

Cambridge AS & A Level · Computer Science 9618 · Notes

Hardware and virtual machines A2

Read online at pastopic.com/cambridge/computer-science-9618/notes/hardware-and-virtual-machines

Last checked 30 September 2026

Read first: [Hardware](#), [Processor fundamentals](#)

© 2026 Pastopic. Free for personal study. Please share the link, not the file.

What you need to know

This is the A Level topic (Paper 3). Logic gates, drawing circuits and writing truth tables start in the AS topic **Hardware**; registers, the fetch-execute cycle and interrupts start in **Processor fundamentals**. By the end of this topic you should be able to:

Processors, parallel processing and virtual machines

1. **Explain the differences between RISC and CISC processors**, and how each handles an **interrupt**.
2. **Explain why pipelining and registers matter in RISC processors**, and how pipelining works.
3. **Describe the four basic computer architectures**: SISD, SIMD, MISD and MIMD.
4. **Describe the characteristics of massively parallel computers**.
5. **Explain what a virtual machine is**, give examples of what virtual machines are used for, and give their **benefits and limitations**.

Boolean algebra and logic circuits

6. **Produce truth tables for logic circuits**, including **half adders** and **full adders**, and gates with more than two inputs.
 7. **Explain SR and JK flip-flops**: draw the logic circuit of a flip-flop, derive its truth table, and explain how flip-flops store data.
 8. **Use Boolean algebra**, including **De Morgan's laws**, to simplify an expression or a logic circuit.
 9. **Use Karnaugh maps (K-maps)** to solve logic problems, and explain why they are useful.
-

Understand it

RISC and CISC

A processor can only carry out the instructions in its **instruction set**. There are two ways to design that set.

A **CISC** (Complex Instruction Set Computer) processor has a **large** set of **complex** instructions. One instruction can do a lot, such as "fetch two values from memory, multiply them and store the result". This keeps programs short and makes the compiler's job easier, but the hardware has to do more:

- instructions have **different lengths** and **many formats**, with **many addressing modes**;
- one instruction takes **several clock cycles**, because the control unit breaks it into smaller steps (often using a **microprogrammed** control unit);
- the circuits are **complex**, and the processor uses **cache** a lot to avoid going back to main memory.

A **RISC** (Reduced Instruction Set Computer) processor has a **small** set of **simple** instructions. A complex job becomes several simple instructions, so programs are longer and the design effort moves to the **software** (the compiler). In return:

- instructions have a **fixed length** and **few formats**, with **few addressing modes**;
- each instruction takes **one clock cycle**;
- the control unit is **hard-wired** (fast circuits, not a stored microprogram), and the chip needs fewer transistors;
- only **load** and **store** instructions touch memory; everything else works on **many general-purpose registers**;
- because every instruction is the same size and takes the same time, it is ideal for **pipelining**.

Pipelining

Every instruction goes through the same stages. In the version used in this course there are five:

Stage	What happens
IF	instruction fetch
ID	instruction decode
OF	operand fetch
IE	instruction execute
WB	write back the result

Without pipelining, one instruction finishes all five stages before the next begins, so most of the processor sits idle. **Pipelining** splits the processor into five units, one per stage, each with its own registers to hold its instruction between cycles. Each stage takes one clock cycle, and as soon as instruction 1 moves from IF to ID, instruction 2 enters IF. Several instructions are being processed at the same time, each at a different stage, like a production line. No two instructions are ever at the same stage in the same cycle.

For six instructions A to F:

Stage	1	2	3	4	5	6	7	8	9	10
IF	A	B	C	D	E	F				
ID		A	B	C	D	E	F			
OF			A	B	C	D	E	F		
IE				A	B	C	D	E	F	

Stage	1	2	3	4	5	6	7	8	9	10
WB					A	B	C	D	E	F

The first instruction takes 5 cycles; after that one instruction finishes every cycle. So n instructions need $n + 4$ cycles instead of $5n$: here 10 instead of 30. Each instruction takes just as long as before, but the **throughput** (instructions completed per second) goes up.

This is why RISC and pipelining go together: fixed-length, single-cycle instructions move through the stages in step. CISC instructions of different lengths and times would jam the pipeline. The many **registers** in a RISC processor also help: the pipeline stages need registers to hold each instruction's values between stages, and keeping data in registers means few slow trips to memory.

Interrupts

An **interrupt** is a signal (from hardware, such as a printer or a timer, or from software, such as a division by zero) asking for the processor's attention. On both RISC and CISC processors:

1. At the **end (or start) of each fetch-execute cycle**, the processor checks for an interrupt.
2. If there is one of higher priority than the current task, the current program is **suspended**: the contents of the PC and the other **registers are saved on the stack**.
3. The address of the right **Interrupt Service Routine (ISR)** is loaded into the PC, and the ISR runs.
4. When the ISR finishes, the saved values are **restored from the stack** and the program carries on where it stopped.

With pipelining this is harder, because when the interrupt arrives up to five instructions are part-way through. The usual answer is: **discard** all the instructions in the pipeline **except the one at the write-back stage**, let that one finish, handle the interrupt, then **restart** from the next instruction in the sequence (the discarded ones are fetched again). Another way is to save the state of each stage in extra registers so the pipeline can carry on afterwards.

The four computer architectures

Computers are grouped by how many **instruction streams** and how many **data streams** they handle at once.

Architecture	Meaning	What it looks like	Used for
SISD	Single Instruction, Single Data	one processor (with its control unit and memory) runs one instruction at a time on one piece of data, sequentially	older single-core computers
SIMD	Single Instruction, Multiple Data	many processors (an array processor) all carry out the same instruction at the same time, each on different data	graphics cards, sound and image processing (brighten every pixel at once)

Architecture	Meaning	What it looks like	Used for
MISD	Multiple Instruction, Single Data	several processors each carry out different instructions on the same data	rare; checking the same data in different ways, such as fault-tolerant flight control
MIMD	Multiple Instruction, Multiple Data	many processors, each running different instructions on different data, working independently (asynchronously)	multi-core PCs, supercomputers

SIMD, MISD and MIMD are all **parallel** architectures: more than one processor works at the same time.

Massively parallel computers

A **massively parallel computer** links a **very large number of processors** (thousands, often whole separate computers, each with its own memory). They work **at the same time on coordinated parts of one large problem**, such as a weather forecast. They are connected by a **network infrastructure** and **communicate by passing messages**. The program has to be written so the problem can be split into parts that run in parallel.

Virtual machines

A **virtual machine (VM)** is **software that emulates a computer system**: it behaves like a separate machine with its own processor, memory and devices, but it runs on a real computer called the **host**.

- The **host operating system** is the normal operating system of the real computer. It **controls the real hardware** and all its resources, and it runs the virtual machine software (the **hypervisor**) and gives you an interface to it.
- The **guest operating system** runs **inside** the virtual machine. It controls the **virtual** hardware, reaches the real hardware only through the virtual machine software and the host, and runs under the host's control. One host can run several guests at once, even different operating systems.

Examples of their use: running software written for another operating system or instruction set; running old (legacy) software on new hardware; trying out a new computer system before its hardware is bought; testing untrusted software or new code safely; running several servers on one physical machine.

Benefits	Limitations
Compatibility: software that won't run on the host can run on a suitable guest; old software can be kept going	Performance: the guest runs slower than on its own hardware, because the extra layer of code uses processor time and memory; it also depends on how powerful the host is
Protection: a crash or a virus in the guest does not affect the host or other VMs, so it is safe for testing	Timing and performance cannot be measured accurately on a VM

Benefits

Cost: no need to buy extra hardware for each system; several OSs share one computer

Easy to recover: a broken VM can be deleted or restored

Limitations

Hardware: some hardware can't be emulated, or can't be reached directly by the guest

Complexity: setting up and maintaining the host and the VMs takes time, skill and money

Don't confuse a virtual machine with **virtual memory** (using disk space as extra RAM), which belongs to System software.

Logic notation and truth tables

At A Level, logic is written in Boolean algebra:

Gate	Boolean algebra	Output is 1 when
NOT	$\overline{A}$	$A = 0$
AND	$A.B$	both are 1
OR	$A + B$	at least one is 1
NAND	$\overline{A.B}$	not both 1
NOR	$\overline{A + B}$	both are 0
XOR	$A \oplus B$	the inputs are different

A gate can have **more than two inputs**. A three-input AND gives 1 only when all three inputs are 1; a three-input OR gives 1 when any of them is 1.

To complete a truth table for a circuit, label the output of every gate (the paper does this for you with letters such as P, Q, R, S), and fill in one **working column** per gate, in the order the signals flow, before the final output. With 3 inputs there are $2^3 = 8$ rows; with 4 inputs, 16.

A **sum-of-products** (SOP) expression is read straight from a truth table: for each row where the output is 1, write one AND term containing every input, with a bar over each input that is 0 in that row; then join the terms with $+$. For example the row $A = 0, B = 1, C = 1$ gives $\overline{A}.B.C$.

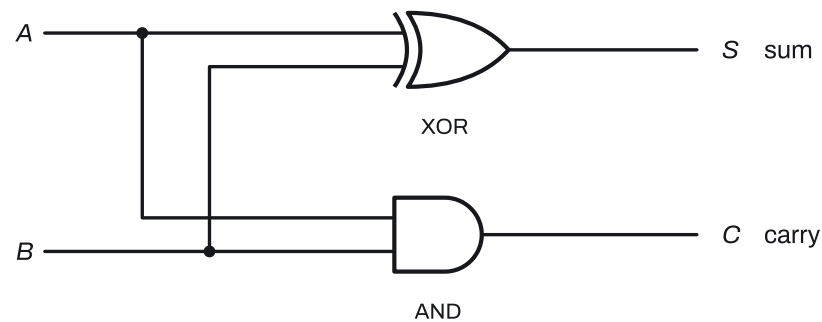
Half adders and full adders

A **half adder** adds two bits, A and B . It has two outputs: the **sum** bit S and the **carry** bit C . It is one XOR gate and one AND gate.

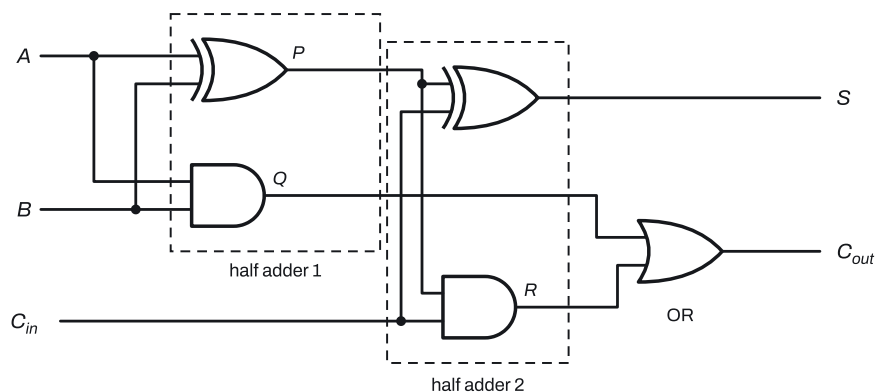
A	B	S (sum)	C (carry)
0	0	0	0
0	1	1	0

A	B	S (sum)	C (carry)
1	0	1	0
1	1	0	1

$S = A \oplus B = \overline{A}.B + A.\overline{B}$ and $C = A.B$. In binary $1 + 1 = 10$: sum 0, carry 1.



A half adder can only add the two bits in one column. To add longer binary numbers, each column must also add the **carry in** from the column to its right. A **full adder** adds three bits, A , B and the carry in C_{in} , and outputs a sum S and a carry out C_{out} . It is built from **two half adders and an OR gate**: the first half adder adds A and B , the second adds that sum to C_{in} , and the OR gate combines the two carries.



A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

The sum is 1 when an **odd** number of inputs are 1; the carry is 1 when **two or more** inputs are 1. As sums-of-products:

$$S = \overline{A}.\overline{B}.C_{in} + \overline{A}.B.\overline{C_{in}} + A.\overline{B}.\overline{C_{in}} + A.B.C_{in}$$

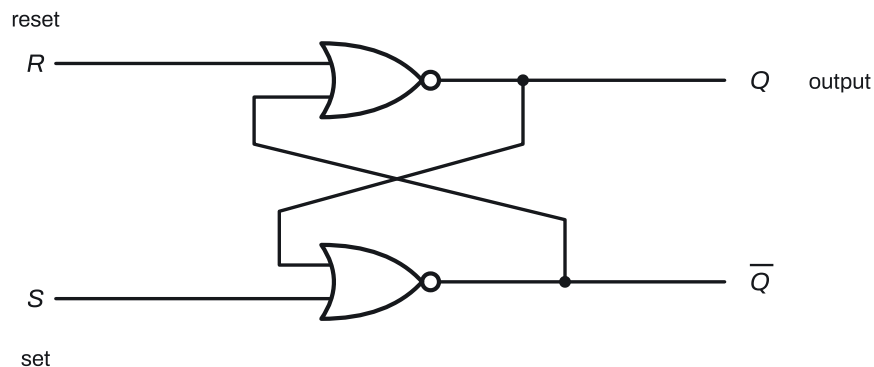
$$C_{out} = \overline{A}.B.C_{in} + A.\overline{B}.C_{in} + A.B.\overline{C_{in}} + A.B.C_{in}$$

Full adders chained together, the carry out of each feeding the carry in of the next, add whole binary numbers.

Flip-flops

Every circuit so far is **combinational**: the output depends only on the inputs now. A **flip-flop** is a **sequential** circuit: its outputs are fed back into its own gates, so its output also depends on what it held before. That lets it **store one bit**. Flip-flops are the storage elements in **registers** and in **SRAM**: a group of 8 stores a byte.

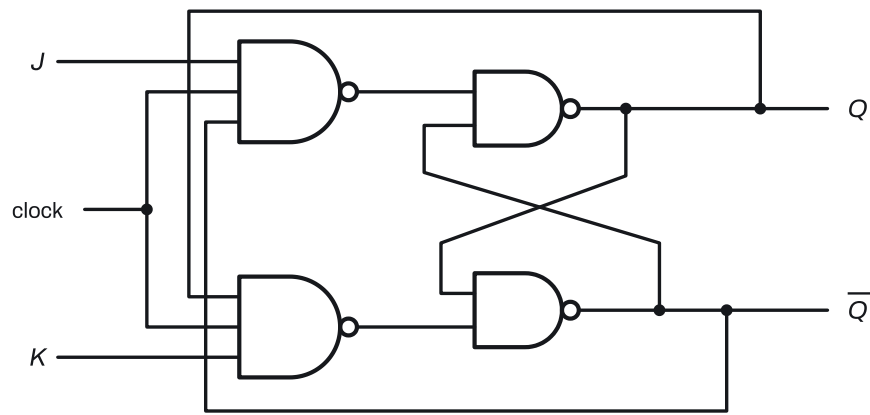
SR flip-flop. Two NOR gates, each gate's output feeding one input of the other. The inputs are S (set) and R (reset); the outputs are Q and $\overline{Q}$, which are normally opposite.



S	R	Q	$\overline{Q}$	What happens
0	0	no change	no change	the stored bit is kept
1	0	1	0	set: store 1
0	1	0	1	reset: store 0
1	1	0	0	invalid: Q and $\overline{Q}$ are no longer opposite

To see why $S = 0, R = 0$ keeps the bit: if $Q = 1$, the bottom gate gets $S = 0$ and $Q = 1$, so $\overline{Q} = 0$; the top gate gets $R = 0$ and $\overline{Q} = 0$, so Q stays 1. The loop holds itself. An SR flip-flop can also be built from two NAND gates; then the inputs work the other way round (a 0 on $\overline{S}$ sets it) and both inputs at 0 is the invalid state.

JK flip-flop. An SR flip-flop has two problems: $S = R = 1$ is invalid, and its outputs change the moment the inputs do. A **JK flip-flop** fixes both. It adds a **clock** input, so it only changes on a clock pulse and all the flip-flops in a register change together (they are **synchronised**). And it feeds Q and $\overline{Q}$ back to the input gates, so $J = K = 1$ becomes useful: it **toggles** the output.



J	K	Q after the clock pulse
0	0	no change
1	0	1 (set)
0	1	0 (reset)
1	1	toggles: the opposite of before

Boolean algebra

Boolean algebra lets you simplify an expression, and so build the same circuit with fewer gates. These are the laws you use. Each one has an AND form and an OR form.

Law	AND form	OR form
Identity	$A.1 = A$	$A + 0 = A$
Null (annulment)	$A.0 = 0$	$A + 1 = 1$
Idempotent	$A.A = A$	$A + A = A$
Inverse (complement)	$A.\bar{A} = 0$	$A + \bar{A} = 1$
Double negation	$\bar{\bar{A}} = A$	
Commutative	$A.B = B.A$	$A + B = B + A$
Associative	$(A.B).C = A.(B.C)$	$(A + B) + C = A + (B + C)$
Distributive	$A.(B + C) = A.B + A.C$	$A + B.C = (A + B).(A + C)$
Absorption	$A.(A + B) = A$	$A + A.B = A$
Redundancy	$A.(\bar{A} + B) = A.B$	$A + \bar{A}.B = A + B$
De Morgan's	$\overline{A.B} = \bar{A} + \bar{B}$	$\overline{A + B} = \bar{A}.\bar{B}$

Most simplifications follow one pattern: **take out a common factor** (distributive), spot $X + \bar{X} = 1$ (inverse), then tidy up with identity, absorption or redundancy. For example:

$$\bar{A}.B + A.B + A.\bar{B} = B.(\bar{A} + A) + A.\bar{B} = B.1 + A.\bar{B} = B + A.\bar{B} = B + A.$$

(distributive, inverse, identity, then redundancy with B in the place of A).

De Morgan's laws remove a bar that covers more than one letter: **break the bar and change the sign**. Split the long bar into a bar over each part, and swap every $\cdot$ under it for $+$, and every $+$ for $\cdot$. Two bars on the same thing cancel (double negation). For example:

$$\overline{\overline{A} \cdot B + C} = \overline{\overline{A} \cdot B} \cdot \overline{C} = (\overline{\overline{A}} + \overline{B}) \cdot \overline{C} = (A + \overline{B}) \cdot \overline{C}.$$

De Morgan's laws also show that NAND gates alone (or NOR gates alone) can build any circuit.

Karnaugh maps

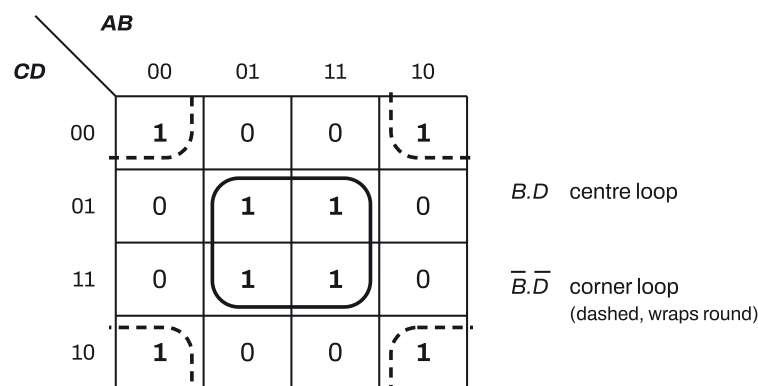
A **Karnaugh map** is a truth table drawn as a grid, arranged so that **neighbouring cells differ in exactly one input**. For that, the rows and columns go in **Gray code order**: 00, 01, 11, 10, not 00, 01, 10, 11. Grouping neighbouring 1s then removes the input that changes within the group, so you can read off the simplest sum-of-products by eye, which is quicker and safer than algebra when there are 3 or 4 inputs.

For 4 inputs the map has 16 cells, one per truth table row; for 3 inputs it has 8 (one input down the side, two across the top). Cambridge usually puts AB across the top and CD down the side; always follow the labels you are given.

Rules for the loops (groups):

- a loop contains **only 1s**, and every 1 must be in at least one loop;
- a loop is a **rectangle of 1, 2, 4, 8 or 16** cells (a power of 2);
- make each loop **as big as possible**, and use **as few loops as possible**;
- loops may **overlap**;
- the map **wraps around**: the top row is next to the bottom row and the left column is next to the right, so a loop can go off one edge and come back on the other, and the **four corners** form a group of four.

Reading a loop: look at each input across the cells of the loop. An input that **changes** within the loop is dropped. An input that stays **1** is written plain; one that stays **0** is written with a bar. Each loop gives one AND term; join the terms with $+$. A group of 8 in a 4-input map leaves 1 input, a group of 4 leaves 2, a pair leaves 3, and a single 1 keeps all 4.



In this map the four centre cells have $B = 1$ and $D = 1$ while A and C change, so they give $B.D$. The four corners have $B = 0$ and $D = 0$, so they give $\overline{B}.\overline{D}$. The whole map is $Z = \overline{B}.\overline{D} + B.D$, where the sum-of-products from the truth table would have 8 terms.

Key facts and formulas

There is no formula list in the 9618 exams, so every fact here is **Learn it**.

Formula	Status
RISC: few, simple, fixed-length instructions; one clock cycle each; few addressing modes; many general-purpose registers; hard-wired control unit; easy to pipeline; emphasis on software	Learn it
CISC: many complex, variable-length instructions; several clock cycles each; many addressing modes and formats; microprogrammed control unit; uses cache; emphasis on hardware	Learn it
Pipeline stages: IF, ID, OF, IE, WB; n instructions in a 5-stage pipeline take $n + 4$ cycles (not $5n$)	Learn it
SISD, SIMD, MISD, MIMD: single or multiple instruction streams, then single or multiple data streams	Learn it
Half adder: $S = A \oplus B$, $C = A.B$	Learn it
Full adder: $S = A \oplus B \oplus C_{in}$ (1 for an odd number of 1s); $C_{out} = 1$ when two or more inputs are 1	Learn it
SR: 10 sets, 01 resets, 00 holds, 11 invalid. JK: the same, but 11 toggles, and it is clocked	Learn it
De Morgan: $\overline{A.B} = \overline{A} + \overline{B}$ and $\overline{\overline{A} + \overline{B}} = \overline{A}.\overline{B}$	Learn it
$A + \overline{A} = 1$, $A.\overline{A} = 0$, $A + A.B = A$, $A + \overline{A}.B = A + B$	Learn it
K-map order 00, 01, 11, 10; loops of 1, 2, 4, 8; wrap around the edges	Learn it

How to do it

In the Paper 3 questions we have tagged for this topic (47 questions, 2021–2025, counting a question each time it was set, as most were set on two or three papers), the types came up this often. 26 of the 47 questions mix types, so the counts add up to more than 47.

Question type	Questions
Karnaugh maps: complete, loop and simplify	18

Question type	Questions
Sum-of-products from a truth table or circuit	15
Truth table from a logic circuit	9
RISC and CISC features and differences	9
Simplify with Boolean algebra and De Morgan's laws	6
Pipelining and interrupt handling	5
The four computer architectures	4
Virtual machines: roles, benefits and limitations	4
Half adders and full adders	4
Massively parallel computers	2
Flip-flops	1

1. Karnaugh maps: complete, loop and simplify (18 questions)

The question gives a truth table, a sum-of-products or a circuit, then asks you to fill in the map, draw the loops and write the result.

1. **Fill in the map.** Read the labels: which inputs are across the top and which down the side, in the order 00, 01, 11, 10. For each truth table row (or each term of the expression), find the cell where the top and side labels match and write 1. Then write **0 in every other cell**: a blank cell loses the mark.
2. **Loop the 1s**, biggest groups first: look for 8, then 4 (including the four corners and loops that wrap round an edge), then pairs, then any single 1 that has no neighbour. Check that every 1 is covered and that no loop is unnecessary.
3. **Write one term per loop**: keep the inputs that stay the same, with a bar if they stay 0. Join the terms with +. When the question says "do not carry out any further simplification", stop here.
4. **"Simplest form"**: if two terms share a letter, take it out as a common factor. For example, $A.C + B.C = C.(A + B)$.

Variations you will meet:

- **A 3-input map** (2 rows × 4 columns), filled from a sum-of-products expression. Each term fills one cell.
- **A single 1 with no neighbouring 1** stays a loop on its own and gives a term with all the inputs.
- **The map is given already filled**, and you only loop and write the expression.
- **Asked "the benefit of a K-map"**: it gives the simplest sum-of-products quickly and reliably, by eye, with fewer steps and fewer mistakes than Boolean algebra, and so a circuit with fewer gates.

2. Sum-of-products from a truth table or circuit (15 questions)

1. Find **every row where the output is 1**. Count them: the answer has exactly that many terms.

2. For each row write an AND term with **all** the inputs, in order, putting a bar over each input that is **0** in that row.
3. Join the terms with $+$. Do **not** simplify: the question wants the unsimplified form.

For a circuit, first complete its truth table (type 3), then read the SOP from the output column. For a half or full adder, write one SOP for each output.

3. Truth table from a logic circuit (9 questions)

1. Write the rows in binary order (000 to 111, or 0000 to 1111 for four inputs) if they aren't printed.
2. **Name every gate's output** (the paper usually labels them P, Q, R, S). Work out a column for each, following the signals from the inputs to the output. A wire that splits at a dot carries the same value to every branch; wires that cross without a dot are not joined.
3. Fill in the output column from the working columns. The working itself earns a mark, so fill in every working column.
4. Check gates with **three inputs**: a 3-input AND is 1 only if all three are 1, a 3-input OR is 1 if any is 1.

Useful shortcuts: a NOT column is the input column flipped; XOR is 1 where its two inputs differ; NAND and NOR are AND and OR flipped.

Variations: after the table, write the SOP for the output (type 2), put it in a K-map (type 1), or simplify the expression given for the circuit (type 5).

4. RISC and CISC features and differences (9 questions)

1. **"Identify features"** of one type: give the number asked, each a short separate fact from "RISC and CISC" above.
2. **"Differences"**: write each difference as a **pair**, one side for each type, for example "RISC has fewer, simpler instructions, but CISC has many complex instructions". Good pairs: instruction set size, instruction length (fixed or variable), clock cycles per instruction (one or several), addressing modes (few or many), registers (many or few), control unit (hard-wired or microprogrammed), pipelining (easy or hard), circuit complexity, memory access (load and store only, or many instructions).
3. **Tick-box tables** (does this apply to RISC or CISC?): "fewer general-purpose registers", "variable length" and "micro-coded control unit" are CISC; "smaller instruction set", "single-cycle" and "limited addressing modes" are RISC.

5. Simplify with Boolean algebra and De Morgan's laws (6 questions)

1. **Remove long bars first** with De Morgan's laws: break the bar, change the sign. Work from the outside in, one bar at a time, and cancel double bars.
2. **Multiply out brackets** (distributive law) if that gives terms you can combine.
3. **Combine terms**: take out a common factor to reach $X + \overline{X} = 1$, or use $A.A = A$, $A.\overline{A} = 0$, then identity, absorption ($A + A.B = A$) and redundancy ($A + \overline{A}.B = A + B$).

4. Write **one step per line**, naming the law used on each line, then write the final answer clearly.

Variations:

- **"Apply De Morgan's laws to"** a single expression: one step, for example $\overline{A + B + C} = \overline{A}.\overline{B}.\overline{C}$.
- **A long SOP** such as $A.\overline{B}.\overline{C}.D + A.\overline{B}.C.D + A.B.\overline{C}.D + A.B.C.D$: take out the common part ($A.D$), then pair terms to reach $\overline{C} + C = 1$ and $\overline{B} + B = 1$, leaving $A.D$.
- **After a K-map**, "use Boolean algebra to give your answer in its simplest form": one factorisation step.

6. Pipelining and interrupt handling (5 questions)

"Describe pipelining in RISC": instructions are split into **stages** (IF, ID, OF, IE, WB), each done in **one clock cycle**; while one instruction is being decoded the next is being fetched, so **several instructions are processed at the same time**, each at a different stage; no two are at the same stage in the same cycle; this raises **throughput**.

"Complete the table" for n instructions:

1. Put the first instruction in IF in cycle 1, then move it down one stage per cycle (a diagonal).
2. Start each later instruction one cycle after the one before, in the same diagonal pattern.
3. Check the last instruction finishes WB in cycle $n + 4$, and that no row has extra entries. Label the cells the way the question does (A, B, ... or 1.1, 1.2, ...).

Interrupts: give the four steps (detect at the end of a cycle; save the registers on the stack; run the ISR; restore the registers). For **"how does pipelining affect it"**: several instructions are in the pipeline at once; all are discarded except the one at write back; the interrupt is handled; then the processor restarts with the next instruction.

7. The four computer architectures (4 questions)

For each architecture asked: **write the name in full** (Single Instruction, Multiple Data), then **what it does**: how many processors, whether they carry out the same or different instructions, on the same or different data, and one extra point (sequential for SISD; array processor or pipelining for SIMD; each processor independent for MISD; asynchronous for MIMD). If told SISD is one of the four, "the other three" are SIMD, MISD and MIMD.

8. Virtual machines: roles, benefits and limitations (4 questions)

1. **"What is a virtual machine"**: software **emulation** of a computer system (hardware and/or software), running on a **host** computer, with a **guest** operating system.
2. **Benefits and limitations**: give each as a point **plus why**, as in the table in "Virtual machines". Two marks per benefit means "what" plus "because ...". Fit them to the scenario if there is one (emulating a new computer: test it without buying the hardware; but brand-new hardware may not be emulated yet).
3. **Host and guest roles**: host OS controls the real hardware and runs the VM software; guest OS runs inside the VM, controls the virtual hardware and runs under the host's control.

9. Half adders and full adders (4 questions)

1. Complete the truth table (type 3), with working columns if the circuit is given.
2. **Name it**: two inputs and two outputs is a **half adder**; three inputs (A , B and a carry in) is a **full adder**.
3. **Purpose of each output**: one is the **sum** bit (1 for an odd number of 1s), the other is the **carry** bit (1 when two or more inputs are 1).
4. If asked for the output expressions "as sum-of-products", give the unsimplified SOP (type 2), not $A \oplus B$.

10. Massively parallel computers (2 questions)

Give three points: a **large number of processors** (or separate computers) linked together; working **simultaneously** on **coordinated** parts of one task (collaborative processing); connected by a **network infrastructure** and **communicating by messages**.

11. Flip-flops (1 question)

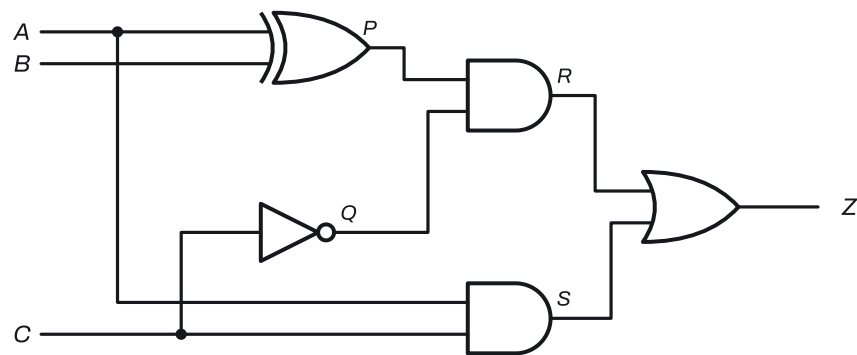
1. **Draw an SR flip-flop**: two NOR gates (or two NAND gates, not a mix), each with two inputs and one output; the output of each gate goes to an input of the other; label the free inputs S and R and the outputs Q and $\overline{Q}$. Check your drawing against the diagram in "Flip-flops": with NOR gates, R goes to the gate that gives Q .
2. **Purpose**: to store one binary digit (bit).
3. For a truth table, use the rows in "Flip-flops", and say that $S = R = 1$ is invalid (SR) or toggles (JK).

Worked examples

The bold codes show where marks are earned, as in Cambridge mark schemes. The number is how many marks.

- **B1** mark for a correct result on its own, with no method needed.
-

Example 1



(a) Complete the truth table for the logic circuit, showing the working columns P , Q , R and S . [3]

(b) Write the Boolean expression for Z as a sum-of-products. [2]

(c) Draw a Karnaugh map for Z , loop the groups and write the simplified sum-of-products. [4]

(a) $P = A \oplus B$, $Q = \overline{C}$, $R = P.Q$, $S = A.C$, $Z = R + S$.

A	B	C	P	Q	R	S	Z
0	0	0	0	1	0	0	0
0	0	1	0	0	0	0	0
0	1	0	1	1	1	0	1
0	1	1	1	0	0	0	0
1	0	0	1	1	1	0	1
1	0	1	1	0	0	1	1
1	1	0	0	1	0	0	0
1	1	1	0	0	0	1	1

B1 for all four working columns, **B1** for the first four rows of Z , **B1** for the last four.

(b) $Z = 1$ in rows 010, 100, 101 and 111:

$$Z = \overline{A}.B.\overline{C} + A.\overline{B}.\overline{C} + A.\overline{B}.C + A.B.C$$

B2 for all four terms (**B1** for two correct).

(c) With A down the side and BC across the top:

$A \backslash BC$	00	01	11	10
0	0	0	0	1
1	1	1	1	0

B1 for the map. Loops: the pair 100, 101 ($A = 1, B = 0, C$ changes) gives $A.\overline{B}$; the pair 101, 111 ($A = 1, C = 1, B$ changes) gives $A.C$; the 1 at 010 has no neighbouring 1 (the cells beside it and below it are 0), so it is a loop on its own, $\overline{A}.B.\overline{C}$. **B1** for the two pairs, **B1** for the single.

$$Z = \overline{A}.B.\overline{C} + A.\overline{B} + A.C$$

B1. The three 1s in the bottom row can't be one loop, because a loop of 3 is not allowed.

Example 2

A truth table has output $Z = 1$ for exactly these inputs $ABCD$: 0000, 0010, 1000, 1010, 1100 and 1110; $Z = 0$ for the other ten.

(a) Write Z as a sum-of-products. [2]

(b) Complete a K-map for Z and draw loops to give an optimal sum-of-products. [4]

(c) Write the simplified sum-of-products, then use Boolean algebra to give it in its simplest form. [3]

(a)

$$Z = \overline{A}.\overline{B}.\overline{C}.\overline{D} + \overline{A}.\overline{B}.C.\overline{D} + A.\overline{B}.\overline{C}.\overline{D} + A.\overline{B}.C.\overline{D} + A.B.\overline{C}.\overline{D} + A.B.C.\overline{D}$$

B2 for all six terms (**B1** for three or more).

(b) AB across the top, CD down the side:

$CD \backslash AB$	00	01	11	10
00	1	0	1	1
01	0	0	0	0
11	0	0	0	0
10	1	0	1	1

B2 for the map with no errors. Loops (**B1** each):

- the **four corners** ($AB = 00$ or $10, CD = 00$ or 10): $B = 0$ and $D = 0$ throughout, A and C change, giving $\overline{B}.\overline{D}$;
- the columns 11 and 10 in rows 00 and 10, a square that **wraps from the top row to the bottom row**: $A = 1$ and $D = 0$ throughout, giving $A.\overline{D}$. It overlaps the corners loop, which is allowed.

(c)

$$Z = \overline{B}.\overline{D} + A.\overline{D}$$

B1 for one correct term, **B1** for both and nothing else. Both terms contain $\overline{D}$, so take it out (distributive law):

$$Z = \overline{D}.(A + \overline{B})$$

B1. Six four-input AND gates and an OR gate have become one NOT, one OR and one AND (plus a NOT for $\overline{B}$).

Example 3

Simplify $X = \overline{\overline{A} \cdot \overline{B}} \cdot (A + C)$ using De Morgan's laws and Boolean algebra. Show all the stages. **[4]**

$$X = (\overline{\overline{A} + \overline{B}}) \cdot (A + C) \quad \text{De Morgan's law}$$

B1

$$X = (A + \overline{B}) \cdot (A + C) \quad \text{double negation}$$

$$X = A \cdot A + A \cdot C + \overline{B} \cdot A + \overline{B} \cdot C \quad \text{distributive law}$$

B1 for a correct second law.

$$X = A + A \cdot C + A \cdot \overline{B} + \overline{B} \cdot C \quad \text{idempotent law}$$

$$X = A + \overline{B} \cdot C \quad \text{absorption law (twice)}$$

B1 for a third correct law, **B1** for the answer. (The OR form of the distributive law, $A + B \cdot C = (A + B) \cdot (A + C)$, gives the same answer in one step.)

Example 4

A RISC processor uses a five-stage pipeline: IF, ID, OF, IE and WB.

(a) Show how four instructions, W, X, Y and Z, pass through the pipeline, and state how many clock cycles they take. **[3]**

(b) State how many clock cycles the same four instructions would take without pipelining. **[1]**

(c) Explain how an interrupt is handled by this processor while the pipeline is full. **[3]**

(d) State two features of RISC processors that make pipelining straightforward. **[2]**

(a)

Stage	1	2	3	4	5	6	7	8
IF	W	X	Y	Z				
ID		W	X	Y	Z			
OF			W	X	Y	Z		
IE				W	X	Y	Z	
WB					W	X	Y	Z

B1 for W starting in IF in cycle 1 and finishing WB in cycle 5, **B1** for X, Y and Z each one cycle later with nothing extra. They take $4 + 4 = 8$ clock cycles. **B1**

(b) $4 \times 5 = 20$ clock cycles. **B1**

(c) Any three of: several instructions are part-way through the pipeline when the interrupt is detected (**B1**); all of them are discarded except the one at the write-back stage, which is completed (**B1**); the registers are saved on the stack and the interrupt service routine is run for that instruction (**B1**); afterwards the registers are restored and the processor restarts from the next instruction, fetching the discarded ones again (**B1**).

(d) Any two of: instructions have a **fixed length** (**B1**); each takes **one clock cycle** (**B1**); there are **few instruction formats and addressing modes**, so decoding is simple (**B1**).

Common mistakes

- **Leaving cells empty in a K-map.** Examiners only give the mark when **every** cell holds a 0 or a 1: no blanks, dashes or other symbols.
- **Putting the columns in binary order.** A K-map goes 00, 01, 11, 10. With 10 before 11, neighbouring cells differ in two inputs and the loops give wrong terms.
- **Loops that aren't allowed:** groups of 3 or 6, L-shapes, diagonal cells, or a loop around a 0. Missing a **wrap-around** group (corners, or top and bottom rows) gives more terms than needed, so it isn't optimal.
- **Simplifying when asked for a sum-of-products.** Examiners report many answers giving a simplified expression when the question wanted the full SOP (for example for the outputs of a full adder), and the reverse: giving the SOP when asked for the simplest form. Read which one the question wants.
- **Leaving an input out of an SOP term.** Every term needs every input, with a bar over each 0.
- **Getting De Morgan's laws half right.** A common error the reports mention: splitting the bar into separate bars but **keeping the sign**, so $\overline{A \cdot B}$ becomes $\overline{A} \cdot \overline{B}$. It is $\overline{A} + \overline{B}$: break the bar **and** change the sign.
- **Not showing the working** in Boolean algebra. A bare final answer loses the method marks; write each step and the law.
- **Skipping the working columns** in a circuit's truth table, or treating a crossing wire as a join. Only a dot joins wires.
- **Virtual machines vs virtual memory.** Candidates asked about virtual memory have written about virtual machines; don't mix the two up.
- **Vague RISC/CISC answers.** "RISC is faster" or "CISC is complex" earns nothing. Give the specific feature: fixed-length instructions, single clock cycle, hard-wired control unit. Many answers mixed up which features belong to which, so learn the two lists.
- **Pipelining answers that miss the main idea:** the mark scheme wants "several instructions are processed **at the same time**, each at a different stage". In the table, don't add extra stages or start two instructions in the same cycle.

How to get the marks

- **K-maps are marked in pieces:** the filled map (2 marks, 1 if one error), one mark per correct loop, then one mark for a correct term and one for the whole expression with **no extra terms**. A mistake in the map can cost several later marks, so check every 1 against the truth table before looping.
- **"Optimal"** means the fewest, largest loops. An extra or unnecessary loop loses the mark for the expression.
- **"Do not carry out any further simplification"** means stop at the SOP from your loops. **"Simplest form"** asks for one more step, usually taking out a common factor.
- **SOP from a truth table** is marked by how many terms are correct, so write every term, in the input order A, B, C, D , clearly separated by $+$.
- **"Show your working"** in a circuit's truth table: the working columns earn a mark of their own.
- **"Simplify ... using Boolean algebra"**: marks go for each correct law applied and for the final answer. Name the law on each line.
- **"Identify"** or **"State"**: short points, one fact each. **"Describe"**: say what it is and how it works. **"Explain"**: give the reason, with "because" or "so". **"Outline"**: brief points in order, such as the steps of interrupt handling.
- **"Two benefits and two limitations"**: each point must be different. Two versions of "it's slower" count once.
- **Architectures**: write the name in full, then what the processors and data do. The name alone earns only one mark.

Quick check

1. Apply De Morgan's laws to $\overline{A \cdot B \cdot C}$.
2. An SR flip-flop made of NOR gates starts with $Q = 0$. The inputs are then, in turn, $S = 1, R = 0$; then $S = 0, R = 0$; then $S = 0, R = 1$. Give Q after each step.
3. A program of 9 instructions runs on a five-stage pipeline. How many clock cycles does it take, and how many would it take without pipelining?
4. Use a K-map to simplify $Z = \overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot C + A \cdot B \cdot C + A \cdot B \cdot \overline{C}$.
5. (a) A graphics chip applies the same brightness change to every pixel of an image at once. Name its architecture. (b) Give one benefit and one limitation of running a company's old accounts program on a virtual machine.

Answers

1. Break the bar and change both signs: $\overline{A} + \overline{\overline{B}} + \overline{C} = \overline{A} + B + \overline{C}$ (double negation on B).
2. $S = 1, R = 0$ sets it: $Q = 1$. $S = 0, R = 0$ holds it: $Q = 1$. $S = 0, R = 1$ resets it: $Q = 0$.
3. $9 + 4 = 13$ clock cycles with pipelining; $9 \times 5 = 45$ without.
4. With A down the side and BC across (00, 01, 11, 10): row 0 is 0, 1, 1, 0 and row 1 is 0, 1, 1, 1. The four cells with $C = 1$ (columns 01 and 11) form a group of four: C . The pair 111, 110 gives $A.B$. So $Z = C + A.B$.
5. (a) SIMD (Single Instruction, Multiple Data): one instruction carried out on many pieces of data at the same time. (b) Benefit: the old program can still run even though it is not compatible with the new computer's operating system, using a suitable guest operating system. Limitation: it runs more slowly than on its own hardware, because the virtual machine software uses extra processing time and memory.

Past questions to try

- [9618/32/M/J/23 Q8](#): describing pipelining and completing a pipeline table.
- [9618/32/O/N/22 Q8](#): drawing an SR flip-flop, its purpose, then a De Morgan simplification.
- [9618/33/M/J/21 Q7](#): the truth table of a full adder, its name and the purpose of each output.
- [9618/33/O/N/23 Q5](#): the features of the SISD and MIMD architectures.