER N WORDS

$$T_{\text{CPU,lost}} \le N \times t_{\text{mem}}$$

One bus cycle is stolen per word, assuming each word needs exactly one memory access. The two sides are equal in the standard model, where the CPU would have used the bus in every stolen cycle and arbitration overhead is ignored. This is the same product as burst mode, but it means something different. In cycle stealing, the loss is spread as N separate one-cycle delays, and the total elapsed time is set by the device's word rate. In burst mode, the CPU is locked out for one continuous stretch.

FRACTION OF BUS CYCLES TAKEN BY CYCLE-STEALING DMA

$$f_{\text{stolen}} = \frac{t_c}{T_w}$$

t_c = time of the one bus cycle stolen per word; T_w = time between successive word transfers, i.e. the device's word period. This assumes the CPU would otherwise use the bus in every cycle, and $t_c \le T_w$.

Some questions give a time x to prepare a word and a time y to transfer it ($y = t_c$). If the device cannot prepare the next word during the transfer, $T_w = x + y$. If preparation overlaps the transfer, $T_w = x$ (assuming $x \ge y$). State which reading you use.

PERCENTAGE OF CPU TIME USED ON DMA SET-UP AND COMPLETION

$$\%_{\text{CPU}} = \frac{t_{\text{setup}} + t_{\text{done}}}{T_{\text{block}}} \times 100, \qquad T_{\text{block}} = \frac{\text{block size}}{\text{device data rate}}$$

t_{setup} = CPU time to program the DMAC for one block; t_{done} = CPU time for the completion interrupt. This assumes the device transfers continuously, one block after another, and ignores the bus cycles DMA steals. Add the stolen-cycle fraction separately if the question asks for it.

Instruction pipelining and pipeline hazards

PIPELINE CLOCK PERIOD

$$T_p = \max(t_1, t_2, \ldots, t_k) + t_{\text{latch}}$$

t_i is the delay of stage i and t_{latch} is the inter-stage register (latch) delay, taken as 0 only if the question says so. The slowest stage sets the clock for every stage.

TOTAL TIME FOR N INSTRUCTIONS, K STAGES (NO STALLS)

$$T(n) = (k + n - 1)\, T_p$$

The first instruction needs k cycles to pass through all the stages (pipeline fill). After that, one instruction completes every cycle. Valid for an ideal pipeline with no stalls.

SPEED-UP OVER NON-PIPELINED EXECUTION (GENERAL)

$$S = \frac{n\, t_{\text{np}}}{(k + n - 1)\, T_p}$$

t_{np} is the time one instruction takes on the non-pipelined machine, as the question defines it: usually the sum of the stage delays. As $n \to \infty$, $S \to t_{\text{np}}/T_p$.

SPEED-UP, BALANCED STAGES (SPECIAL CASE)

$$S = \frac{n\, k}{k + n - 1}$$

Only when $t_{\text{np}} = k\, T_p$, i.e. all stages have equal delay and latch delay is ignored (or the non-pipelined machine is defined as taking k pipelined cycles per instruction). Then $S \to k$ as $n \to \infty$: the ideal maximum speed-up equals the number of stages.

THROUGHPUT

$$\text{Throughput} = \frac{n}{(k + n - 1)\, T_p}$$

Instructions completed per unit time with no stalls. It approaches $1/T_p$ as $n \to \infty$.

PIPELINE EFFICIENCY

$$\eta = \frac{n}{k + n - 1}$$

Efficiency is the fraction of stage-cycles that do useful work: nk busy stage-cycles out of $k(k + n - 1)$. It is below 1 for any finite n because of fill and drain, and it tends to 1 as $n \to \infty$. It equals S/k only in the balanced-stage case, where $t_{\text{np}} = k\, T_p$.

AVERAGE CPI WITH STALLS

$$\mathrm{CPI}_{\mathrm{avg}} = \mathrm{CPI}_{\mathrm{ideal}} + \sum_{i} p_i \times \mathrm{penalty}_i$$

For a scalar in-order pipeline, $\mathrm{CPI}_{\mathrm{ideal}} = 1$. Here p_i is the fraction of all instructions that suffer hazard type i, and $\mathrm{penalty}_i$ is the number of stall cycles it costs. The sum is the average number of stall cycles per instruction. This is the steady-state figure: pipeline fill cycles are ignored.

EXECUTION TIME FROM CPI

$$T_{\mathrm{exec}} = \text{Instruction count} \times \mathrm{CPI}_{\mathrm{avg}} \times T_{\mathrm{clk}}$$

The standard performance equation. On the pipelined machine, $T_{\mathrm{clk}} = T_p$.

SPEED-UP WITH STALLS

$$\begin{aligned} S &= \frac{t_{\mathrm{np}}}{\mathrm{CPI}_{\mathrm{avg}} \times T_p} \\ &= \frac{k}{1 + \text{stall cycles per instruction}} \quad (\text{if } t_{\mathrm{np}} = k\,T_p) \end{aligned}$$

t_{np} is the non-pipelined time per instruction. The second line holds only for balanced stages with no latch overhead. Both lines ignore fill cycles, so they describe a long program.

RAW STALLS WITHOUT FORWARDING (5-STAGE, SPLIT-CYCLE REGISTER FILE)

$$\text{stalls} = \max(0,\, 3 - d)$$

d is the distance in program order: $d = 1$ for the very next instruction. This assumes the register file is written in the first half of WB and read in the second half of ID, so a read in the same cycle as the write sees the new value. The consumer's ID must then fall no earlier than the producer's WB, so $d \geq 3$ needs no stall. If the register file is not split-cycle (the read must come in a cycle after the write), the result becomes $\max(0,\, 4 - d)$.

BRANCH PENALTY (PER TAKEN OR MISPREDICTED BRANCH)

$$\mathrm{Penalty} = s_{\mathrm{resolve}} - 1$$

s_{resolve} is the number of the stage at whose end both the outcome and the target are known. The correct instruction is fetched in the next cycle. No delay slot or prediction is assumed. For example, resolving at the end of EX (stage 3) squashes the 2 instructions in IF and ID, a penalty of 2 cycles; resolving at the end of ID gives 1.

AVERAGE CPI CONTRIBUTION FROM BRANCHES

$$\Delta\mathrm{CPI} = p_{\mathrm{branch}} \times p_{\mathrm{penalised}} \times \mathrm{Penalty}$$

p_{branch} is the fraction of all instructions that are branches. $p_{\mathrm{penalised}}$ is the fraction of those branches that pay the penalty: under predict-not-taken, the taken fraction; under a predictor, the mispredicted fraction. If the pipeline stalls on every branch regardless of outcome, $p_{\mathrm{penalised}} = 1$.

FROM IAE ACADEMY

Preparing for GATE CS?

IAE Academy teaches GATE CS in Ballygunge, Kolkata, and online. Counselling is free: ask us about batches, fees or your preparation.

FREE COUNSELLING

Chat on WhatsApp

Call +91 84204 36524

Open every day, 10:30 am to 7:30 pm.

Scan to chat on WhatsApp

THE GATE CS COURSE

Classroom batches in Ballygunge and online batches. Fees and details are on the course page.

Scan: course page

See the course

FREE ON IAEACADEMY.IN

Notes and solved previous-year questions for Computer Organization and Architecture, topic by topic.

Scan: the notes

Open the notes

Found this useful? Send it to a friend preparing for GATE.

Send to a friend on WhatsApp

IAE Academy free study material. The formulas come from the free Computer Organization and Architecture notes on iaeacademy.in. Prepared and published by IAE Academy, Kolkata. Free to read and free to share, unaltered and with attribution.

IAE ACADEMY INSTITUTE FOR ENGINEERS

GATE coaching in Kolkata since 2012 · EC, EE, IN, CS and IT, DA, BT and CE · in the classroom and online

227A Rash Behari Ave, Ballygunge, Kolkata 700019

+91 84204 36524 · iaeacademy.in · Updated 29 September 2026

Get directions