Sample Mid-semester Exam #5

Table of contents

How to use this . . . 2
Question index . . . 2
Q1 — Excess-128 representation . . . 4
Q2 — Hexadecimal to octal . . . 5
Q3 — Two's complement range . . . 5
Q4 — Converting between representations . . . 6
Q5 — XNOR gate chain . . . 8
Q6 — Boolean simplification . . . 9
Q7 — Gate circuit for an expression . . . 10
Q8 — Expression to truth table . . . 11
Q9 — Timing diagram to expression . . . 13
Q10 — Sum-of-products form . . . 15
Q11 — 3-input NOR from 2-input NANDs . . . 15
Q12 — D flip-flop timing . . . 17
Q13 — Two's complement addition . . . 19
Q14 — Overflow detection . . . 20
Q15 — Overflow and result sign . . . 21
Q16 — Multiplexer as function generator . . . 22
Q17 — Multiplexer output function . . . 24
Q18 — Multiplexer-fed flip-flops . . . 26
Q19 — Synchronous counter trace . . . 27
Q20 — Flip-flops for a count sequence . . . 29
Q21 — State diagram to state table . . . 29
Q22 — State encoding width . . . 32
Q23 — Tracing a state sequence . . . 33
Q24 — ALU function select . . . 34
Q25 — ALU constant generation . . . 36

This is UQ's fifth sample mid-semester paper for CSSE2010/CSSE7201 Introduction to Computer Systems, printed for the Semester One mid-semester examinations, 2020, and set for St

Lucia campus students. It runs for 60 minutes with 10 minutes of reading time, under open-book conditions — no electronic aids beyond a Casio FX82-series or UQ-approved calculator — and answers go on a supplied True/False and Multiple Choice answer sheet. There are 25 questions; each correct answer scores one mark, and an incorrect, missing or multiple answer scores zero, so there is no penalty for guessing. See csse2010[1] for the course note.

> ⚠ 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. The first is a timed run: 25 questions in 60 minutes, working straight down the paper. On the site every answer is collapsed behind a disclosure triangle, so nothing is spoiled until you click it — that is the version to practise from. The second is targeted practice: use the question index below to pick out only the questions for one week or one concept, do those, and follow the concept link when the working does not make sense.

In the PDF build the collapsed callouts are always open, so the PDF is an answer key rather than a practice paper.

Question index

Q	Topic	Week	Concept notes
1	Excess-128 representation	1	binary-number-representations[2] · week1-lab-number-representations[3]
2	Hexadecimal to octal	1	binary-number-representations[4] · week1-lab-number-representations[5]
3	Two's complement range	1	binary-number-representations[6] · week1-lab-number-representations[7]

[1] courses/csse2010/csse2010.pdf
[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
[6] courses/csse2010/binary-number-representations.pdf
[7] courses/csse2010/week1-lab-number-representations.pdf

Q	Topic	Week	Concept notes
4	Converting between representations	1	binary-number-representations[8] · week1-lab-number-representations[9]
5	XNOR gate chain	1	logic-gates[10]
6	Boolean simplification	2	boolean-algebra[11]
7	Gate circuit for an expression	1	logic-gates[12]
8	Expression to truth table	2	boolean-algebra[13]
9	Timing diagram to expression	2	timing-diagrams[14]
10	Sum-of-products form	2	boolean-algebra[15]
11	3-input NOR from 2-input NANDs	1	logic-gates[16]
12	D flip-flop timing	3	flip-flops-and-latches[17]
13	Two's complement addition	2	binary-number-representations[18]
14	Overflow detection	2	binary-number-representations[19]
15	Overflow and result sign	2	binary-number-representations[20]
16	Multiplexer as function generator	2	combinational-logic-blocks[21]
17	Multiplexer output function	2	combinational-logic-blocks[22]
18	Multiplexer-fed flip-flops	3	flip-flops-and-latches[23]

[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/logic-gates.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
[21] courses/csse2010/combinational-logic-blocks.pdf
[22] courses/csse2010/combinational-logic-blocks.pdf
[23] courses/csse2010/flip-flops-and-latches.pdf

Q	Topic	Week	Concept notes
19	Synchronous counter trace	4	counters[24]
20	Flip-flops for a count sequence	4	state-machines[25]
21	State diagram to state table	4	state-machines[26]
22	State encoding width	4	state-machines[27]
23	Tracing a state sequence	4	state-machines[28]
24	ALU function select	5	arithmetic-logic-unit[29]
25	ALU constant generation	5	arithmetic-logic-unit[30]

Q1 — Excess-128 representation

What is -75_{10} expressed in 8-bit excess-128 format?

(a) 00110101

(b) 01001011

(c) 10110100

(d) 10110101

(e) 11001011

Answer

(a)
Excess-128 stores a value N as the plain unsigned binary of $N + 128$. So

$$-75 + 128 = 53$$

[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

and $53 = 32 + 16 + 4 + 1$, which in 8 bits is 00110101.
Sanity check against the two's complement form: -75 in 8-bit two's complement is 10110101 (option (d)), and excess-128 is exactly two's complement with the sign bit flipped — flipping the leading 1 to 0 gives 00110101 again. See binary-number-representations[31].

Q2 — Hexadecimal to octal

What is the minimum number of octal digits needed to represent the number $1ACED2010_{16}$?

(a) 7

(b) 9

(c) 11

(d) 12

(e) None of the above

Answer

(c)
Go via binary, since both bases are powers of two. The number has nine hex digits, and each expands to four bits:

$$1\,A\,C\,E\,D\,2\,0\,1\,0 \;\rightarrow\; 0001\ 1010\ 1100\ 1110\ 1101\ 0010\ 0000\ 0001\ 0000$$

That is 36 bits with three leading zeros, so 33 significant bits. Octal digits take three bits each, and $\lceil 33/3 \rceil = 11$.
Regrouping the 36 bits into twelve groups of three from the right confirms it — the first group is 000 and drops away:

$$000\ 110\ 101\ 100\ 111\ 011\ 010\ 010\ 000\ 000\ 010\ 000 \;=\; 65473220020_8$$

which is 11 digits.

Q3 — Two's complement range

A 5-bit number in two's complement format can represent integers from

(a) -32 to 31

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

(b) −31 to 31

(c) −16 to 15

(d) −15 to 15

(e) 0 to 31

Answer

(c)
An n-bit two's complement number covers -2^{n-1} to $2^{n-1}-1$. With $n=5$ that is −16 to 15: 10000 is the most negative value and 01111 the most positive. The asymmetry is the giveaway — two's complement has one more negative value than positive, which rules out (b) and (d) immediately, and (a) is the 6-bit range. See binary-number-representations[32].

Q4 — Converting between representations

Which of the following statements about binary number representations is FALSE?

(a) Any 8-bit two's complement number can be converted to excess-128 representation by inverting the sign-bit

(b) Any negative n-bit two's complement number can be converted to ones' complement representation by subtracting 1

(c) Any negative n-bit ones' complement number can be converted to two's complement representation by adding 1

(d) Any negative n-bit ones' complement number can be converted to signed-magnitude representation by inverting all bits except the sign-bit

(e) None of the above

Answer

(e) — all four statements are true, so none of them is the false one.

(a) For a value v in $[-128, 127]$, two's complement stores $v \bmod 256$ and excess-128 stores $v+128$. For $v \geq 0$ the stored patterns differ by exactly +128, which sets a bit 7 that was 0; for $v < 0$ they differ by exactly −128, which clears a bit 7 that was 1. Either way, flip the sign bit. Q1 is a worked instance: −75 is 10110101 in two's complement and 00110101 in excess-128.

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

(b) A negative v is stored as $2^n + v$ in two's complement and $2^n - 1 + v$ in ones' complement, a difference of exactly 1. Example with $n = 4$: -3 is 1101 two's complement, and $1101 - 1 = 1100$, which is ones' complement -3. The one value this misses is -2^{n-1}, which two's complement can hold and ones' complement cannot, so strictly nothing converts it; read as the standard conversion rule, which is how the paper means it, (b) is true.

(c) The same identity read backwards, so it is true too: $1100 + 1 = 1101$.

(d) Ones' complement represents $-m$ as the bitwise complement of the pattern for $+m$. Since $+m$ has a 0 sign bit, the magnitude bits of $-m$ are the complement of the magnitude of m; inverting them back recovers m and leaves the 1 sign bit in place, which is exactly signed-magnitude. Example with $n = 4$: -3 is 1100 in ones' complement, and inverting the low three bits gives \$1011 = \$ signed-magnitude -3.

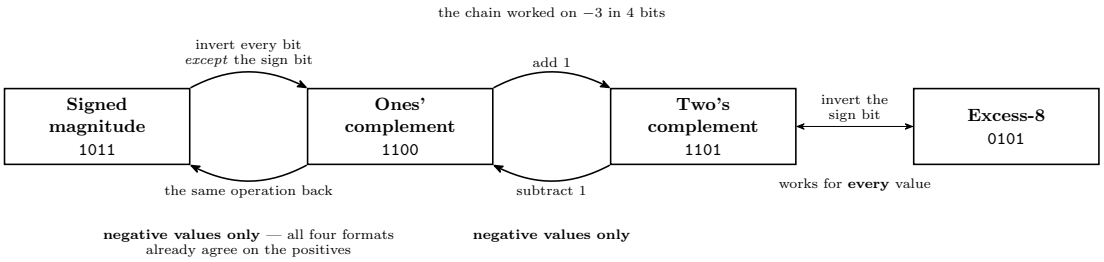


Figure 1: The four statements as one map, worked on a single value. Options (d), (b)/(c) and (a) are the three links in the chain, read from left to right; each arrow is one of the paper's four claims. The bit patterns are 4-bit representations of -3, so you can check every arrow by eye: invert the magnitude bits, subtract or add one, flip the sign bit. The two left-hand links hold for negative values only, which is not a limitation in practice because all four formats already agree on the positives — the right-hand link is the one that works either way.

See binary-number-representations[33] and week1-lab-number-representations[34] for all four formats side by side.

34 courses/csse2010/binary-number-representations.pdf

34 courses/csse2010/week1-lab-number-representations.pdf

Q5 — XNOR gate chain

What function is implemented by the following circuit?

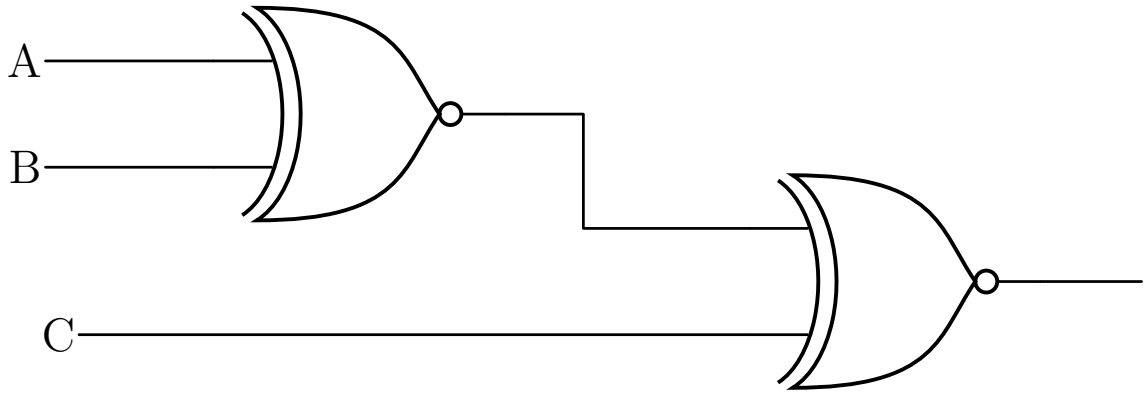


Figure 2: Question 5's circuit: A and B into an XNOR gate, whose output goes with C into a second XNOR gate.

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

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

(c) $\overline{A} \oplus \overline{B} \oplus \overline{C}$

(d) $A \oplus B \oplus C$

(e) None of the above

Answer

(d)
Both gates are XNORs — XOR bodies with an inversion bubble on the output. The first produces $\overline{A \oplus B}$, and the second XNORs that with C:

$$X = \overline{\overline{A \oplus B} \oplus C}$$

Now use the fact that complementing one input of an XOR complements its output, i.e. $\overline{x} \oplus y = \overline{x \oplus y}$. Applying it to the inner term:

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

so the outer inversion cancels it and $X = A \oplus B \oplus C$.
The other three options are all the *same* function as each other, and all equal $\overline{A \oplus B \oplus C}$ — the complement of the answer. That is worth noticing: (a), (b) and (c) each apply

an odd number of complements to an XOR chain, so they must agree with one another and disagree with (d). Two XNORs in series apply two complements, which cancel. See logic-gates[35].

Q6 — Boolean simplification

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

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

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

(c) $A \oplus B \oplus C$

(d) C

(e) 0 (False)

Answer

(b)
Read the overbars carefully: in the source there are two bars over A, three over B and one over C in the first factor, which parses as $\overline{\overline{A \cdot \overline{B}} \cdot C}$.
Take the inner term first and push the complement in with De Morgan:

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

so the first factor is

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

Now multiply out against the second factor:

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

The middle two products each contain a variable and its complement, so they vanish, leaving

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

which is option (b). See boolean-algebra[36].

[35] courses/csse2010/logic-gates.pdf

[36]

Q7 — Gate circuit for an expression

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

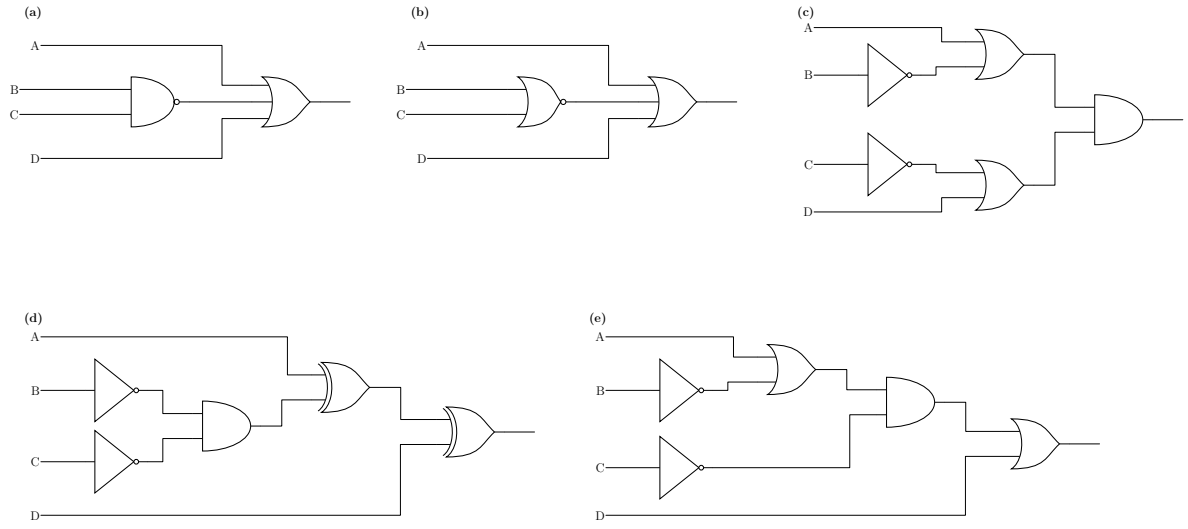


Figure 3: Question 7's five candidate circuits, (a) to (e), each taking A, B, C and D.

(a) — the circuit labelled (a) above

(b) — the circuit labelled (b) above

(c) — the circuit labelled (c) above

(d) — the circuit labelled (d) above

(e) — the circuit labelled (e) above

Answer

(b)
The target has one product term, $\overline{B} \cdot \overline{C}$, ORed with the two bare literals A and D. A NOR gate *is* that product term — $\overline{B + C} = \overline{B} \cdot \overline{C}$ by De Morgan — so the circuit wanted is a 2-input NOR on B and C feeding a 3-input OR along with A and D. That is (b). The others:

- (a) uses a NAND instead of a NOR, giving $A + \overline{B \cdot C} + D = A + \overline{B} + \overline{C} + D$. Same shape, wrong gate.
- (c) gives $(A + \overline{B}) \cdot (\overline{C} + D)$ — a product of sums, not the right function (it is 0 when $A = 1, C = 1, D = 0$, where the target is 1).

courses/csse2010/boolean-algebra.pdf

- (d) gives $(A \oplus \overline{B}\,\overline{C}) \oplus D$; XOR is not OR, so e.g. $A = D = 1$ gives 0 rather than 1.
- (e) gives $(A + \overline{B}) \cdot \overline{C} + D$, which is 0 when $A = 1, C = 1, D = 0$.

See logic-gates[37].

Q8 — Expression to truth table

Which of the following truth tables is equivalent to the function $X = A \cdot \Big(B \cdot \overline{C} + \overline{A \cdot (B + \overline{C})}\Big)$?

(a)

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

(b)

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

(c)

[37] courses/csse2010/logic-gates.pdf

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

(d)

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

(e) None of the above

Answer

(d)

The whole expression is ANDed with A, so $X = 0$ whenever $A = 0$ — that kills the top half of every table and is true of all four candidates, so it does not discriminate. Set $A = 1$ and simplify:

$$A \cdot (B + \overline{C}) = B + \overline{C} \qquad \text{so} \qquad \overline{A \cdot (B + \overline{C})} = \overline{B + \overline{C}} = \overline{B} \cdot C$$

and therefore

$$X = B\overline{C} + \overline{B}C = B \oplus C$$

So the function is $X = A \cdot (B \oplus C)$: 1 exactly when $A = 1$ and B and C differ. That is rows 101 and 110, and nothing else:

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

which is table (d). Table (c) differs only in the 101 row and (b) only in the 111 row, so read those two rows carefully. See boolean-algebra[38].

Q9 — Timing diagram to expression

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

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

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

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

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

(e) None of the above

Answer

(e) — every one of (a) to (d) *is* equivalent, so none of them is the odd one out.
First read the diagram off. A toggles every slot, B every two, C every four, so the nine intervals shown are the eight input combinations in binary counting order (with A as the least significant bit) plus a repeat of the first. X is high over slots 0, 1, 4 and 7:

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

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

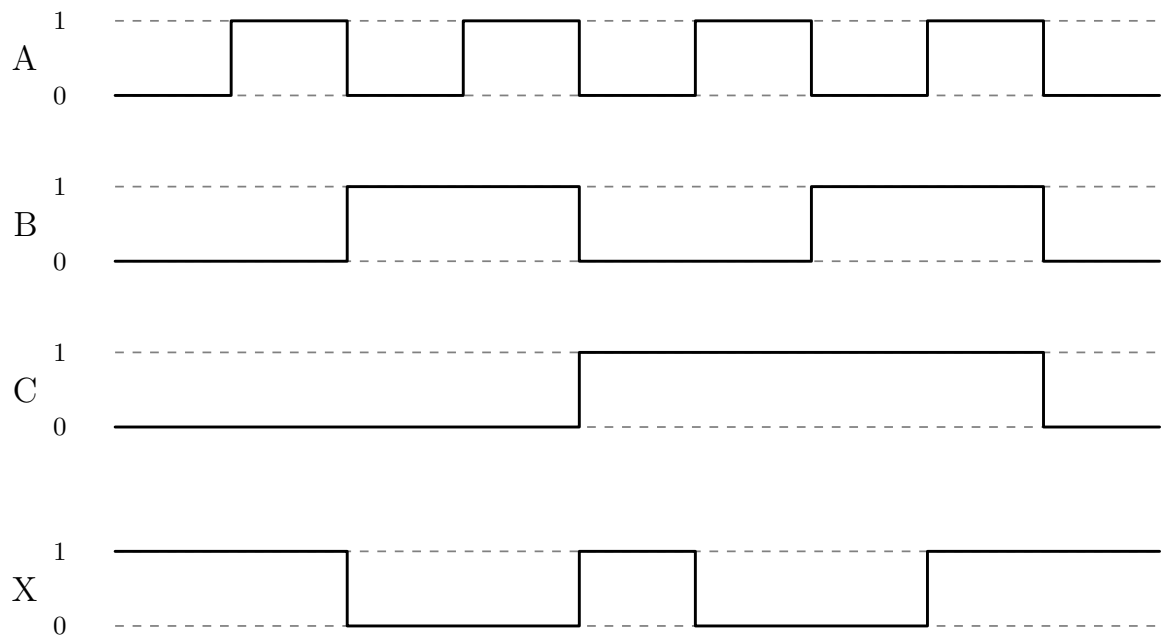


Figure 4: Question 9's timing diagram: A, B and C step through all eight input combinations (A is the fastest-changing signal) and X is the resulting output.

0	0	1	1
1	0	1	0
0	1	1	0
1	1	1	1

So $X = 1$ on exactly $\{\overline{A}\,\overline{B}\,\overline{C},\ A\overline{B}\,\overline{C},\ \overline{A}\,\overline{B}C,\ ABC\}$ — writing the combinations as ABC bit strings, $\{000, 100, 001, 111\}$.
Now test each option on that set:

- (a) $\overline{B}(\overline{A} + \overline{C})$ covers $\overline{A}\,\overline{B}$ (that is 000 and 001) and $\overline{B}\,\overline{C}$ (000 and 100); adding ABC (111) gives $\{000, 001, 100, 111\}$.
- (b) $\overline{B}\,\overline{C}$ covers $\{000, 100\}$. $\overline{A} \oplus B$ is 1 when $A = B = 0$ or $A = B = 1$, so $C(\overline{A} \oplus B)$ covers $\{001, 111\}$. Union is the same set.
- (c) $\overline{A}\,\overline{B}$ covers $\{000, 001\}$. $\overline{B} \oplus C$ is 1 when $B = 0, C = 0$ or $B = 1, C = 1$, so $A(\overline{B} \oplus C)$ covers $\{100, 111\}$.
- (d) $\overline{A}\,\overline{B}$ gives $\{000, 001\}$, $\overline{C}\,\overline{B}$ gives $\{000, 100\}$, and $B \oplus \overline{C}$ is 1 when $B = 0, C = 0$ or $B = 1, C = 1$, so $A(B \oplus \overline{C})$ gives $\{100, 111\}$.

All four match, so the answer is (e). See timing-diagrams[39].

Q10 — Sum-of-products form

Which of the following is a sum-of-products expression equivalent to $(\overline{A} + B) \cdot \overline{(C + B \cdot D)}$?

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

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

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

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

(e) None of the above

Answer

(a)
Push the complement in with De Morgan, twice:

$$\overline{C + B \cdot D} = \overline{C} \cdot \overline{BD} = \overline{C} \cdot (\overline{B} + \overline{D})$$

so the whole expression is

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

Multiply out the two sums:

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

since $B\overline{B} = 0$. Distributing the $\overline{C}$ gives

$$\overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{C}\,\overline{D} + B\,\overline{C}\,\overline{D}$$

which is (a). Option (d) is also *algebraically* equal to the original, but it is not a sum of products — $\overline{BD}$ is a complemented product, not a literal — and the question asks specifically for sum-of-products form. Option (b) is (a) with the last term missing, so it drops the case $A = 1, B = 1, C = 0, D = 0$. See boolean-algebra[40].

Q11 — 3-input NOR from 2-input NANDs

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

[39] courses/csse2010/timing-diagrams.pdf

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

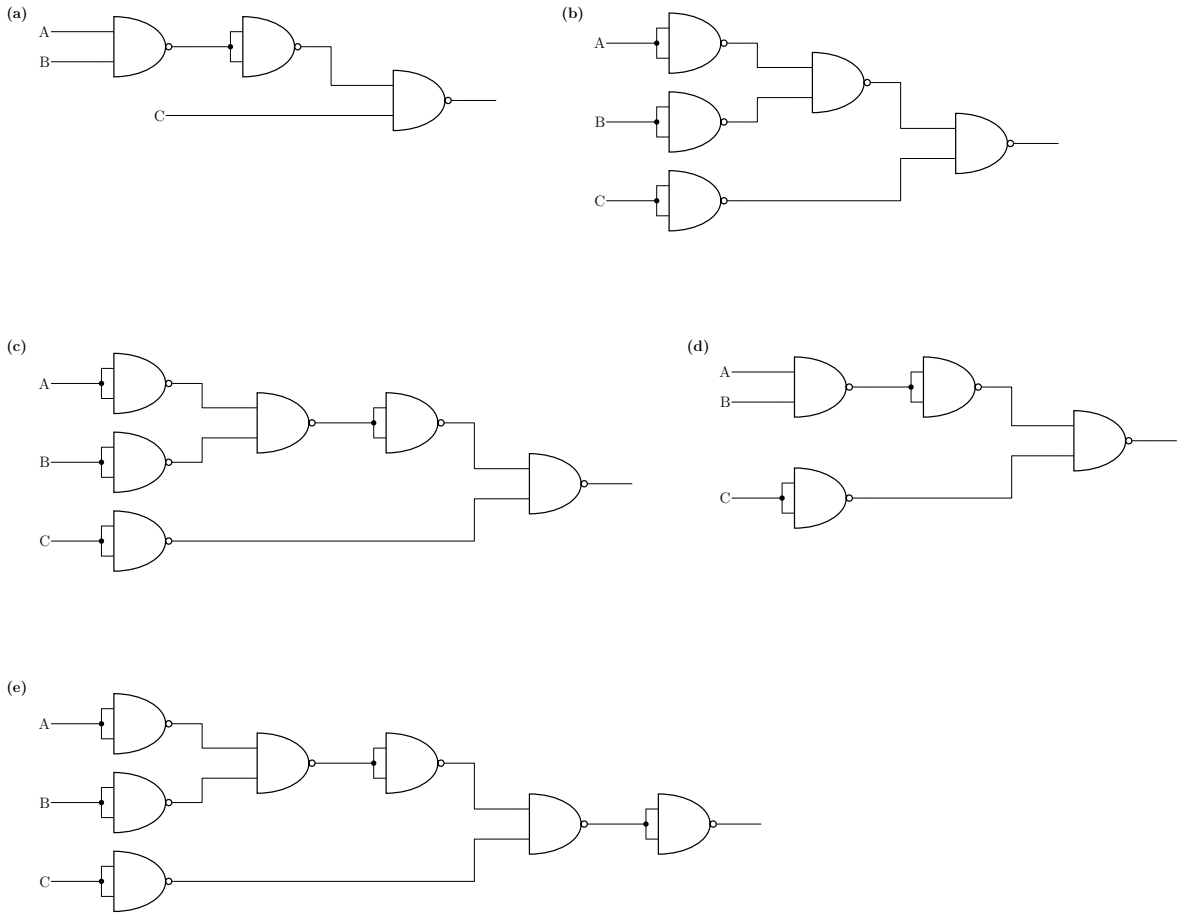


Figure 5: Question 11's five candidate NAND-only networks, (a) to (e). A NAND with its two inputs tied together — shown by the junction dot on its input lead — acts as an inverter.

(a) — the circuit labelled (a) above

(b) — the circuit labelled (b) above

(c) — the circuit labelled (c) above

(d) — the circuit labelled (d) above

(e) — the circuit labelled (e) above

Answer

(e)
Start from De Morgan: $\overline{A+B+C} = \overline{A} \cdot \overline{B} \cdot \overline{C}$. So the network must invert all three inputs, AND them together, and a 2-input NAND only gives the AND *complemented*, so each AND needs a following inverter. A NAND with both inputs tied is an inverter, which is what the junction dots on the input leads mean.
Trace each option:

- **(a)** NAND(A, B), inverted, gives $A \cdot B$; NANDed with C that is $\overline{A \cdot B \cdot C}$ — a 3-input NAND, not a NOR.
- **(b)** Inverts A, B and C, then NAND$(\overline{A}, \overline{B}) = A + B$, then NANDs that with $\overline{C}$: $\overline{(A+B) \cdot \overline{C}}$. Wrong.
- **(c)** Same as (b) but with an inverter after the first NAND, so the second NAND sees $\overline{A}\,\overline{B}$ and $\overline{C}$ and gives $\overline{\overline{A}\,\overline{B}\,\overline{C}} = A + B + C$. That is a 3-input OR — one inversion short.
- **(d)** $A \cdot B$ NANDed with $\overline{C}$, i.e. $\overline{AB\overline{C}}$. Wrong.
- **(e)** Exactly (c) with a seventh NAND wired as an inverter on the output, turning $A + B + C$ into $\overline{A+B+C}$.

Counting gates is a quick filter: any correct answer needs at least three inverters (one per input), one NAND, one inverter to undo it, one more NAND and a final inverter — seven gates. Only (e) has seven. See logic-gates[41].

Q12 — D flip-flop timing

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) — the A waveform labelled (a) above

[41] courses/csse2010/logic-gates.pdf

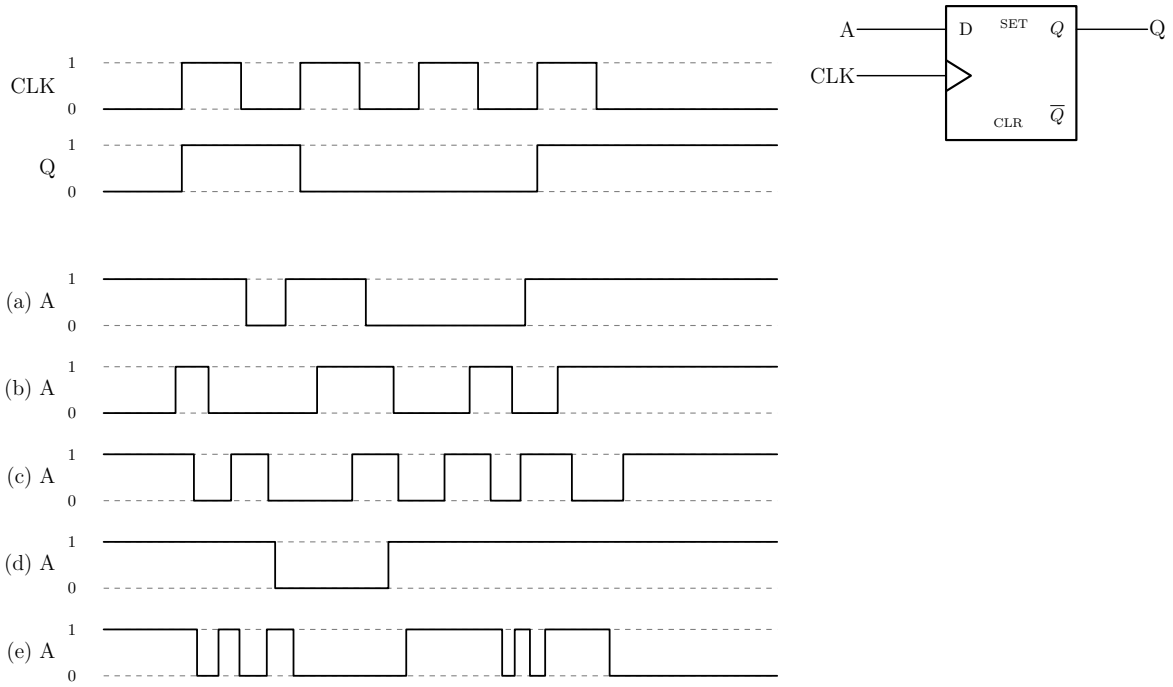


Figure 6: Question 12's given CLK and Q waveforms and the D flip-flop they belong to, with the five candidate A waveforms (a) to (e) below.

(b) — the A waveform labelled (b) above

(c) — the A waveform labelled (c) above

(d) — the A waveform labelled (d) above

(e) — the A waveform labelled (e) above

Answer

(c)
The flip-flop is a plain D type with a triangle and no bubble on the clock input, so it samples D on each *rising* edge and holds in between. There are four rising edges in the CLK waveform. Reading Q against them:

Rising edge	Q before	Q after	So A must have been
1st	0	1	1
2nd	1	0	0
3rd	0	0	0
4th	0	1	1

So the required A is 1, 0, 0, 1 at the four rising edges, and completely unconstrained everywhere else — which is why several candidates look plausible until you check all four.

- (a) is 1 at edge 1 but has risen again before edge 2, so Q would stay 1.
- (b) is right for the first three edges but has fallen again before edge 4, so Q would stay 0.
- (c) is 1, 0, 0, 1 at the four edges.
- (d) is 1, 0, then back to 1 before edge 3, which would set Q high a clock early.
- (e) is 1, 0, then 1 before edge 3 — same failure as (d).

See flip-flops-and-latches[42] and timing-diagrams[43].

Q13 — Two's complement addition

What is the result of adding the 7-bit two's complement numbers 1011001 and 0101111?

(a) 0000000

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

[43] courses/csse2010/timing-diagrams.pdf

(b) 0001000

(c) 1110110

(d) 1110111

(e) None of the above

Answer

(b)
Add the bit patterns as unsigned binary and throw away the carry out of the top bit, which is what two's complement addition always is:

```
   1111111   <- carries
   1011001
 + 0101111
 ---------
  10001000
```

Discarding the ninth-column carry leaves 0001000.
Check in decimal: 1011001 has its sign bit set, so it is $89 - 128 = -39$; $0101111 = 47$. Then $-39 + 47 = 8 = 0001000$. The signs of the operands differ, so overflow is impossible and the result is trustworthy. See binary-number-representations[44].

Q14 — Overflow detection

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

(a) 01000 + 01000

(b) 11111 + 00001

(c) 11000 + 01111

(d) 11101 + 11010

(e) None of the above

Answer

(a)
5-bit two's complement covers −16 to +15 (see Q3), and overflow means the true sum

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

falls outside that. Adding operands of *opposite* sign can never overflow, so only (a) and (d) are even candidates.

- (a) $8+8 = 16$, which is one past the top of the range. The bit sum is $01000+01000 = 10000$, which reads as -16 — two positives producing a negative, the classic overflow signature.
- (b) $-1 + 1 = 0$; signs differ, no overflow.
- (c) $-8 + 15 = 7$; signs differ, no overflow.
- (d) $-3+(-6) = -9$, comfortably inside the range. The bit sum is $11101+11010 = 110111$; dropping the carry gives $10111 = -9$. Two negatives giving a negative, so no overflow.

See binary-number-representations[45].

Q15 — Overflow and result sign

Consider the binary addition operation 110111 + 101001. Which of the following statements is true if the numbers are interpreted as two's complement?

(a) The result is negative and no overflow occurs

(b) The result should be negative but overflow occurs

(c) The result is positive and no overflow occurs

(d) The result is positive but overflow occurs

(e) The result is zero

Answer

(a)
These are 6-bit numbers, so the representable range is -32 to $+31$.

$$110111_2 = 55 \;\rightarrow\; 55 - 64 = -9 \qquad 101001_2 = 41 \;\rightarrow\; 41 - 64 = -23$$

and $-9 + (-23) = -32$, which is exactly the most negative representable value — inside the range, so no overflow.
The bit-level addition agrees:

```
111111   <- carries
110111
```

45 courses/csse2010/binary-number-representations.pdf

```
 + 101001
 --------
  1100000
```

Dropping the carry out leaves $100000 = -32$. Both operands are negative and the result is negative, so no overflow; equivalently, the carry *into* the sign column and the carry *out* of it are both 1, and overflow is their XOR, which is 0. See binary-number-representations[46].

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 $X = S_1 \oplus S_0 \oplus (G \cdot S_1)$?

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

(b) A= 0 B= 1 C= G D= $\overline{G}$

(c) A= 1 B= 0 C= $\overline{G}$ D= G

(d) A= 0 B= 1 C= $\overline{G}$ D= G

(e) None of the above

Answer

(d)
A multiplexer routes input i to the output when the select lines hold i, so each data input is just the value the function takes for that one combination of S_1S_0. Evaluate $X = S_1 \oplus S_0 \oplus (G \cdot S_1)$ four times:

S_1S_0	selected input	X
00	A	$0 \oplus 0 \oplus (G \cdot 0) = 0$
01	B	$0 \oplus 1 \oplus (G \cdot 0) = 1$
10	C	$1 \oplus 0 \oplus (G \cdot 1) = 1 \oplus G = \overline{G}$
11	D	$1 \oplus 1 \oplus (G \cdot 1) = 0 \oplus G = G$

So A = 0, B = 1, C = $\overline{G}$, D = G, which is (d). The $G \cdot S_1$ term is what makes only the bottom half depend on G, and the $S_1 \oplus S_0$ part is what flips $\overline{G}$ into G between lines 2 and 3. See combinational-logic-blocks[47].

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

[47]

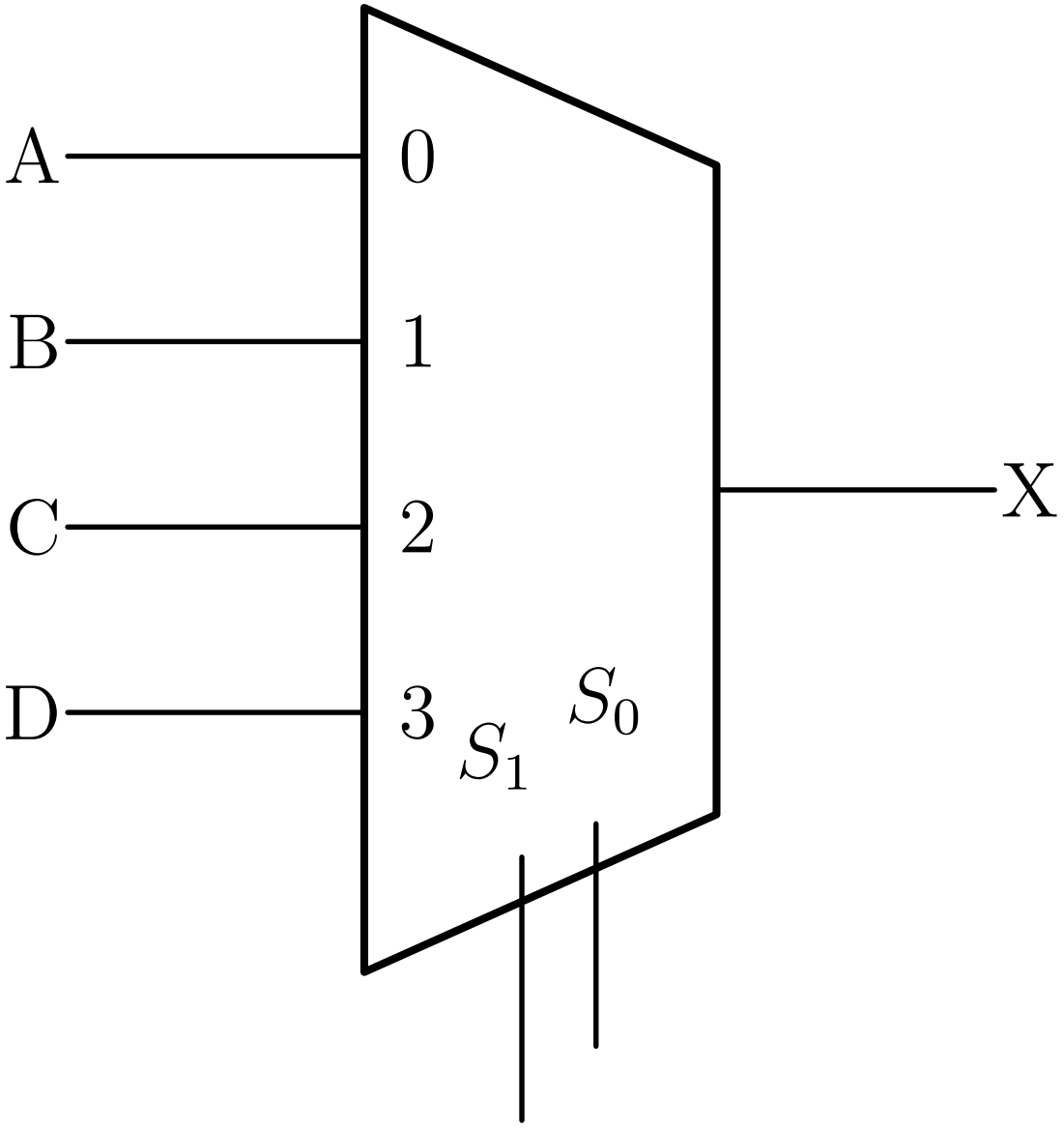


Figure 7: Question 16's 4-to-1 multiplexer, with data inputs A, B, C and D on lines 0 to 3 and select inputs S1 and S0.

Q17 — Multiplexer output function

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

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

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

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

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

(e) None of the above

Answer

(d)

The select lines are $S_2 S_1 S_0 = A\,B\,C$, so the selected line number is $4A + 2B + C$ and A picks which half of the input list is in play.

With $A = 0$ the mux can only reach lines 0–3, whose values are 1, 1, 1, 0:

B	C	line	X
0	0	0	1
0	1	1	1
1	0	2	1
1	1	3	0

which is 0 only when both B and C are 1 — that is $X = \overline{B \cdot C}$.

With $A = 1$ it reaches lines 4–7, whose values are 0, 1, 1, 1:

B	C	line	X
0	0	4	0
0	1	5	1
1	0	6	1
1	1	7	1

which is 1 whenever either is 1 — that is $X = B + C$.

So (d). This is the standard trick of using an n-select mux as a lookup table for an n-variable function. See combinational-logic-blocks[48].

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

courses/csse2010/combinational-logic-blocks.pdf

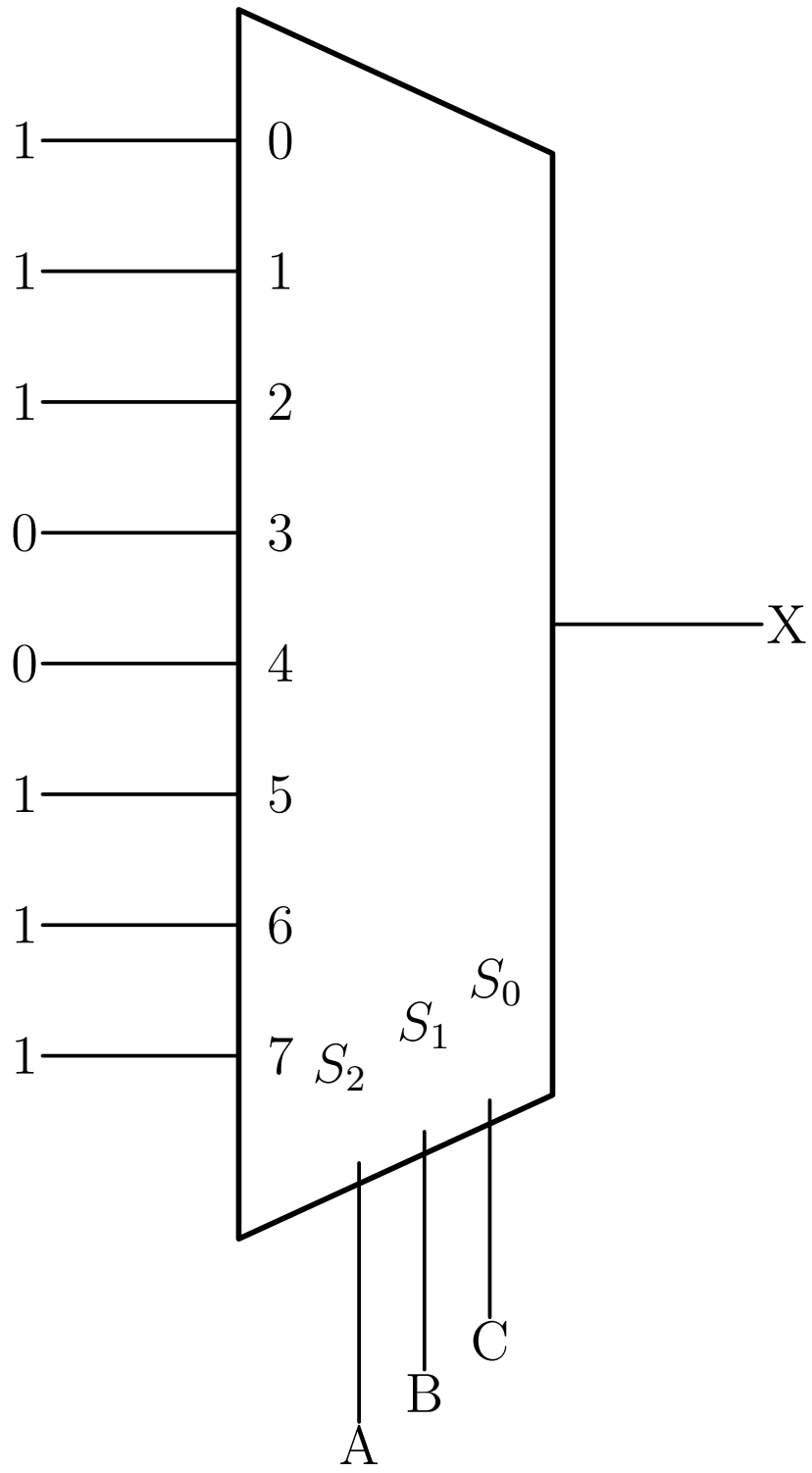


Figure 8: Question 17's 8-to-1 multiplexer, with the eight data lines hard-wired to constants and the select inputs S2, S1 and S0 driven by A, B and C.

Q18 — Multiplexer-fed flip-flops

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

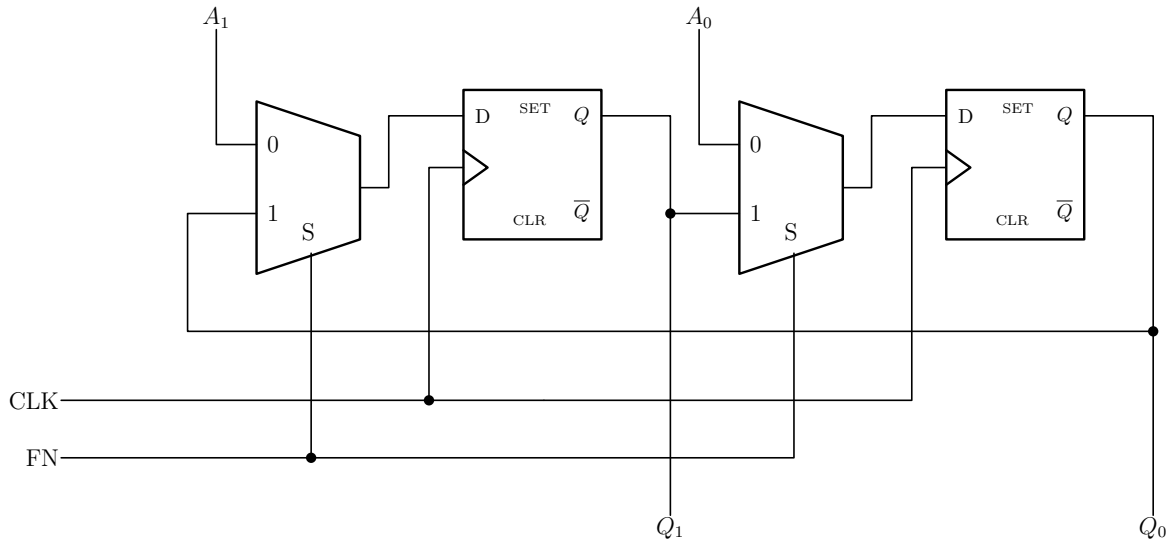


Figure 9: Question 18's circuit: two D flip-flops, each fed by a 2-to-1 multiplexer selected by FN — line 0 loads A1/A0, line 1 cross-couples the two Q outputs.

(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

Answer

(c)
Read the multiplexers first. Each flip-flop's D input comes from its own mux, and both muxes are selected by FN:

- FN = 0 selects line 0 on both, so $D_1 = A_1$ and $D_0 = A_0$ — a parallel **load**.
- FN = 1 selects line 1, and the wiring is crossed: D_1 comes from Q_0 and D_0 comes from Q_1 — a **swap**.

Now clock it. Note that the two load cycles wipe out whatever the flip-flops held

beforehand, which is why (e) is not needed.

Edge	FN	D_1D_0	Q_1Q_0 after
start	—	—	unknown
1	0	$A_1A_0 = 01$	0 1
2	0	$A_1A_0 = 01$	0 1
3	1	$Q_0Q_1 = 10$	1 0
4	1	$Q_0Q_1 = 01$	0 1
5	1	$Q_0Q_1 = 10$	1 0

After five edges, $Q_1 = 1$ and $Q_0 = 0$. The swap has odd parity — three swaps is the same as one — which is the whole point of the question. See flip-flops-and-latches[49] and combinational-logic-blocks[50].

Q19 — Synchronous counter trace

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

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

(b) Q_1Q_0: 00 → 10 → 00 → …

(c) Q_1Q_0: 00 → 11 → 01 → 11 → …

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

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

Answer

(e)
Read the two next-state equations off the schematic. The OR gate is drawn mirrored — its output points left, back to the D input of the left flip-flop — and its two inputs are the Q_0 rail running along the top and the $\overline{Q}$ of the left flip-flop. The NAND takes the same Q_0 rail and Q_1, and drives the right flip-flop's D. So

$$D_1 = Q_0 + \overline{Q_1} \qquad D_0 = \overline{Q_1 \cdot Q_0}$$

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

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

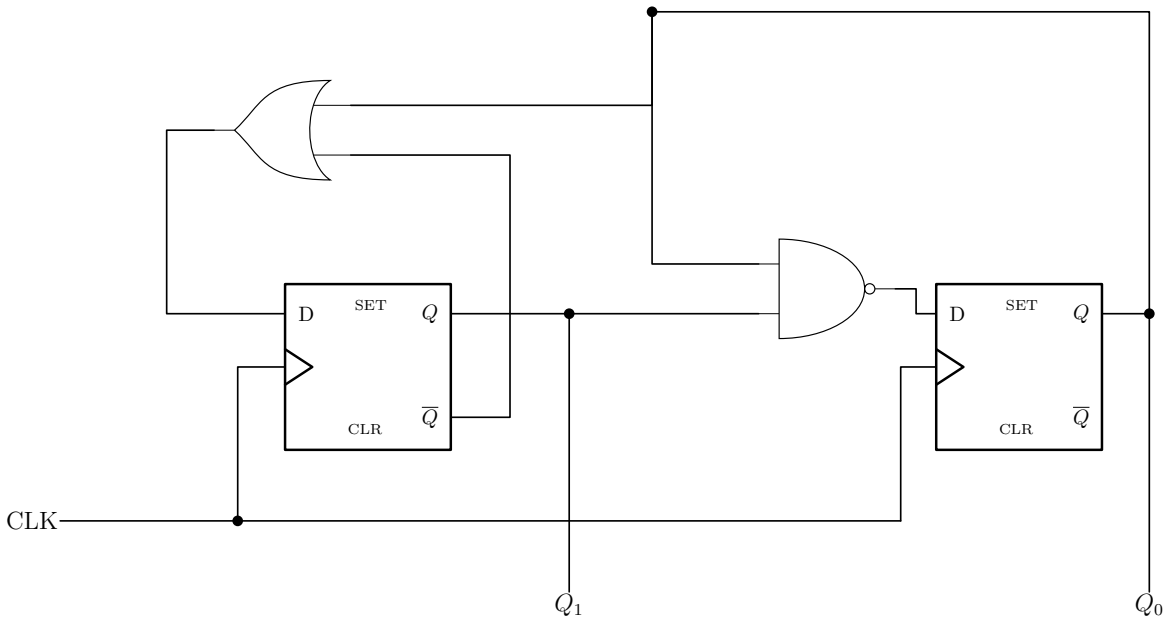


Figure 10: Question 19's synchronous counter. The OR gate is drawn mirrored, with its output pointing left back to D1; it takes Q0 and Q1-bar. The NAND takes Q0 and Q1 and drives D0.

Both flip-flops share the clock, so evaluate both from the *current* state each cycle:

Q_1Q_0	$D_1 = Q_0 + \overline{Q_1}$	$D_0 = \overline{Q_1 Q_0}$	next
00	$0 + 1 = 1$	$\overline{0} = 1$	11
11	$1 + 0 = 1$	$\overline{1} = 0$	10
10	$0 + 0 = 0$	$\overline{0} = 1$	01
01	$1 + 1 = 1$	$\overline{0} = 1$	11

So from 00 the machine runs $00 \to 11 \to 10 \to 01 \to 11 \to 10 \to 01 \to \ldots$, settling into a three-state loop and never returning to 00. That is (e); (d) has the same first four states but wrongly loops back to 00. See counters[51].

[51] courses/csse2010/counters.pdf

Q20 — Flip-flops for a count sequence

What is the minimum number of flip-flops needed to implement a state machine which counts through the sequence 00000→00001→00011→11000→00001→… ?

(a) 2

(b) 3

(c) 4

(d) 5

(e) 32

Answer

(a)
The five-bit patterns are the machine's *outputs*, not necessarily its state encoding. Count the distinct patterns: 00000, 00001, 00011, 11000 — four of them, since the 00001 at the end of the sequence is the same state it was the first time round (the machine loops 00001 → 00011 → 11000 → 00001 forever after leaving 00000).
Four states need $\lceil \log_2 4 \rceil = 2$ flip-flops, with combinational output logic decoding the two state bits into the five output bits. So (a).
Option (d) is the trap: five flip-flops is what you would need if you insisted on storing the output pattern directly in the register, which is the natural but not the minimal design.
See state-machines[52].

Q21 — State diagram to state table

Consider the state diagram below.

Which of the following is an equivalent state table?

(a)

Current State	Input (A)	Next State	Outputs (X,Y)
S1	0	S1	01
S1	1	S2	01
S2	0	S2	00
S2	1	S4	00
S4	0	S8	10

52 courses/csse2010/state-machines.pdf

Current State	Input (A)	Next State	Outputs (X,Y)
S4	1	S8	10
S8	0	S16	11
S8	1	S1	11
S16	0	S1	01
S16	1	S2	01

(b)

Current State	Input (A)	Next State	Outputs (X,Y)
S1	0	S1	01
S1	1	S2	01
S2	0	S2	00
S2	1	S4	00
S4	0	S8	10
S4	1	S8	10
S8	0	S1	11
S8	1	S16	11
S16	0	S1	01
S16	1	S2	01

(c)

Current State	Next State (A=0)	Next State (A=1)	Outputs (X,Y)
S1	S1	S2	01
S2	S2	S4	00
S4	S4	S8	10
S8	S16	S1	11
S16	S1	S2	01

(d)

Current State	Next State (A=0)	Next State (A=1)	Outputs (X,Y)
S1	S1	S2	01
S2	S2	S4	00
S4	S8	S8	01
S8	S16	S1	11
S16	S1	S2	01

(e) None of the above

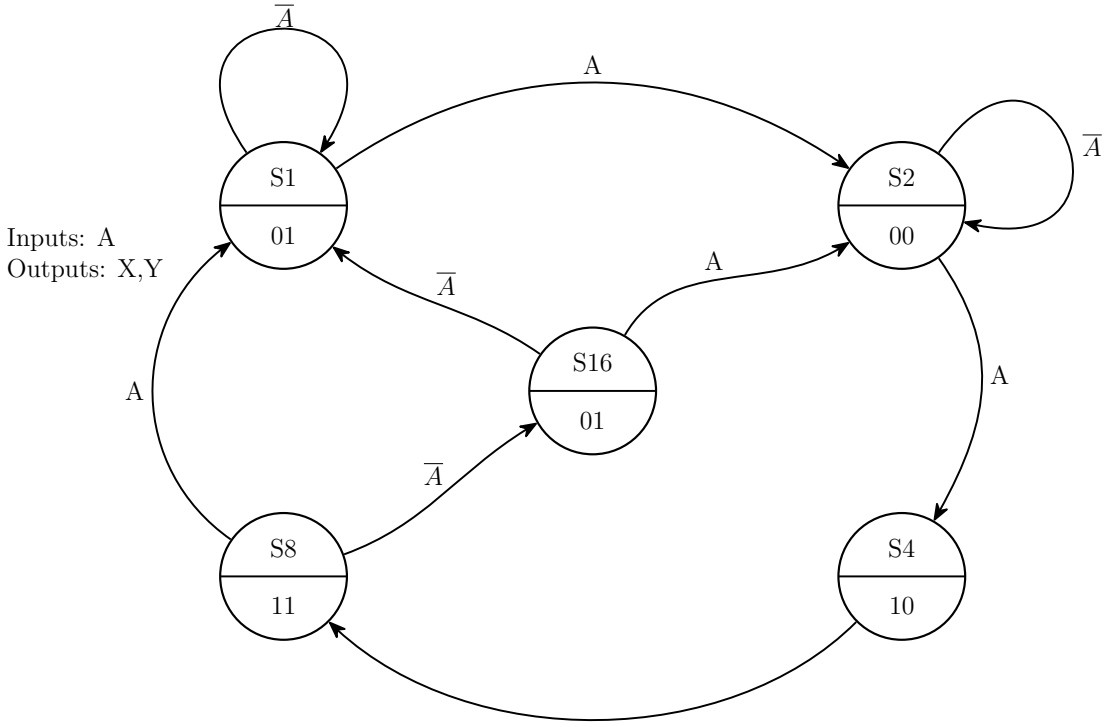


Figure 11: Question 21's Moore state diagram. Each bubble shows the state name above and its X,Y outputs below; the S4 to S8 transition carries no label, so it is taken for either value of A.

Answer

(a)
Read the ten transitions off the diagram. The outputs sit inside the bubbles, so this is a Moore machine and the output column depends only on the current state:

From	On	To
S1	$\overline{A}$	S1 (self-loop)
S1	A	S2
S2	$\overline{A}$	S2 (self-loop)
S2	A	S4
S4	*(unlabelled)*	S8
S8	$\overline{A}$	S16
S8	A	S1
S16	$\overline{A}$	S1
S16	A	S2

The S4 → S8 arrow carries no label, which means it is taken regardless of A — S4 has no other outgoing arrow, so it must be unconditional. Table (a) matches all of this line for line.
The other three each break in exactly one place:

- (b) swaps S8's two destinations: it sends S8 to S1 on A = 0 and to S16 on A = 1, the reverse of the diagram.
- (c) gives S4 a self-loop on A = 0 instead of the unconditional move to S8.
- (d) has the right transitions but prints S4's outputs as 01 when the bubble says 10.

See state-machines[53].

Q22 — State encoding width

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

(a) 1

(b) 2

(c) 3

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

(d) 4

(e) 5

Answer

(c)
The diagram in Q21 has five states — S1, S2, S4, S8 and S16. A register of n flip-flops distinguishes 2^n states, so $n = \lceil \log_2 5 \rceil = 3$: two flip-flops would only cover four states, and three cover eight, leaving three encodings unused.
The state *names* are a distraction — they look like one-hot codes (1, 2, 4, 8, 16), and one-hot encoding really would need five flip-flops, which is option (e). But the question asks for the minimum, and nothing forces one-hot. See state-machines[54].

Q23 — Tracing a state sequence

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 S1 and A goes through the sequence 110011 (i.e. two 1's followed by two 0's followed by two 1's)?

(a) S1→S1→S2→S4→S8→S16→S2

(b) S1→S2→S4→S8→S1→S2→S4

(c) S1→S2→S2→S2→S4→S8→S16

(d) S1→S2→S4→S8→S16→S2→S4

(e) None of the above

Answer

(d)
Six input values means six transitions, so the answer is a list of seven states starting at S1. Walk the table from Q21:

Step	A	From	To
1	1	S1	S2
2	1	S2	S4
3	0	S4	S8 *(unconditional)*
4	0	S8	S16

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

5	1	S16	S2
6	1	S2	S4

giving S1→S2→S4→S8→S16→S2→S4, which is (d).
The step that separates the options is step 3: S4 moves on to S8 whatever A is, so the first 0 does *not* hold the machine still. Option (b) makes the mistake of sending S8 to S1 on A = 0, and (c) parks the machine in S2. See state-machines[55].

Q24 — ALU function select

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 result in the ALU output being $A \oplus B$? (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 = 1, ENB = 1, INVB = 0, $F_2 = 0$, $F_1 = 0$, $F_0 = 0$, Right Shift In = 0, Carry in = 0

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

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

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

(e) None of the above

Answer

(e)
First read the slice. Two internal rails carry the conditioned inputs:

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

and the X rail is also what leaves as Right Shift Out. Eight things feed the multiplexer, which is selected by $F_2F_1F_0$ as a 3-bit number:

$F_2F_1F_0$	mux line	Output
000	0	$\overline{X+Y}$
001	1	$X \cdot Y$
010	2	$X+Y$

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

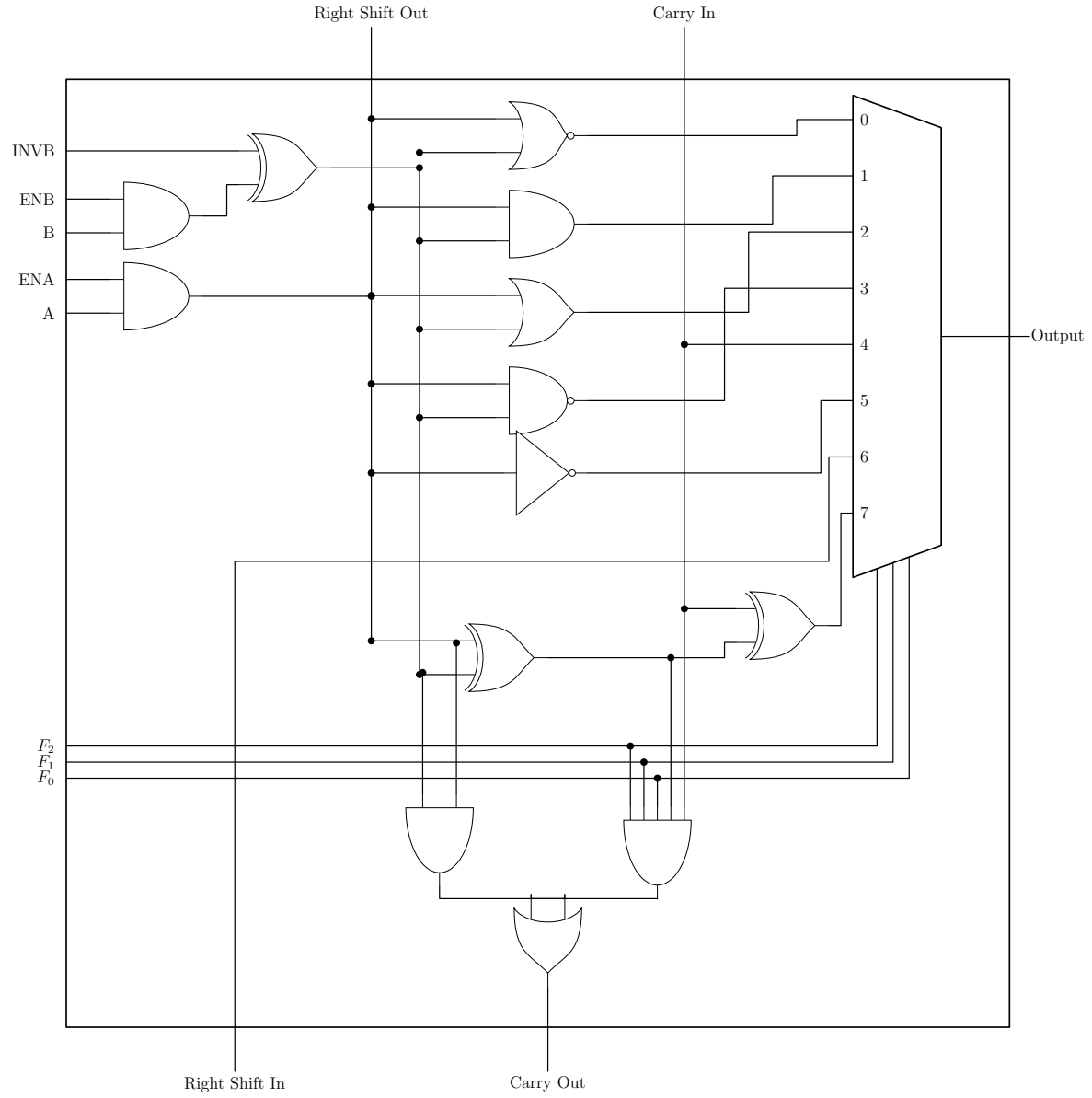


Figure 12: Question 24's ALU bit slice. ENA and ENB/INVB condition A and B into two internal rails; five logic blocks, the carry in, the right shift in and the full adder sum feed the eight inputs of a multiplexer selected by F2 F1 F0. The conditioned A rail is also the Right Shift Out.

011	3	$\overline{X \cdot Y}$
100	4	Carry In
101	5	$\overline{X}$
110	6	Right Shift In
111	7	$X \oplus Y \oplus C_{in}$ (the adder sum)

and the carry out is $X \cdot Y + F_2F_1F_0 \cdot (X \oplus Y) \cdot C_{in}$ — an ordinary full-adder carry, with the propagate term switched off unless the adder is actually selected.
Now the options. All four set ENA = ENB = 1, so $X = A$ throughout.

- (a) INVB = 0 gives $Y = B$, and $F = 000$ selects NOR: $\overline{A + B}$.
- (b) INVB = 1 gives $Y = \overline{B}$, and $F = 000$ selects NOR: $\overline{A + \overline{B}} = \overline{A} \cdot B$.
- (c) $Y = \overline{B}$ and $F = 010$ selects OR: $A + \overline{B}$.
- (d) $Y = B$ and $F = 111$ selects the adder. The per-bit output is $A_i \oplus B_i \oplus C_i$, which *looks* like XOR, and for bit 0 (where Carry in = 0 is given) it is. But $F = 111$ also switches the carry chain on, so bit 0 sends $A_0 \cdot B_0$ into bit 1, bit 1 sends its carry into bit 2, and so on. What the 8-bit ALU computes is $A + B$, not $A \oplus B$; the two agree only when no pair of bits is 1 together.

No mux line produces XOR without the carry chain, and there is no way to select the sum while stopping the carries, so the answer is (e).
Check this one. The intended answer may well be (d) if the examiner meant only the per-slice sum expression $A \oplus B \oplus C_{in}$ — but in an 8-bit ALU the carries into slices 1–7 are not zero, so (d) really computes addition. The parenthetical note that Carry in refers only to the least significant bit is the hint that the ripple is meant to be taken seriously. See arithmetic-logic-unit[56].

Q25 — ALU constant generation

For the ALU in Q24, which of the following control input settings will **NOT** result in the ALU output being the constant value –1 (negative one)?

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

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

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

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

(e) ENA = 0, ENB = 1, INVB = 1, $F_2 = 1$, $F_1 = 0$, $F_0 = 1$, Right Shift In = 0, Carry in = 1

[56] courses/csse2010/arithmetic-logic-unit.pdf

Answer

(d)

In two's complement, -1 is all ones, so the question is really: which setting fails to force every bit of the output to 1 *regardless of the data in A and B*? Use the rails and the mux table from Q24: $X = A \cdot \text{ENA}$ and $Y = (B \cdot \text{ENB}) \oplus \text{INVB}$.

- (a) $\overline{\text{ENA}} = \text{ENB} = \text{INVB} = 0$, so $X = 0$ and $Y = 0$. $F = 000$ selects NOR: $\overline{0 + 0} = 1$ on every bit. gives -1.
- (b) $X = 0$; ENB $= 0$ with INVB $= 1$ gives $Y = 0 \oplus 1 = 1$. $F = 010$ selects OR: $0 + 1 = 1$.
- (c) $\overline{\text{ENA}} = 1$ so $X = A$; ENB $= 0$, INVB $= 0$ so $Y = 0$. $F = 011$ selects NAND: $\overline{A \cdot 0} = \overline{0} = 1$ whatever A is.
- (d) ENA $= \overline{\text{ENB}} = 1$ and INVB $= 1$, so $X = A$ and $Y = \overline{B}$. $F = 011$ selects NAND: $\overline{A \cdot \overline{B}}$, which is 0 whenever $A = 1$ and $B = 0$. Data-dependent, so not the constant -1.
- (e) ENA $= 0$ so $X = 0$; $F = 101$ selects $\overline{X} = \overline{0} = 1$.

Only (d) leaves the output depending on the data, so it is the one that will not give -1. The general lesson is the point of the question: ENA and ENB let the ALU force a rail to a constant 0, and INVB (or the NOT/NAND/NOR lines) turns that into a constant 1 across the whole word. That is how an ALU with no dedicated constant hardware still manages to produce -1, and by extension $B - 1$ and $B - A$. See arithmetic-logic-unit[57].

[57] courses/csse2010/arithmetic-logic-unit.pdf