Sample Mid-semester Exam #6

Table of contents

How to use this . . . 2
Question index . . . 2
Q1 — Two's complement of a negative . . . 4
Q2 — Octal to hexadecimal width . . . 5
Q3 — Excess-32 range . . . 6
Q4 — Ones' to two's complement . . . 6
Q5 — Reading a two-gate circuit . . . 8
Q6 — Simplifying a product of sums . . . 9
Q7 — Circuit for an XOR expression . . . 10
Q8 — Truth table for an expression . . . 12
Q9 — Timing diagram to expression . . . 14
Q10 — Simplifying a complemented product . . . 16
Q11 — 3-input NOR from 2-input NORs . . . 17
Q12 — D flip-flop input from its output . . . 19
Q13 — Two's complement addition . . . 21
Q14 — Detecting overflow . . . 22
Q15 — Overflow on an addition . . . 23
Q16 — Multiplexer as function generator . . . 23
Q17 — Reading an 8-input multiplexer . . . 25
Q18 — Load/shift register trace . . . 27
Q19 — Synchronous counter sequence . . . 29
Q20 — Flip-flops for an output sequence . . . 31
Q21 — State diagram to state table . . . 32
Q22 — Flip-flops for a state machine . . . 34
Q23 — Tracing a state machine . . . 35
Q24 — ALU control settings . . . 36
Q25 — ALU control settings . . . 38

This is UQ's sixth sample mid-semester paper for csse2010[1] — CSSE2010/CSSE7201 Introduc-

[1] courses/csse2010/csse2010.pdf

tion to Computer Systems — printed for the Semester One mid-semester examinations of 2020, St Lucia campus. It runs for **60 minutes** with **10 minutes** of reading time, is an **open book** examination (calculators limited to the Casio FX82 series or a UQ-approved labelled model, no electronic aids), and is answered entirely on a supplied True/False and Multiple Choice answer sheet. There are **25 questions**, each worth **one mark**; incorrect, missing or multiple answers score zero, so there is no penalty for guessing.

The paper covers the first five weeks of the course: number representations and binary arithmetic, logic gates and Boolean simplification, combinational blocks, flip-flops and counters, state machines, and the ALU bit slice.

> ⚠ Answers are derived, not official
>
> The source paper ships without an answer key, so every answer below was worked out during transcription rather than taken from UQ. Each one shows its reasoning so it can be checked rather than trusted; anything uncertain is flagged. Treat a disagreement between your working and the answer here as a reason to check both.

How to use this

Two ways. **Timed run** — do all 25 questions in 60 minutes without looking at anything; on the website the answers are collapsed, so they stay hidden until you click them. **Targeted practice** — use the index table below to pick out only the questions for one week or one concept, and read the working straight away.

Note that in the PDF build every answer is expanded and visible, so the site version is the one to practise from.

Question index

Q	Topic	Week	Concept notes
1	Two's complement of a negative	1	binary-number-representations[2] · week1-lab-number-representations[3]
2	Octal to hexadecimal width	1	binary-number-representations[4] · week1-lab-number-representations[5]

[2] courses/csse2010/binary-number-representations.pdf
[3] courses/csse2010/week1-lab-number-representations.pdf
[4] courses/csse2010/binary-number-representations.pdf
[5] courses/csse2010/week1-lab-number-representations.pdf

Q	Topic	Week	Concept notes
3	Excess-32 range	1	binary-number-representations[6] · week1-lab-number-representations[7]
4	Ones' to two's complement	1	binary-number-representations[8] · week1-lab-number-representations[9]
5	Reading a two-gate circuit	1	logic-gates[10]
6	Simplifying a product of sums	2	boolean-algebra[11]
7	Circuit for an XOR expression	1	logic-gates[12]
8	Truth table for an expression	2	boolean-algebra[13]
9	Timing diagram to expression	2	timing-diagrams[14]
10	Simplifying a complemented product	2	boolean-algebra[15]
11	3-input NOR from 2-input NORs	2	boolean-algebra[16]
12	D flip-flop input from its output	3	flip-flops-and-latches[17]
13	Two's complement addition	2	binary-number-representations[18]
14	Detecting overflow	2	binary-number-representations[19]
15	Overflow on an addition	2	binary-number-representations[20]

[6] courses/csse2010/binary-number-representations.pdf
[7] courses/csse2010/week1-lab-number-representations.pdf
[8] courses/csse2010/binary-number-representations.pdf
[9] courses/csse2010/week1-lab-number-representations.pdf
[10] courses/csse2010/logic-gates.pdf
[11] courses/csse2010/boolean-algebra.pdf
[12] courses/csse2010/logic-gates.pdf
[13] courses/csse2010/boolean-algebra.pdf
[14] courses/csse2010/timing-diagrams.pdf
[15] courses/csse2010/boolean-algebra.pdf
[16] courses/csse2010/boolean-algebra.pdf
[17] courses/csse2010/flip-flops-and-latches.pdf
[18] courses/csse2010/binary-number-representations.pdf
[19] courses/csse2010/binary-number-representations.pdf
[20] courses/csse2010/binary-number-representations.pdf

Q	Topic	Week	Concept notes
16	Multiplexer as function generator	2	combinational-logic-blocks[21]
17	Reading an 8-input multiplexer	2	combinational-logic-blocks[22]
18	Load/shift register trace	3	shift-registers[23]
19	Synchronous counter sequence	4	counters[24]
20	Flip-flops for an output sequence	4	state-machines[25]
21	State diagram to state table	4	state-machines[26]
22	Flip-flops for a state machine	4	state-machines[27]
23	Tracing a state machine	4	state-machines[28]
24	ALU control settings	5	arithmetic-logic-unit[29]
25	ALU control settings	5	arithmetic-logic-unit[30]

Q1 — Two's complement of a negative

What is -73_{10} expressed in 8-bit two's complement format?

(a) 01001000

(b) 01001001

(c) 10110110

(d) 10110111

[21] courses/csse2010/combinational-logic-blocks.pdf
[22] courses/csse2010/combinational-logic-blocks.pdf
[23] courses/csse2010/shift-registers.pdf
[24] courses/csse2010/counters.pdf
[25] courses/csse2010/state-machines.pdf
[26] courses/csse2010/state-machines.pdf
[27] courses/csse2010/state-machines.pdf
[28] courses/csse2010/state-machines.pdf
[29] courses/csse2010/arithmetic-logic-unit.pdf
[30] courses/csse2010/arithmetic-logic-unit.pdf

(e) 11001001

Answer

(d)
Start from the magnitude. $73 = 64 + 8 + 1$, so

$$73_{10} = 0100\,1001_2$$

which is option (b) — the positive value, offered as a distractor. Negate it the usual way (see binary-number-representations[31]): invert every bit, then add one.

$$\overline{0100\,1001} = 1011\,0110 \qquad 1011\,0110 + 1 = 1011\,0111$$

So $-73_{10} = 1011\,0111_2$. Option (c) is the one's complement — the inversion without the $+1$ — which is the usual slip here.
Checking: 1011 0111 as two's complement is $-128+32+16+4+2+1 = -128+55 = -73$.

Q2 — Octal to hexadecimal width

What is the minimum number of hexadecimal digits needed to represent the octal number 120105?

(a) 4

(b) 5

(c) 6

(d) 18

(e) None of the above

Answer

(a)
Go through binary, since one octal digit is exactly three bits and one hex digit is exactly four:

$$120105_8 = \underbrace{001}_{1}\ \underbrace{010}_{2}\ \underbrace{000}_{0}\ \underbrace{001}_{1}\ \underbrace{000}_{0}\ \underbrace{101}_{5}$$

That is 18 bits, but the leading two are zeros, so the value is really 16 significant bits.

[31] courses/csse2010/binary-number-representations.pdf

Regroup from the **right** in fours:

$$1010\ 0000\ 0100\ 0101_2 = \mathrm{A045}_{16}$$

Four hex digits. Option (d), 18, is the bit count rather than the digit count; option (c), 6, is the octal digit count.
Checking in decimal: $120105_8 = 8^5 + 2 \cdot 8^4 + 8^2 + 5 = 32768 + 8192 + 64 + 5 = 41029$, and $\mathrm{A045}_{16} = 10 \cdot 4096 + 4 \cdot 16 + 5 = 41029$.

Q3 — Excess-32 range

A 6-bit number in excess-32 format can represent integers from

(a) –32 to 31

(b) –31 to 32

(c) –32 to 32

(d) 0 to 63

(e) None of the above

Answer

(a)
In excess-K (biased) notation the stored bit pattern is read as an ordinary unsigned number and then K is subtracted to get the value it represents:

$$\text{value} = \text{unsigned}(b) - K$$

Six bits give unsigned values 0 to $2^6 - 1 = 63$, and here $K = 32$, so the representable values run from $0 - 32 = -32$ up to $63 - 32 = 31$.
Option (d) is the unsigned range before the bias is applied; option (c) has 65 values, which six bits cannot hold.

Q4 — Ones' to two's complement

How can any n-bit ones' complement number be converted to an (n+1)-bit two's complement representation?

(a) By prepending 0 and adding 1

(b) By prepending the sign-bit, flipping all the bits and then adding 1

(c) By prepending the sign-bit

(d) By prepending the sign-bit and adding 1

(e) By prepending the sign-bit and adding the value of the sign-bit

Answer

(e)

The word **any** is what the question turns on: the recipe has to work for positive and negative values alike, and the four wrong options each work for only one sign.

Write the n-bit ones' complement number as sign bit s over the remaining bits, whose unsigned value is M. Its value is

$$V = -s\,(2^{n-1} - 1) + M$$

because a negative ones' complement number is the bitwise inverse of its magnitude. Now prepend a copy of the sign bit, giving an $(n+1)$-bit pattern $s\,s\,b_{n-2}\ldots b_0$, and read it as two's complement:

$$-s\,2^n + s\,2^{n-1} + M = -s\,2^{n-1} + M$$

That is short of V by exactly s. Adding s — "the value of the sign-bit" — closes the gap:

$$-s\,2^{n-1} + M + s = -s\,(2^{n-1} - 1) + M = V$$

Concretely, with $n = 4$:

Ones' complement	Value	Prepend sign	Add s	Two's complement value
0011	$+3$	00011	$+0 = 00011$	$+3$
1010	-5	11010	$+1 = 11011$	-5
1000	-7	11000	$+1 = 11001$	-7
1111	-0	11111	$+1 = 00000$	0

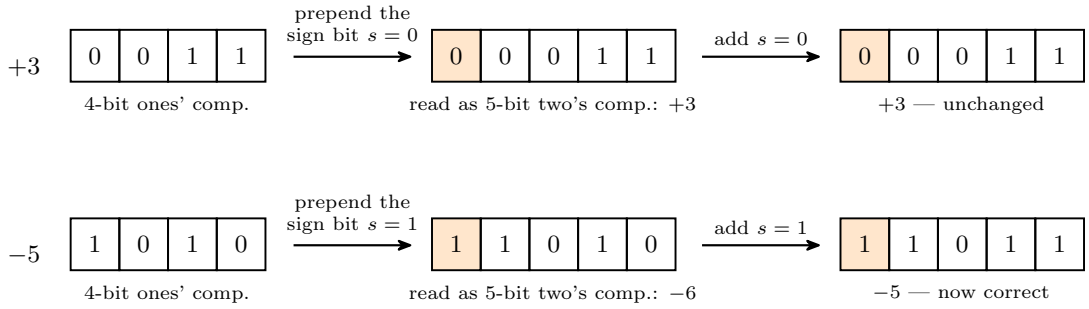


Figure 1: The first two rows of that table drawn out. Prepending the sign bit is the same move in both cases, but it lands in the right place only for the positive value: reading the widened negative pattern as two's complement gives −6, one too low, because ones' complement's top bit is worth $-(2^{n-1}-1)$ while two's complement's is worth -2^{n-1}. The correction is exactly that difference of 1, and it is needed precisely when the sign bit is 1 — which is what makes 'add the value of the sign-bit' the only option that works for both signs. The prepended bit is shaded.

Option (d) adds 1 unconditionally, which is right for negatives and one too many for positives (it would turn +3 into +4); option (c) never adds, which is right for positives and one too few for negatives.
See week1-lab-number-representations[32] for the two formats' bit-position weights side by side, which is where the difference of 1 comes from.

Q5 — Reading a two-gate circuit

What function is implemented by the following circuit?

(a) $\overline{A.B.C}$

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

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

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

[32] courses/csse2010/week1-lab-number-representations.pdf

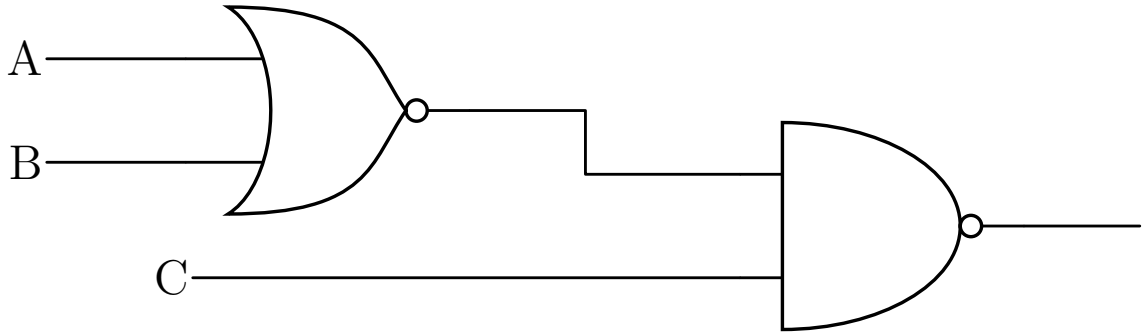


Figure 2: Question 5's circuit: a two-input NOR on A and B, whose output is one input of a two-input NAND with C.

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

Answer

(b)
Read the circuit gate by gate. The first gate is an OR body with an output bubble — a **NOR** — so its output is

$$\overline{A + B}$$

That feeds one input of an AND body with an output bubble — a **NAND** — whose other input is C. A NAND of P and C is $\overline{P \cdot C}$, so the whole circuit is

$$\overline{\overline{A + B} \cdot C}$$

which is option (b) exactly as printed.
The distractors are all *plausible* readings of a two-gate circuit: option (d) is what you get if the first gate is misread as a NAND ($\overline{A \cdot B} = \overline{A} + \overline{B}$ into a NAND with C), and option (e) swaps the two gate types over entirely. Options (a) and (c) do not have the right shape at all — (a) is a single three-input NAND, (c) has the outer bar covering an OR rather than the AND with C.

Q6 — Simplifying a product of sums

Which of the following logic functions is identical to $\overline{(A + B).(\overline{A} + B + C)}$?

(a) $\overline{B}$

(b) $\overline{A}.\overline{B}.\overline{C}$

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

(d) $\overline{A} + \overline{B} + \overline{C}$

(e) None of the above

Answer

(c)
Simplify what is inside the bar first, by multiplying the two sums out:

$$\begin{aligned}(A+B)(\overline{A}+B+C) &= A\overline{A} + AB + AC + \overline{A}B + BB + BC \\ &= 0 + AB + AC + \overline{A}B + B + BC \\ &= B\,(A + \overline{A} + 1 + C) + AC \\ &= B + AC\end{aligned}$$

Everything with a B in it collapses into the bare B term, leaving AC beside it. Now complement, using De Morgan twice (see boolean-algebra[33]):

$$\begin{aligned}\overline{B + AC} &= \overline{B} \cdot \overline{AC} \\ &= \overline{B}\,(\overline{A} + \overline{C}) \\ &= \overline{A}\,\overline{B} + \overline{B}\,\overline{C}\end{aligned}$$

which is option (c).
Spot-check a row: $A = 1, B = 0, C = 0$ gives $(1+0)(0+0+0) = 1 \cdot 0 = 0$, so the complement is 1; and $\overline{A}\overline{B} + \overline{B}\overline{C} = 0 \cdot 1 + 1 \cdot 1 = 1$.

Q7 — Circuit for an XOR expression

Which of the following logic circuits is an implementation of the function $(\overline{A.B}) \oplus (\overline{C + D})$?

Answer

(e)
The trap here is that the target expression complements both halves before the XOR, and **XOR is blind to complementing both of its inputs**:

$$\overline{P} \oplus \overline{Q} = \overline{P}\,Q + P\,\overline{Q} \ldots$$

more directly, $\overline{P} \oplus \overline{Q} = (\overline{P} \oplus Q) \oplus 1 \oplus 1 \cdots$ — easiest just to check the four rows:

[33] courses/csse2010/boolean-algebra.pdf

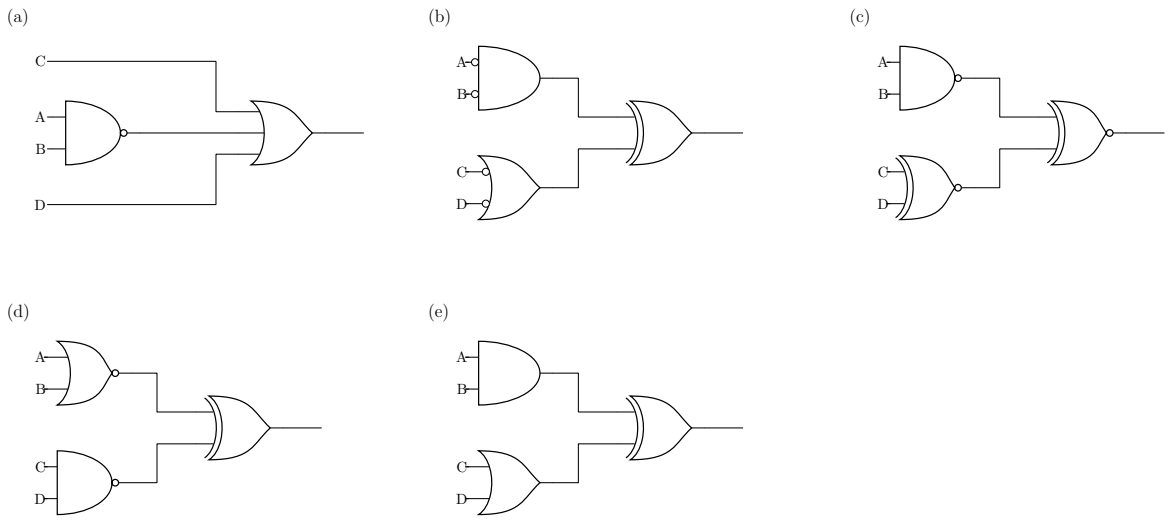


Figure 3: Question 7's five candidate circuits, laid out as one figure: (a), (b) and (c) on the top row, (d) and (e) on the bottom. Small circles on a gate's inputs in (b) are inversion bubbles; a circle on the output is the usual NAND/NOR/XNOR bubble.

P	Q	$P \oplus Q$	$\overline{P} \oplus \overline{Q}$
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

They are the same column. So with $P = A.B$ and $Q = C + D$,

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

and the circuit that draws that literally is **(e)**: a plain AND on A, B, a plain OR on C, D, and an XOR joining them.
Checking the others, in the same $P \oplus Q$ terms:

- **(a)** has no XOR at all — it is $C + \overline{A.B} + D$.
- **(b)** is $(\overline{A}.\overline{B}) \oplus (\overline{C} + \overline{D}) = \overline{A + B} \oplus \overline{C}.\overline{D} = (A + B) \oplus (C.D)$ — the AND and OR the wrong way round.
- **(c)** is $\overline{\overline{A.B} \oplus \overline{C \oplus D}}$ — an XNOR of an XNOR, nothing like it.
- **(d)** is $\overline{A + B} \oplus \overline{C}.\overline{D}$, the same function as (b), which is another reason neither can be the answer.

Q8 — Truth table for an expression

Which of the following truth tables is equivalent to the function

$$X = \left(\overline{\overline{A}.\overline{C}}\right).\left(\overline{\overline{A}.B} + \overline{C}.B\right)$$

(a)

A	B	C	X
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

(b)

A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

(c)

A	B	C	X
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1

A	B	C	X
1	1	0	0
1	1	1	1

(d)

A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

(e) None of the above

Answer

(a)
Simplify before tabulating — it is much less error-prone than eight substitutions. De Morgan turns both complemented products into sums:

$$\overline{\overline{A}\,\overline{C}} = A + C \qquad\qquad \overline{\overline{A}\,B} = A + \overline{B}$$

so

$$X = (A + C)\,(A + \overline{B} + \overline{C}\,B)$$

The second bracket absorbs: $\overline{B} + B\,\overline{C} = \overline{B} + \overline{C}$ (the classic $P + \overline{P}Q = P + Q$ with $P = \overline{B}$, $Q = \overline{C}$). That leaves

$$X = (A + C)\,(A + \overline{B} + \overline{C})$$

Multiply out. Every term containing A collapses into a single A, and $C \cdot \overline{C} = 0$:

$$X = A + C\,\overline{B} + C\,\overline{C} = A + \overline{B}\,C$$

Tabulating $A + \overline{B}C$:

A	B	C	$\overline{B}C$	X
0	0	0	0	**0**
0	0	1	1	**1**
0	1	0	0	**0**
0	1	1	0	**0**
1	0	0	0	**1**
1	0	1	1	**1**
1	1	0	0	**1**
1	1	1	0	**1**

That is 0, 1, 0, 0, 1, 1, 1, 1 — option (a). Option (d) is just $X = A$, i.e. what you get if you drop the $\overline{B}C$ term.

Q9 — Timing diagram to expression

Which of the following expressions is NOT equivalent to the following timing diagram?

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

(b) $(A \oplus C) + \overline{A}.B.\overline{C}$

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

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

(e) All of the above are equivalent to the timing diagram

Answer

(e)
First read the diagram off as a truth table. A toggles every interval, B every second interval and C every fourth, so the eight intervals are the eight combinations of ABC — but note they arrive in the order CBA counting up, not ABC. Reading the level of each track in each interval:

Interval	A	B	C	X
1	0	0	0	0
2	1	0	0	1
3	0	1	0	1
4	1	1	0	1
5	0	0	1	1

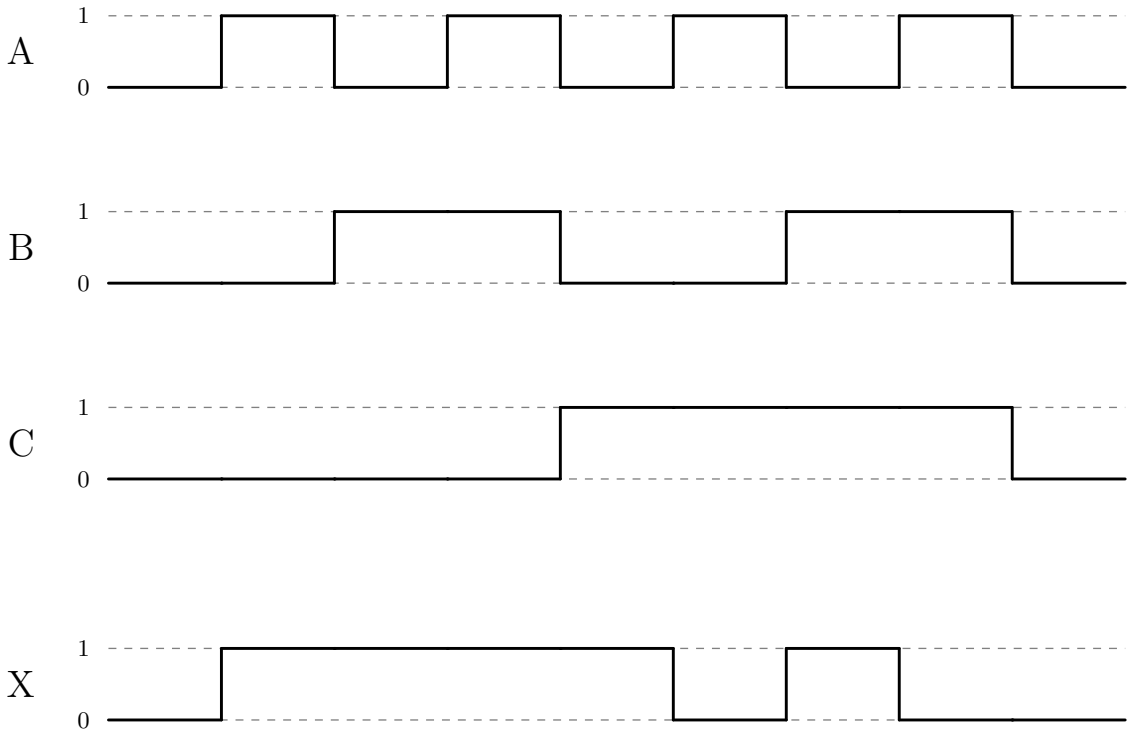


Figure 4: Question 9's timing diagram. A toggles every interval, B every two and C every four, so the eight intervals cover all eight combinations of A, B and C once each; X is the function being described. All four signals return to 0 in the tail at the right.

6	1	0	1	0
7	0	1	1	1
8	1	1	1	0

Sorted into ABC order, X is 1 for $ABC \in \{001, 010, 011, 100, 110\}$ and 0 for $\{000, 101, 111\}$. Grouping the minterms:

$$X = \overline{A}\,\overline{B}\,C + \overline{A}B\overline{C} + \overline{A}BC + A\overline{B}\,\overline{C} + AB\overline{C} = \overline{A}B + A\overline{C} + \overline{A}\,\overline{B}C = \overline{A}B + \overline{A}C + A\overline{C}$$

Now evaluate all four options on those eight rows. Writing the X column as the string $X_{000}X_{001}\ldots X_{111}$, the diagram gives **01111010**, and:

Option	X column	Matches?
(a) $A\overline{C} + B\overline{C} + \overline{A}B + \overline{A}C$	01111010	yes
(b) $(A \oplus C) + \overline{A}B\overline{C}$	01111010	yes
(c) $A\overline{B}\,\overline{C} + B\overline{C} + \overline{A}C$	01111010	yes
(d) $A\overline{B}\,\overline{C} + AB\overline{C} + \overline{A}B + \overline{A}\,\overline{B}C$	01111010	yes

All four are the same function, written four ways: (d) is the raw minterm list with two terms merged, (a) adds the redundant $B\overline{C}$ cover, (c) splits $A\overline{C}$ against B, and (b) notices that everything except $\overline{A}B\overline{C}$ is exactly $A \neq C$. So nothing on the list fails, and the answer is (e).
Worth noticing as a sanity check: since (a)–(d) are all *identical to each other*, the answer could only ever be "all of them" or "none of them", and there is no "none" option — so the diagram had to match.

Q10 — Simplifying a complemented product

Which of the following logic functions is identical to $\overline{(A + \overline{C}).(\overline{A} + B.\overline{C})}$?

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

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

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

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

(e) None of the above

Answer

(a)
Multiply out the inside first:

$$\begin{aligned}(A+\overline{C})(\overline{A}+B\overline{C}) &= A\overline{A}+AB\overline{C}+\overline{A}\,\overline{C}+B\overline{C}\,\overline{C}\\ &= 0+AB\overline{C}+\overline{A}\,\overline{C}+B\overline{C}\\ &= \overline{A}\,\overline{C}+B\overline{C}\end{aligned}$$

$AB\overline{C}$ is absorbed by $B\overline{C}$ (it is a subset of it), and $\overline{C}\,\overline{C}=\overline{C}$. Factor:

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

Now complement with De Morgan (see boolean-algebra[34]):

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

which is option (a).
Spot-check: $A=1,B=0,C=0$ gives $(1+1)(0+0)=0$ inside, so the answer should be 1; and $A\overline{B}+C=1\cdot 1+0=1$. And $A=0,B=1,C=0$ gives $(0+1)(1+1)=1$ inside, so 0 out; $A\overline{B}+C=0+0=0$.

Q11 — 3-input NOR from 2-input NORs

Which of the following circuits implements a 3 input NOR gate ($\overline{A+B+C}$) using only 2-input NOR gates?

Answer

(b)
Writing $\mathrm{NOR}(P,Q)=\overline{P+Q}$, note that a NOR with both inputs tied together is an inverter: $\mathrm{NOR}(P,P)=\overline{P}$. That is the only extra trick needed.
The target is $\overline{A+B+C}$. The obvious first move — NOR A with B, then NOR that with C — does **not** work, because the first NOR has already complemented $A+B$:

$$\mathrm{NOR}(\mathrm{NOR}(A,B),\,C) = \overline{\overline{A+B}+C} = (A+B)\,\overline{C}$$

which is option (a). The fix is to un-complement in between. Take $\mathrm{NOR}(A,B)=\overline{A+B}$, feed it into a tied-input NOR to get $A+B$ back, then NOR *that* with C:

[34] courses/csse2010/boolean-algebra.pdf

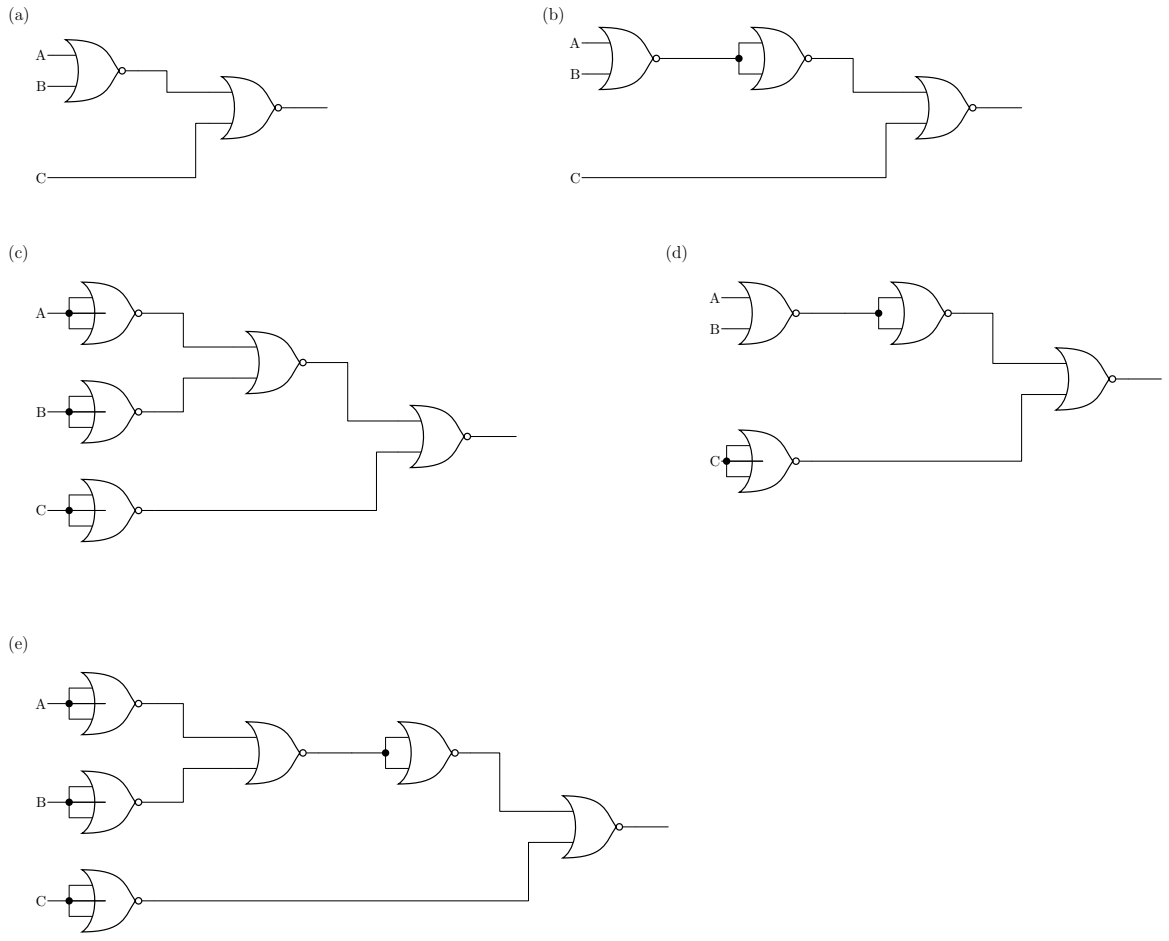


Figure 5: Question 11's five candidate NOR-only circuits. A gate whose two inputs are tied together (the bracket and dot on its left) is being used as an inverter.

$$\text{NOR}(\overline{\overline{A+B}},\, C) = \overline{(A+B)+C} = \overline{A+B+C}$$

Three gates, and that is exactly the arrangement drawn in **(b)**.
The others:

- **(c)** inverts all three inputs first, then NORs $\overline{A}$ with $\overline{B}$ to get $\overline{\overline{A}+\overline{B}} = A.B$, then NORs that with $\overline{C}$: $\overline{A.B+\overline{C}}$.
- **(d)** is (b) with C inverted on the way in, giving $\overline{A+B+\overline{C}}$.
- **(e)** is (c) with an extra inverter after the $A.B$ stage, giving $\overline{\overline{A.B}+\overline{C}} = A.B.C$.

Only (b) leaves all three variables uncomplemented under one bar.

Q12 — D flip-flop input from its output

Consider the flip-flop circuit shown below and the associated CLK and output (Q) signals. Which timing diagram shows input (A) values over time which would have resulted in the given output (Q).

(a) A

(b) A

(c) A

(d) A

(e) A

Answer

(c)
The symbol has a triangle on the clock input and no bubble, so it is a **positive-edge-triggered** D flip-flop: Q takes the value D had immediately before each **rising** CLK edge, and does nothing at any other time (see flip-flops-and-latches[35]).
There are four rising edges. Reading Q across them:

Rising edge	Q before	Q after	So A must have been
1	1	0	**0**
2	0	1	**1**
3	1	1	**1**
4	1	0	**0**

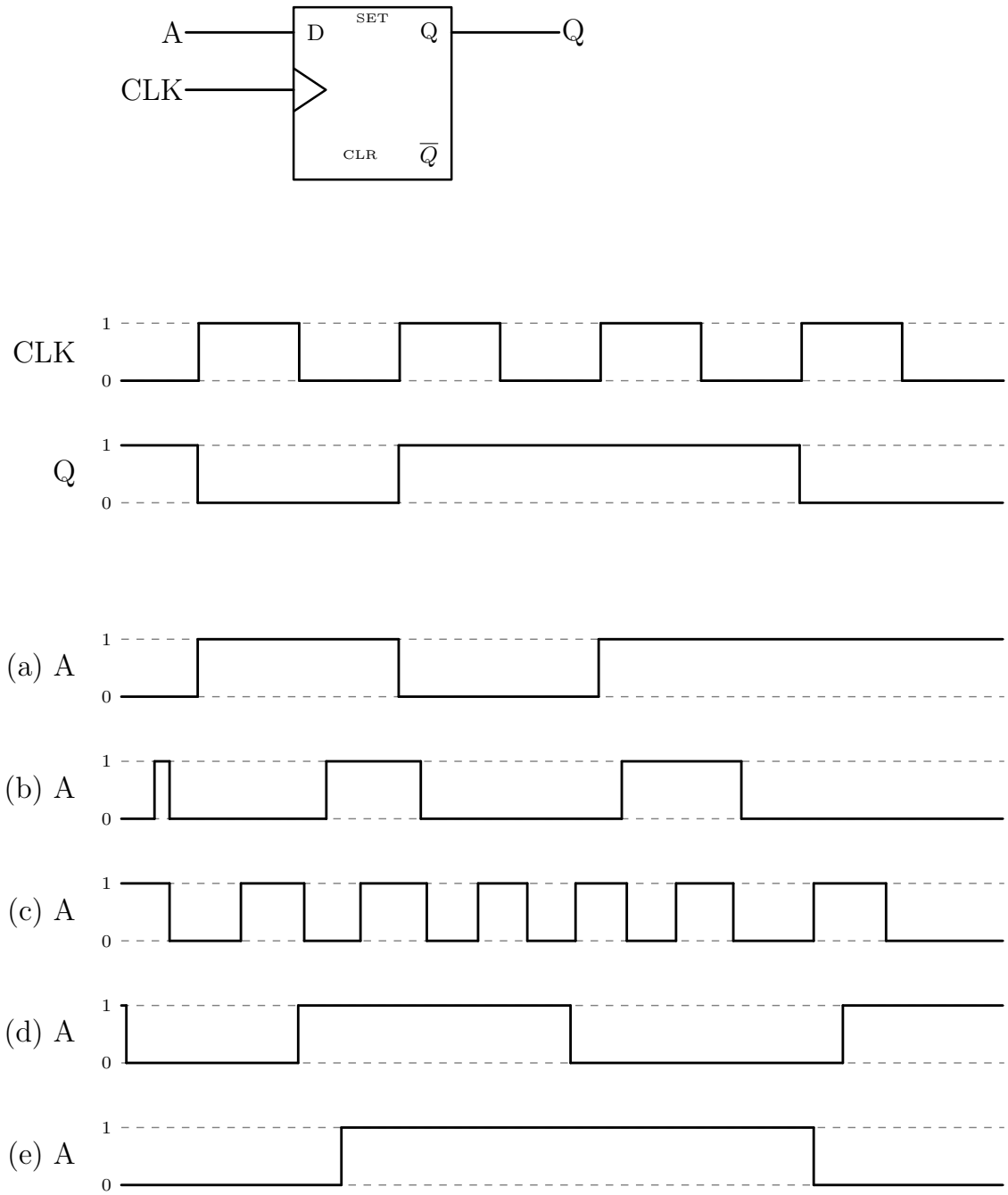


Figure 6: Question 12's flip-flop and waveforms: the D flip-flop symbol, then the given CLK and Q traces, then the five candidate A traces (a)–(e) on the same time axis.

The requirement is therefore $A = 0, 1, 1, 0$ at the four edges, and what A does between edges is irrelevant. Checking each candidate at the four edge times:

	edge 1	edge 2	edge 3	edge 4	
required	0	1	1	0	
(a)	0	1	**0**	1	fails at edge 3
(b)	0	1	**0**	0	fails at edge 3
(c)	0	1	1	0	**matches**
(d)	0	1	**0**	0	fails at edge 3
(e)	0	1	1	**1**	fails at edge 4

So (c). It is the busy, rapidly toggling trace, which is the point of the question: a flip-flop ignores everything except the value present at the clock edge, so a wildly active input can produce a placid output.
Trace (e) is the near miss — it is right for the first three edges and only falls too late, just *after* the fourth rising edge, so Q would have stayed high.

Q13 — Two's complement addition

What is the result of adding the 6-bit two's complement numbers 101010 and 110011?

(a) 011100

(b) 011101

(c) 011110

(d) 111011

(e) None of the above

Answer

(b)
Add the bit patterns as if they were unsigned — two's complement addition is ordinary binary addition — and throw away any carry out of the top bit:

```
  101010
 +110011
 -------
 1011101
```

[35] courses/csse2010/flip-flops-and-latches.pdf

Column by column from the right: $0+1=1$; $1+1=10$ so write 0 carry 1; $0+0+1=1$; $1+0=1$; $0+1=1$; $1+1=10$ so write 0 and carry 1 **out of the word**. Six bits of result: **011101**.
As a decimal check, $101010 = -32+8+2 = -22$ and $110011 = -32+16+2+1 = -13$, so the true sum is -35. That is outside the 6-bit two's complement range $[-32, 31]$, so this addition **overflows** — two negatives producing a positive-looking $011101 = +29$, and indeed $-35+64 = 29$. The question only asks what the adder produces, though, and that is 011101.
Option (d) is what you get by ORing the two operands rather than adding them.

Q14 — Detecting overflow

Which of the following operations will result in an overflow in 5-bit two's complement arithmetic?

(a) 01010 – 01110

(b) 10111 – 11100

(c) 11010 – 00111

(d) 01011 + 11111

(e) None of the above

Answer

(e)
Two's complement overflow can only happen when the two values being *added* have the **same sign** and the result comes out with the other sign. A subtraction $P-Q$ is really $P+(-Q)$, so what matters is the sign of P and of $-Q$. The 5-bit range is $[-16, +15]$.
Take each in turn, converting to decimal and also checking the sign rule.
(a) 01010 − 01110 $= 10-14 = -4$. Adding a positive to a negative can never overflow, and -4 is in range. In binary: $01010+10010 = 11100 = -4$. No overflow.
(b) 10111 − 11100 $= -9-(-4) = -5$. Again opposite signs going into the adder (-9 and $+4$). $10111+00100 = 11011 = -5$. No overflow.
(c) 11010 − 00111 $= -6-7 = -13$. Here both addends *are* negative (-6 and -7), so overflow is possible — but -13 still fits in $[-16, 15]$. $11010+11001 = 1\,10011$; discard the carry out and $10011 = -13$, which is negative as expected. No overflow.
(d) 01011 + 11111 $= 11+(-1) = 10$. Opposite signs. $01011+11111 = 1\,01010$; discard the carry and $01010 = 10$. No overflow.
Note that (d) is a good reminder that a **carry out is not overflow** — the carry out of the top bit here is 1, yet the answer is perfectly correct. None of the four overflows, so

the answer is (e).

Q15 — Overflow on an addition

Consider the binary addition operation 010111 + 001001. Which of the following statements is true if the numbers are interpreted as two's complement?

(a) The result is positive and no overflow occurs

(b) The result should be positive but overflow occurs

(c) The result is negative and no overflow occurs

(d) The result should be negative but overflow occurs

(e) The result is zero

Answer

(b)
Both operands have a 0 in the sign position, so both are positive: $010111 = 16+4+2+1 = 23$ and $001001 = 8+1 = 9$. The true sum is 32, and the 6-bit two's complement range is only $[-32, +31]$ — so 32 does not fit.
Doing the addition:

$$\begin{array}{r} 010111 \\ +001001 \\ \hline 100000 \end{array}$$

The result pattern 100000 has a 1 in the sign position, so the adder is reporting -32. **Two positives producing a negative is the definition of two's complement overflow**, which makes (b) the right description: the result should have been positive, but overflow occurred.
Note that no carry came *out* of the word here, which again shows carry-out and overflow are different things (compare Q14).

Q16 — Multiplexer as function generator

Consider the following multiplexer. What must the inputs A,B,C,D be so that the multiplexer implements the function $X = S_1 \oplus S_0 \oplus \overline{G}$?

(a) A= 0 B= 1 C= 1 D= 0

(b) A= G B= $\overline{G}$ C= $\overline{G}$ D= G

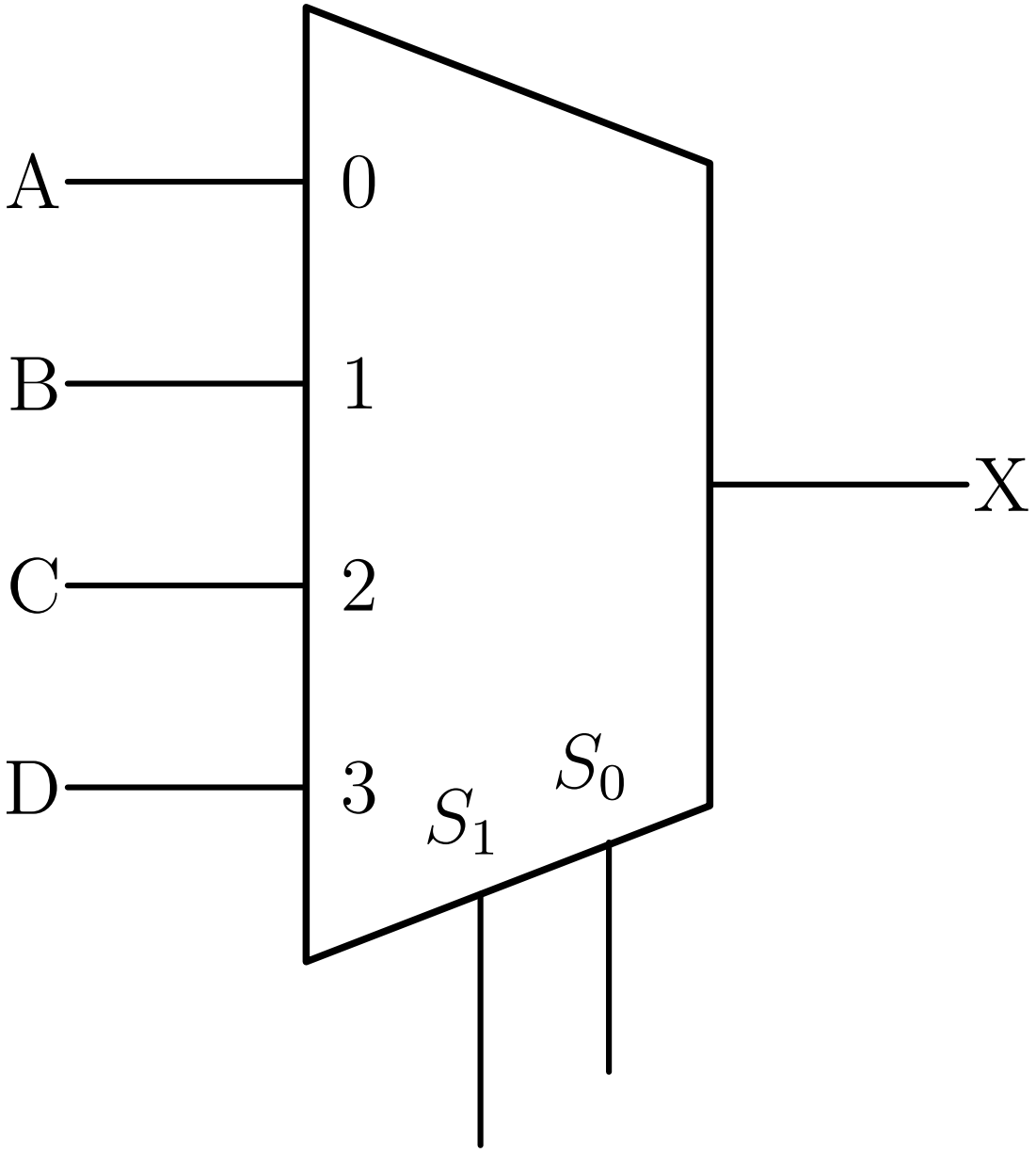


Figure 7: Question 16's 4-to-1 multiplexer: data inputs A, B, C and D on lines 0 to 3, select inputs S1 and S0, output X.

(c) A= $\overline{G}$ B= 1 C= 1 D= 1

(d) A= $\overline{G}$ B= G C= G D= $\overline{G}$

(e) None of the above

Answer

(d)
A multiplexer used this way is a lookup table: whatever is wired to input n appears at X whenever the select lines hold n. So for each of the four values of S_1S_0, evaluate the required function and wire that expression to the matching input (see combinational-logic-blocks[36]).

$$X = S_1 \oplus S_0 \oplus \overline{G}$$

S_1S_0	Input selected	$S_1 \oplus S_0$	Required $X = (S_1 \oplus S_0) \oplus \overline{G}$
00	A	0	$0 \oplus \overline{G} = \overline{G}$
01	B	1	$1 \oplus \overline{G} = G$
10	C	1	$1 \oplus \overline{G} = G$
11	D	0	$0 \oplus \overline{G} = \overline{G}$

using $0 \oplus P = P$ and $1 \oplus P = \overline{P}$. So $A = \overline{G}$, $B = G$, $C = G$, $D = \overline{G}$ — option (d). Option (b) is the same pattern inverted throughout, which is what you get by dropping the bar on G in the target function.

Q17 — Reading an 8-input multiplexer

Consider the following multiplexer. Which of the following statements is TRUE? (+ indicates the logical OR operation)

(a) If A is 0, $X = \overline{C}$ If A is 1, $X = \overline{B + C}$

(b) If A is 0, $X = \overline{B}$ If A is 1, $X = \overline{B + C}$

(c) If A is 0, $X = \overline{C}$ If A is 1, $X = \overline{B.C}$

(d) If A is 0, $X = C$ If A is 1, $X = \overline{B.C}$

(e) None of the above

[36] courses/csse2010/combinational-logic-blocks.pdf

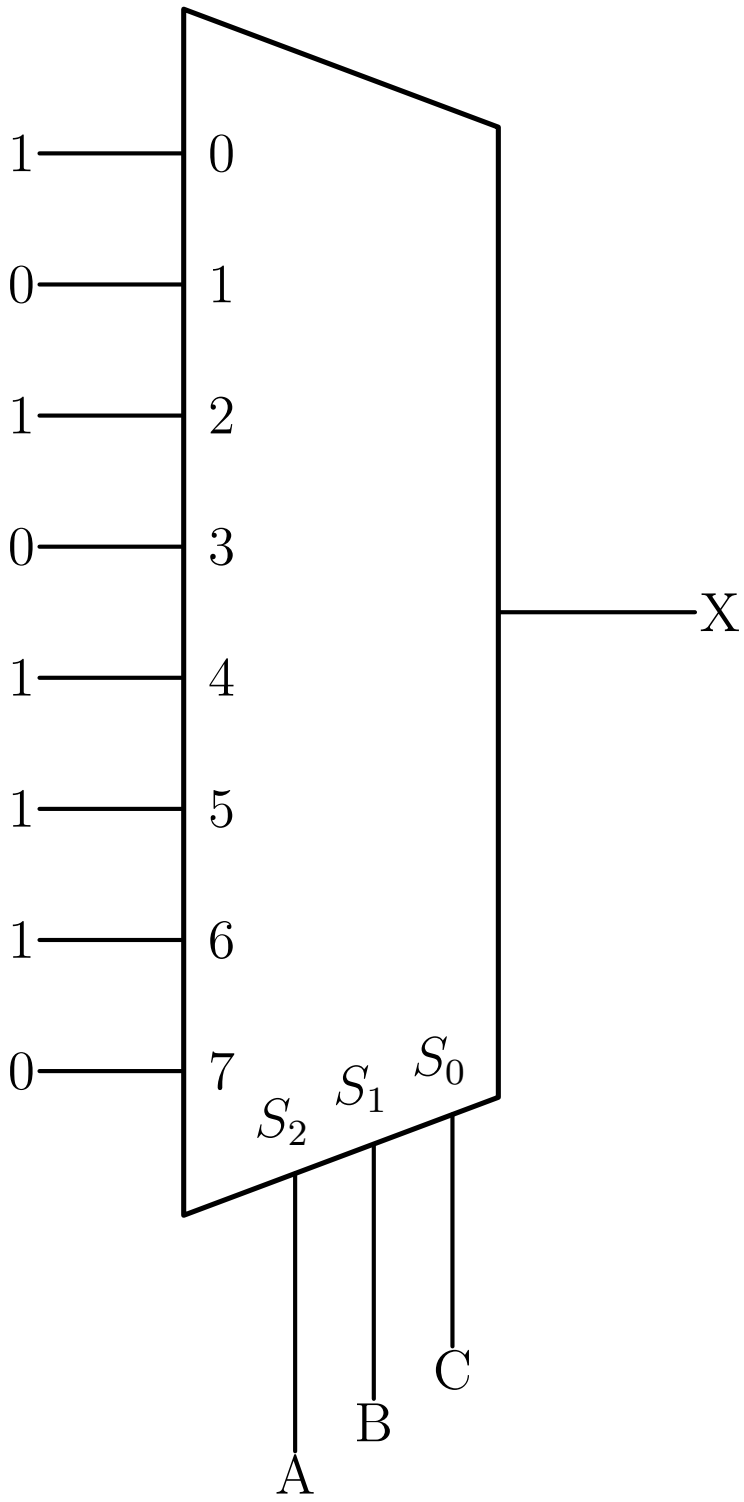


Figure 8: Question 17's 8-to-1 multiplexer. The eight data inputs are hardwired to the constants 1, 0, 1, 0, 1, 1, 1, 0 on lines 0 to 7; the select inputs S2, S1 and S0 are driven by A, B and C respectively.

Answer

(c)
A is the most significant select bit, so it splits the eight constants into two halves: $A = 0$ selects among inputs 0–3, $A = 1$ among inputs 4–7. Within each half, BC picks the row.
With $A = 0$ the four constants are, for $BC = 00, 01, 10, 11$:

B	C	Input	Value
0	0	0	1
0	1	1	0
1	0	2	1
1	1	3	0

X is 1 exactly when $C = 0$, whatever B does — so $X = \overline{C}$.
With $A = 1$ the constants are:

B	C	Input	Value
0	0	4	1
0	1	5	1
1	0	6	1
1	1	7	0

X is 0 only when B and C are both 1 — so $X = \overline{B.C}$.
Both halves match option (c). Option (a) has the right first half but reads the second half as a NOR rather than a NAND, which would need the last *three* constants to be 0 rather than just the last one.

Q18 — Load/shift register trace

Consider the circuit below. If A_1 is 1 and A_0 is 0, what are the values of Q_1 and Q_0 after four rising clock edges if FN is 0 for one rising clock edge then 1 for three rising clock edges?

(a) $Q_1 = 0$ $Q_0 = 0$

(b) $Q_1 = 0$ $Q_0 = 1$

(c) $Q_1 = 1$ $Q_0 = 0$

(d) $Q_1 = 1$ $Q_0 = 1$

(e) Insufficient information is provided to determine the values

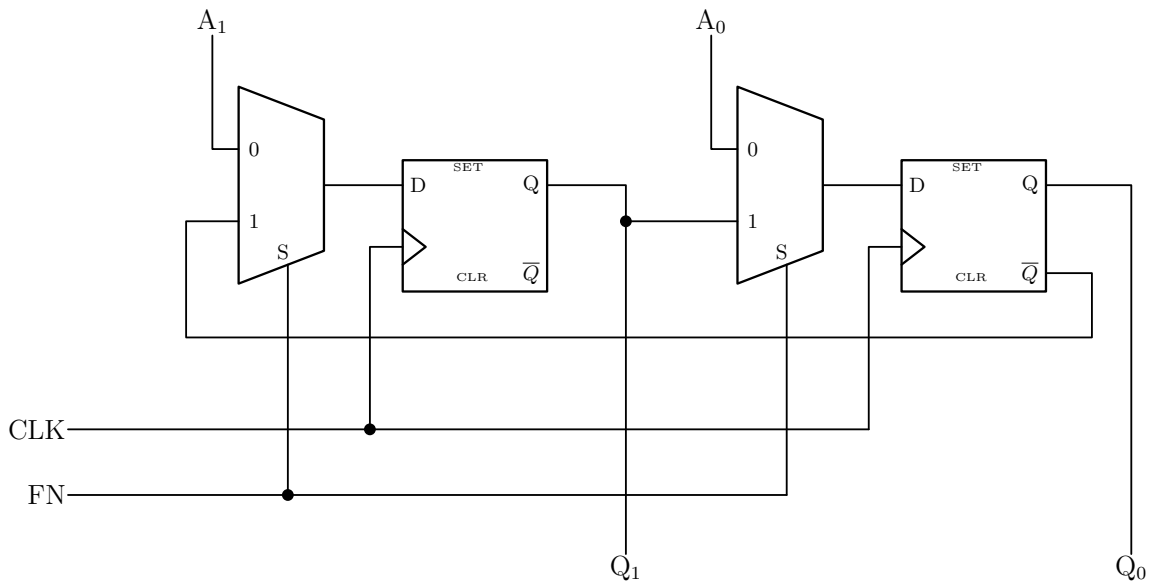


Figure 9: Question 18's circuit: two D flip-flops, each fed by a 2-to-1 multiplexer under the shared select FN. On FN = 0 the flip-flops load A1 and A0; on FN = 1 the left one takes Q0-bar and the right one takes Q1, which makes the pair a twisted-ring counter.

Answer

(a)

First read the two multiplexers. Both have FN on their select input, so:

$$D_1 = \begin{cases} A_1 & \text{FN} = 0 \\ \overline{Q_0} & \text{FN} = 1 \end{cases} \qquad D_0 = \begin{cases} A_0 & \text{FN} = 0 \\ Q_1 & \text{FN} = 1 \end{cases}$$

The $\overline{Q_0}$ feedback is the long run along the bottom, from the right flip-flop's $\overline{Q}$ pin all the way back to the left multiplexer's input 1. With FN = 1 this is a **twisted-ring (Johnson) counter**: the pair shifts left-to-right and the complement of the last bit re-enters at the front.

Now trace the four edges. Both flip-flops are clocked together, so both take their new values from the *old* Q values at each edge.

Edge	FN	D_1	D_0	New Q_1Q_0
—	—	—	—	starting value unknown
1	0	$A_1 = 1$	$A_0 = 0$	**1 0**
2	1	$\overline{Q_0} = \overline{0} = 1$	$Q_1 = 1$	**1 1**
3	1	$\overline{Q_0} = \overline{1} = 0$	$Q_1 = 1$	**0 1**
4	1	$\overline{Q_0} = \overline{1} = 0$	$Q_1 = 0$	**0 0**

After four edges $Q_1 = 0$ and $Q_0 = 0$ — option (a).
Option (e) is the trap: the starting contents of the flip-flops genuinely *are* unknown, but the first edge is a parallel load that overwrites both of them, so the answer is fully determined from edge 1 onwards.

Q19 — Synchronous counter sequence

What sequence will the following synchronous counter count through, if it starts at $Q_1Q_0 = 00$?

(a) Q_1Q_0: 00 → 10 → 01 → 11 → 00 → …

(b) Q_1Q_0: 00 → 10 → 11 → 00 → …

(c) Q_1Q_0: 00 → 01 → 10 → 11 → 00 → …

(d) Q_1Q_0: 00 → 01 → 11 → 00 → …

(e) None of the above

Answer

(d)
Read the excitation equations off the circuit. Both gates take the same pair of inputs — the two state bits fed back — so:

$$D_1 = Q_1 \oplus Q_0 \qquad D_0 = \overline{Q_1.Q_0}$$

Now build the state table (see counters[37]):

Q_1Q_0	$D_1 = Q_1 \oplus Q_0$	$D_0 = \overline{Q_1Q_0}$	Next state
00	0	1	**01**
01	1	1	**11**
11	0	0	**00**
10	1	1	11

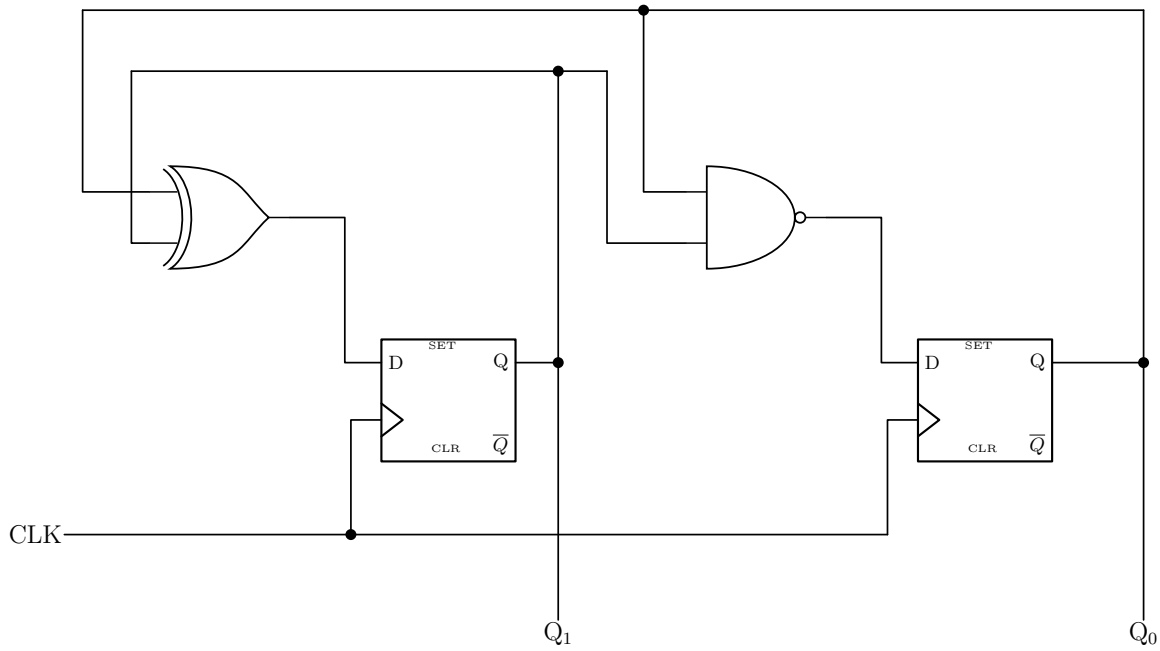


Figure 10: Question 19's synchronous counter. Both gates see the same two feedback lines: Q0 comes back along the upper run and Q1 along the lower one, so D1 is the XOR of the two state bits and D0 their NAND. The source paper draws the XOR mirrored, with its output on the left; it is drawn the usual way round here, and the connections are unchanged.

Starting from 00 the counter runs

$$00 \to 01 \to 11 \to 00 \to \ldots$$

a three-state cycle, which is option (d). State 10 is outside the cycle; it drops into 11 on the next edge and joins the loop, so the counter is self-correcting.
The near miss is (b), $00 \to 10 \to 11 \to 00$: the same three-state cycle with the two bits swapped over, which is what you get if Q_1 and Q_0 are mixed up when reading the circuit.

Q20 — Flip-flops for an output sequence

What is the minimum number of flip-flops needed to implement a state machine whose outputs always go through the sequence: 00001→00011→00111→11000→10000→00000→00001→… ?

(a) 3

(b) 4

(c) 5

(d) 6

(e) None of the above

Answer

(a)
The outputs are five bits wide, which is the distractor — it tempts option (c), one flip-flop per output bit. But the flip-flops hold the **state**, not the output; the output is whatever combinational logic decodes from the state (see state-machines[38]).
So count the states. The sequence visits

$$00001,\ 00011,\ 00111,\ 11000,\ 10000,\ 00000$$

and then repeats. All six patterns are distinct, and because the machine "always" goes through this sequence, each output value identifies exactly one state — six states.
N flip-flops give 2^N states, so we need the smallest N with $2^N \geq 6$:

$$2^2 = 4 < 6 \qquad 2^3 = 8 \geq 6$$

Three flip-flops, option (a). The five output bits are then produced by output logic decoding those three, which is exactly the Moore-machine structure.

[37] courses/csse2010/counters.pdf

Q21 — State diagram to state table

Consider the state diagram below.

Inputs: A
Outputs: X,Y

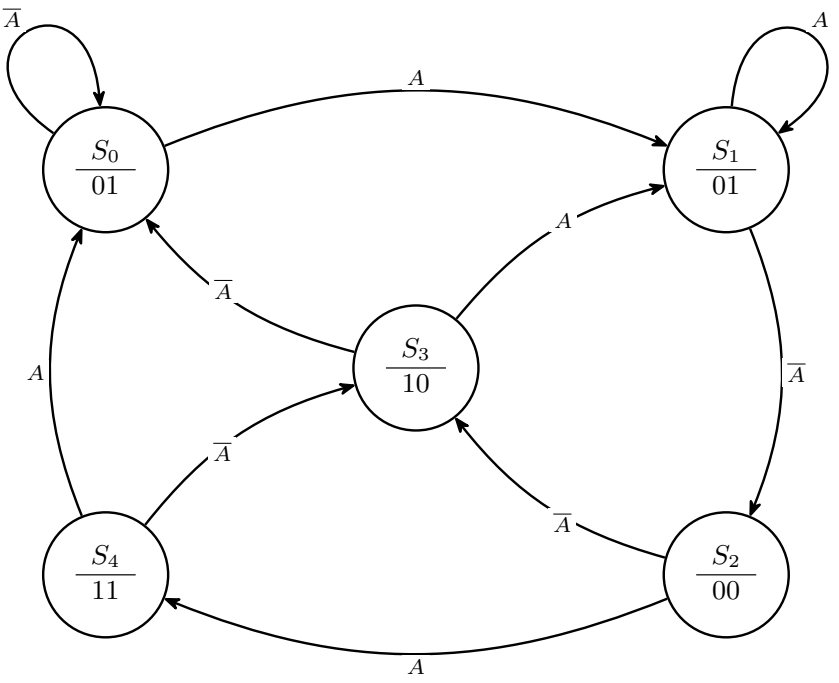


Figure 11: Question 21's state diagram. Each state shows its name above the rule and its X,Y outputs below; the single input is A, and every state has one arrow for A and one for A-bar.

Which of the following is an equivalent state table?

(a)

Current State	Input (A)	Next State	Outputs (X,Y)
S0	0	S0	01
S0	1	S1	01
S1	0	S2	01
S1	1	S1	01
S2	0	S3	00

[38] courses/csse2010/state-machines.pdf

Current State	Input (A)	Next State	Outputs (X,Y)
S2	1	S4	00
S3	0	S0	10
S3	1	S1	10
S4	0	S0	11
S4	1	S3	11

(b)

Current State	Input (A)	Next State	Outputs (X,Y)
S0	0	S0	01
S0	1	S1	01
S1	0	S2	01
S1	1	S2	01
S2	0	S3	00
S2	1	S4	00
S3	0	S0	10
S3	1	S1	10
S4	0	S3	11
S4	1	S0	11

(c)

Current State	Next State (A=0)	Next State (A=1)	Outputs (X,Y)
S0	S0	S1	01
S1	S2	S1	01
S2	S3	S4	00
S3	S0	S1	10
S4	S3	S0	11

(d)

Current State	Next State (A=0)	Next State (A=1)	Outputs (X,Y)
S0	S0	S1	01
S1	S2	S1	10
S2	S3	S4	00
S3	S0	S1	10
S4	S3	S0	11

(e) None of the above

Answer

(c)
Read the ten arrows off the diagram, remembering that a label A means the transition is taken when $A = 1$ and $\overline{A}$ when $A = 0$, and that this is a Moore machine so the outputs belong to the state, not the transition (see state-machines[39]).

Current state	$A = 0$	$A = 1$	Outputs X, Y
S_0	S_0 (self loop, $\overline{A}$)	S_1 (top arc, A)	01
S_1	S_2 (down the right, $\overline{A}$)	S_1 (self loop, A)	01
S_2	S_3 ($\overline{A}$)	S_4 (bottom arc, A)	00
S_3	S_0 ($\overline{A}$)	S_1 (A)	10
S_4	S_3 ($\overline{A}$)	S_0 (A)	11

Each state has exactly two arrows leaving it, one per value of A, so the diagram is complete. This table is option **(c)** exactly.
The other three each differ in one place:

- **(a)** swaps S_4's two destinations — it has $S_4 \to S_0$ on $A = 0$ and $S_4 \to S_3$ on $A = 1$, the wrong way round.
- **(b)** sends S_1 to S_2 on **both** input values, losing S_1's self loop.
- **(d)** is (c) with S_1's output changed from 01 to 10.

Q22 — Flip-flops for a state machine

What is the minimum number of flip-flops needed to build the state machine shown in question 21?

(a) 1

(b) 2

(c) 3

(d) 4

(e) 5

[39] courses/csse2010/state-machines.pdf

Answer

(c)
The diagram in Q21 has five states: S_0, S_1, S_2, S_3 and S_4. With N flip-flops there are 2^N possible codes, so we need the smallest N with $2^N \geq 5$:

$$2^2 = 4 < 5 \qquad 2^3 = 8 \geq 5$$

so **three** flip-flops, with three of the eight codes left unused.
Option (e), five, is the one-hot encoding — legal and often convenient, but not the *minimum*. Option (b), two, would only stretch to four states.

Q23 — Tracing a state machine

Consider the state machine represented by the state diagram in question 21 above. What sequence of states is traversed if the machine starts in state S3 and A goes through the sequence 10001 (i.e. 1 followed by three 0s followed by a 1)?

(a) S3→S0→S0→S0→S0→S1

(b) S3→S1→S2→S3→S0→S1

(c) S3→S1→S2→S4→S3→S1

(d) S3→S0→S1→S2→S3→S1

(e) None of the above

Answer

(b)
Walk the table from Q21 one input bit at a time, starting in S_3:

Step	Current state	A	Next state
1	S_3	**1**	S_1
2	S_1	**0**	S_2
3	S_2	**0**	S_3
4	S_3	**0**	S_0
5	S_0	**1**	S_1

so the machine visits

$$S_3 \to S_1 \to S_2 \to S_3 \to S_0 \to S_1$$

which is option (b).
Option (c) goes wrong at step 3 by taking S_2's $A = 1$ arrow instead of its $A = 0$ one; option (a) takes S_3's $\overline{A}$ arrow on the first bit, which is 1.

Q24 — ALU control settings

Consider the ALU bit slice below.

For an ALU made up of 8 of the bit slices above, which of the following control input settings will NOT result in the ALU output being $\overline{B}$? (The "Right Shift In" control input refers to that for the most significant bit; the "Carry in" control input refers to that for the least significant bit.)

(a) ENA=0 INVA=0 ENB=1 INVB=0 F_2=0 F_1=1 F_0=1 Right Shift In=0 Carry in=0

(b) ENA=0 INVA=0 ENB=1 INVB=1 F_2=0 F_1=0 F_0=1 Right Shift In=0 Carry in=0

(c) ENA=0 INVA=1 ENB=1 INVB=0 F_2=0 F_1=1 F_0=0 Right Shift In=0 Carry in=0

(d) ENA=0 INVA=1 ENB=1 INVB=1 F_2=0 F_1=0 F_0=0 Right Shift In=0 Carry in=0

(e) All of the above result in the ALU output being $\overline{B}$

Answer

(e)
Start by writing down what the slice actually computes. The conditioning gates give the two internal rails

$$X = (A \cdot \text{ENA}) \oplus \text{INVA} \qquad Y = (B \cdot \text{ENB}) \oplus \text{INVB}$$

and the multiplexer, selected by $F_2F_1F_0$ read as a binary number, chooses between:

$F_2F_1F_0$	Input	Output
000	0	$X.Y$
001	1	$X + Y$
010	2	$\overline{X.Y}$
011	3	$\overline{X + Y}$
100	4	Carry In
101	5	$\overline{X}$

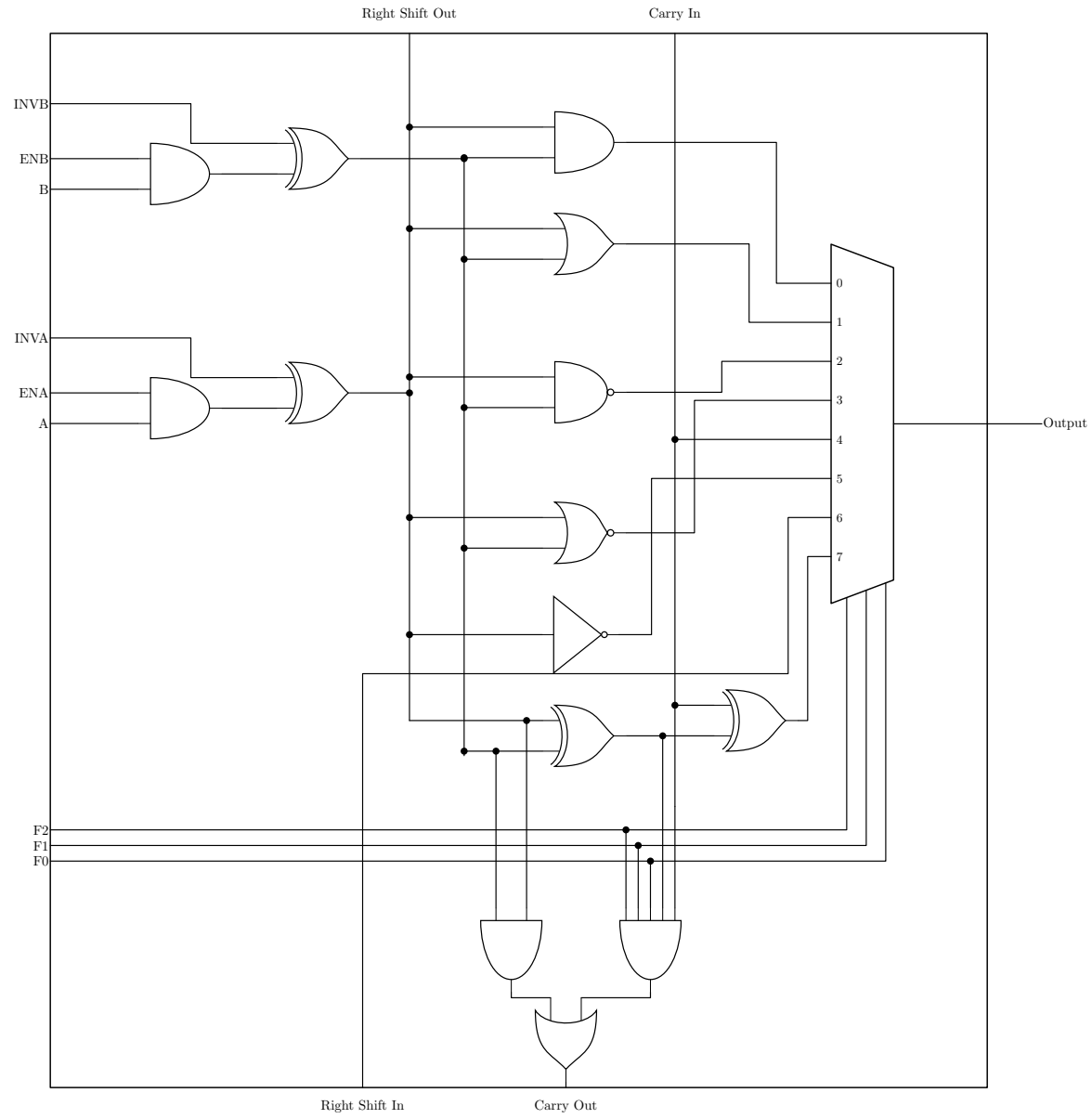


Figure 12: Question 24's ALU bit slice. The ENA/INVA and ENB/INVB gates on the left condition A and B into the two internal rails X and Y; the five logic blocks and the full adder all work from those rails, and the 8-input multiplexer under F2 F1 F0 picks which result reaches the output. Right Shift Out is the X rail itself. The carry logic at the bottom points downwards as in the source: an AND on the two rails for the generate term, and a five-input AND for the propagate term, whose extra three inputs are F2, F1 and F0 — so a carry only travels when the adder is the selected function.

110	6	Right Shift In
111	7	$X \oplus Y \oplus C_{in}$ (the adder's sum)

Every option here has ENB = 1, so Y is B or $\overline{B}$; and every option has ENA = 0, so X is a **constant** — 0 when INVA = 0 and 1 when INVA = 1. Feeding a constant into one side of a two-input gate is exactly how this ALU turns a logic block into a pass-through or an inverter.
(a) $X = 0$, $Y = B$, select 011 = NOR: $\overline{0 + B} = \overline{B}$.
(b) $X = 0$, $Y = \overline{B}$, select 001 = OR: $0 + \overline{B} = \overline{B}$.
(c) $X = 1$, $Y = B$, select 010 = NAND: $\overline{1 \cdot B} = \overline{B}$.
(d) $X = 1$, $Y = \overline{B}$, select 000 = AND: $1 \cdot \overline{B} = \overline{B}$.
All four produce $\overline{B}$, so the answer is (e). Note that none of them selects input 4, 6 or 7, which is why the Right Shift In and Carry In settings never matter — they are there to make the options look longer than they are.
The paper does not label which select pin of the multiplexer carries which weight, but two things settle it. First, the figure routes F_2 to the **leftmost** of the three select pins and F_0 to the rightmost, and everywhere else in this paper — the multiplexers in Q16 and Q17 — the leftmost select pin is the most significant (S_1 then S_0; S_2 then S_1 then S_0). Second, reading F_2 as the most significant bit is what makes all four settings land on $\overline{B}$; any other assignment sends at least one of them to the Carry In or Right Shift In input instead, which would give a question with three wrong answers rather than none.
The four settings are the four ways of getting $\overline{B}$ out of the four two-input logic blocks: complement the function (NOR/NAND) or complement the operand (INVB), and choose the constant that makes the other input transparent.

Q25 — ALU control settings

For the ALU in question 24, which of the following control input settings will result in the ALU output being $A.B$?

(a) ENA=1 INVA=0 ENB=1 INVB=0 F_2=0 F_1=0 F_0=1 Right Shift In=0 Carry in=0

(b) ENA=1 INVA=0 ENB=1 INVB=0 F_2=0 F_1=1 F_0=0 Right Shift In=0 Carry in=0

(c) ENA=1 INVA=0 ENB=1 INVB=0 F_2=0 F_1=1 F_0=1 Right Shift In=0 Carry in=0

(d) ENA=1 INVA=1 ENB=1 INVB=1 F_2=0 F_1=1 F_0=1 Right Shift In=0 Carry in=0

(e) None of the above

Answer

(d)
Same slice as Q24, same two steps: work out X and Y, then look up what $F_2F_1F_0$ selects.
(a) ENA = ENB = 1 with both INV inputs 0, so $X = A$ and $Y = B$. Select 001 = 1 is the OR block: $A + B$.
(b) Same X and Y; select 010 = 2 is the NAND block: $\overline{A.B}$. — the complement of what is wanted, which is the tempting wrong answer.
(c) Same X and Y; select 011 = 3 is the NOR block: $\overline{A + B}$.
(d) INVA = INVB = 1, so $X = \overline{A}$ and $Y = \overline{B}$; select 011 = 3 is the NOR block:

$$\overline{\overline{A} + \overline{B}} = \overline{\overline{A}} \cdot \overline{\overline{B}} = A.B$$

by De Morgan.
So (d) — the slice has no AND-with-uncomplemented-inputs route selected here, and instead reaches $A.B$ the De Morgan way, by inverting both operands into a NOR. Selecting the AND block (000) with INVA = INVB = 0 would have done it directly, but that combination is not on the list.