ALLAMA IQBAL OPEN UNIVERSITY



AIOU LIBRARY
aioulibrary.com

Name:

Registration No.:

Course Code: 3006

Program: BS

Semester: Autumn 2026

ALLAMA IQBAL OPEN UNIVERSITY, ISLAMABAD

(Department of Computer Science)

Course: Digital Logic Design (3006) • Level: BS

Semester: Autumn 2026

Assignment No. 2 (Units 5–9)

Total Marks: 100 • Pass Marks: 50

SOLVED ASSIGNMENT GUIDE

(For Study & Reference)

Note: This guide is prepared for study and reference purposes. Students are advised to read the relevant units of the course book, understand the method shown in each answer, and then write their assignments in their own words. Copying any solved guide word for word is against the university's rules, which are printed on the question booklet itself.

Transparency Note (course code): The question booklet prints the course as "Digital Logic Design (3006)". The source question file supplied for this booklet is named CS3006.doc in the assignment listing. This guide follows the code exactly as printed in the booklet — (3006) — in its title block, footer and file name.

Question 1: Constructing a JK Flip-Flop from a D Flip-Flop, a Multiplexer and an Inverter (20 Marks)

Question as printed: Construct a JK flip-flop using a D flip-flop, a two-to-one-line multiplexer, and an inverter. (20)

Introduction

Flip-flop conversion — building one type of flip-flop out of another by adding a small amount of combinational logic at its input — is a standard exercise in sequential design, and it is also genuinely useful: integrated circuits and programmable logic devices often provide only D flip-flops, so any other behaviour must be synthesised from them. This question asks for the most capable of the common flip-flops, the JK, to be built from a D flip-flop, a single two-to-one-line multiplexer and a single inverter. The method is to write down what the D input must be in order for the D flip-flop to imitate the JK characteristic, and then to recognise that the required expression is precisely the output equation of a 2-to-1 multiplexer whose select line is the present output Q and whose data inputs are J and the complement of K.

What the D Input Must Be

The JK flip-flop is defined by its characteristic equation Q(t + 1) = J Q′(t) + K′ Q(t): with J = 0, K = 0 the output holds its value; with J = 0, K = 1 it resets to 0; with J = 1, K = 0 it sets to 1; and with J = K = 1 it toggles. A D flip-flop obeys Q(t + 1) = D, so the conversion problem is solved the moment we arrange that D = J Q′ + K′ Q at every clock edge, where Q without a time argument means the present state Q(t). Now compare this with the equation of a two-to-one-line

multiplexer, $Y = S' I_0 + S I_1$, where S is the select input and I_0, I_1 are the two data inputs. The two expressions have exactly the same shape. Matching them term by term gives S = Q (the present output itself is the select line), I_0 = J (the input selected when Q = 0) and I_1 = K′ (the input selected when Q = 1). The complement K′ is produced by the single inverter the question allows, connected in the K line. No other components are needed.

The Construction

Figure 1: JK flip-flop constructed from a D flip-flop, a 2-to-1 multiplexer and an inverter (Question 1)

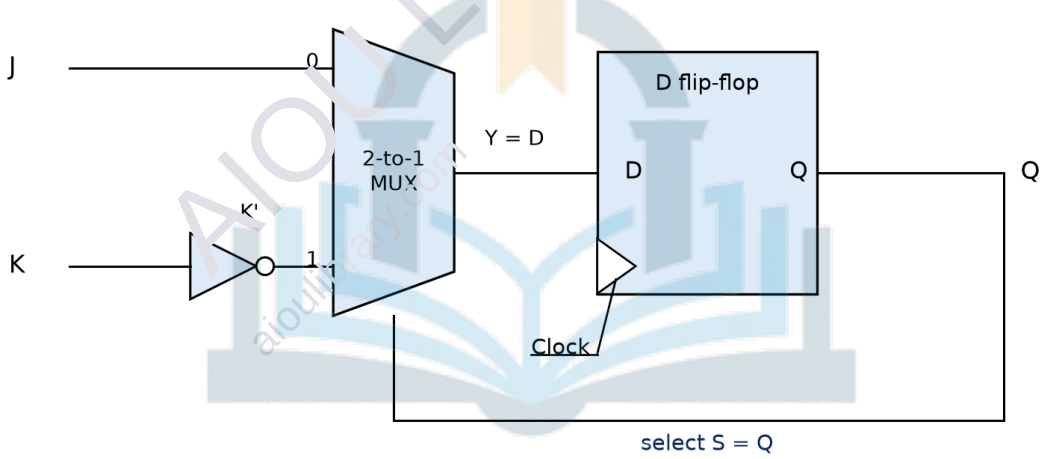


Figure 1 shows the complete circuit. The inverter takes K as its input and produces K′. The multiplexer receives J at its 0-input and K′ at its 1-input; its select line is driven by the flip-flop's own output Q, fed back from the output side of the circuit. The multiplexer output Y is connected to the D input of the D flip-flop, and the flip-flop's Q output is the output of the constructed JK flip-flop. The clock is applied to the D flip-flop in the ordinary way, so the whole circuit is a single-clock, edge-triggered device, exactly as a JK flip-flop should be. Note that the feedback of Q to the select terminal is not a combinational loop in the troublesome sense: Q

changes only at the clock edge, and between edges it is a stable stored value, so the multiplexer output settles to a steady D well before the next edge.

Verification Against the JK Characteristic Table

The construction is verified by checking all eight combinations of J, K and the present state Q. When Q = 0 the multiplexer selects I_0 = J, so D = J: with J = 0 the next state is 0 (hold, since Q was 0), and with J = 1 the next state is 1 regardless of K — covering the set row (J = 1, K = 0) and the toggle row (J = K = 1) for present state 0. When Q = 1 the multiplexer selects I_1 = K′, so D = K′: with K = 0 the next state is 1 (hold, and also the set row for present state 1), and with K = 1 the next state is 0 — covering the reset row (J = 0, K = 1) and the toggle row for present state 1. Every row of the JK characteristic table is reproduced, so the circuit is a true JK flip-flop.

J	K	Present Q	MUX select S = Q	D (= MUX output)	Next Q(t + 1)	JK behaviour
0	0	0	0	J = 0	0	Hold
0	0	1	1	K′ = 1	1	Hold
0	1	0	0	J = 0	0	Reset
0	1	1	1	K′ = 0	0	Reset
1	0	0	0	J = 1	1	Set
1	0	1	1	K′ = 1	1	Set
1	1	0	0	J = 1	1	Toggle
1	1	1	1	K′ = 0	0	Toggle

It is worth seeing why the apparently simpler idea — feeding J and K′ into an ordinary AND–OR network to form D = J Q′ + K′ Q — is not what the question asks for, even though it would work. The question deliberately restricts the parts list to the multiplexer and the inverter, and the elegance of the solution is that the multiplexer's own equation already contains the required structure: a multiplexer with select Q is, by definition, a hardware realisation of the form "if Q is 0 take the first input, if Q is 1 take the second", and the JK characteristic is exactly a statement of that form. This is a first example of a general lesson of the course — that data selectors can implement arbitrary small Boolean functions of their select variables.

One further point connects this construction with the theory of the JK flip-flop itself. The toggle mode, J = K = 1, is what makes the JK valuable — it is the basis of counters — but in a simple level-clocked JK latch it also creates the race-around condition, in which the output toggles repeatedly during a single long clock pulse. The present construction inherits the cure automatically: because the storage element is an edge-triggered D flip-flop, the output can change only once, at the active clock edge, no matter how long the clock pulse lasts, and the multiplexer simply presents the correct next-state value to the D input before that edge arrives. The select feedback from Q is evaluated with the old value of Q throughout the set-up interval, so the value sampled at the edge is exactly the JK next state computed from the state that existed before the edge — which is precisely the behaviour the characteristic table defines.

Conclusion

Connecting the D input of a D flip-flop to the output of a 2-to-1 multiplexer with select line Q, data inputs J and K′ (K′ from the single inverter), gives D = J Q′ + K′ Q, which is the characteristic equation of the JK flip-flop. The eight-row check confirms hold, set, reset and toggle behaviour, so the required JK flip-flop has been constructed using exactly the components named in the question — one D flip-flop, one two-to-one-line multiplexer and one inverter.

Question 2: State Table and State Diagram of the Sequential Circuit of Fig. P5.8 (20 Marks)

Question as printed: Derive the state table and the state diagram of the sequential circuit shown in Fig. P5.8 . Explain the function that the circuit performs (20) [The printed figure shows two T flip-flops, A (left) and B (right), clocked in common. Under each flip-flop is an OR gate whose output drives that flip-flop's T input. The wiring traced from the printed figure is stated in full in the answer below.]

Introduction

The analysis of a clocked sequential circuit follows a fixed routine: read the flip-flop input equations off the diagram, combine them with the characteristic equation of the flip-flop type used, tabulate the next state for every present state, and draw the state diagram. The circuit of Fig. P5.8 contains two T (toggle) flip-flops, labelled A and B, both driven by the same clock, and there are no external inputs and no separate output logic — the state of the two flip-flops is

itself the output. Circuits of this kind are counters and sequence generators, and the final part of the answer identifies exactly which counting sequence this circuit produces.

Reading the Flip-Flop Input Equations from the Figure

Tracing the printed figure connection by connection gives the following wiring. The T input of flip-flop A is driven by an OR gate whose two inputs are A itself (the wire that leaves the A output, runs up and over the top of the diagram, down the left-hand side and along the bottom into the left-hand input of that OR gate) and B (the wire from the B output that runs down the right-hand side and along the bottom into the right-hand input of the same OR gate). Hence $T_a = A + B$. The T input of flip-flop B is driven by a second OR gate whose inputs are A′ (the wire that leaves the complemented output of flip-flop A, runs across and then down the middle of the diagram to the left hand input of that OR gate) and again B (the bottom wire has a junction dot where the B line also feeds this second OR gate). Hence $T_\beta = A' + B$. The complemented output B′ is brought out at the top of flip-flop B but is not connected to anything else in the figure.

A T flip-flop holds its value when T = 0 and complements it when T = 1, so its characteristic equation is $Q(t + 1) = Q \oplus T$. Applying this to each flip-flop gives the next-state equations directly: $A(t + 1) = A \oplus (A + B)$, and $B(t + 1) = B \oplus (A' + B)$. These can be simplified before tabulating. For A: if B = 0, then $A \oplus A = 0$; if B = 1, then $A \oplus 1 = A'$; in both cases the result equals A′B, so $A(t + 1) = A'B$. For B: if B = 1, then $B \oplus 1 = 0$; if B = 0, then $0 \oplus A' = A'$; in both cases the result equals A′B′, so $B(t + 1) = A'B'$. The whole behaviour of the circuit is therefore captured by the two compact equations $A(t + 1) = A'B$ and $B(t + 1) = A'B'$.

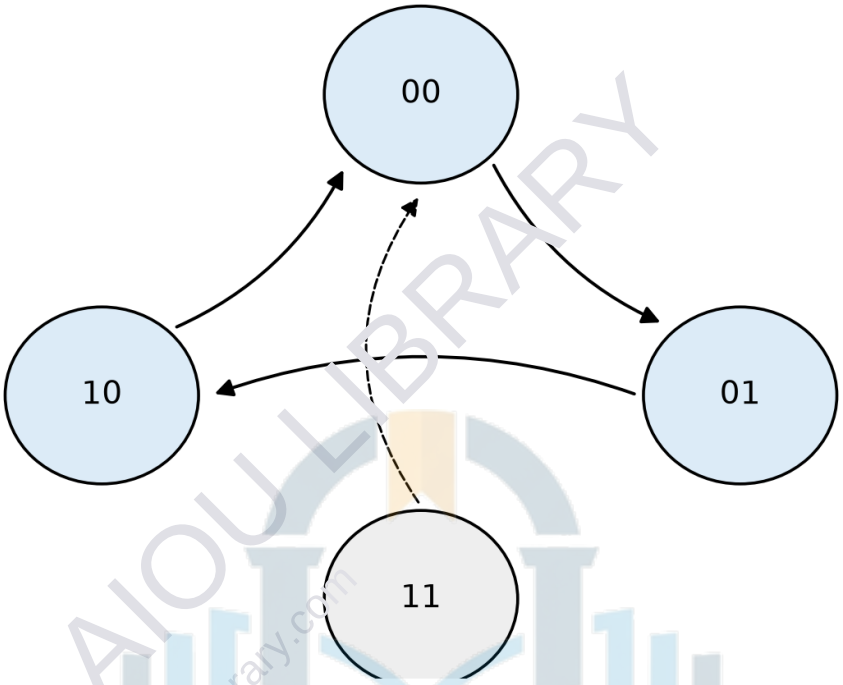
The State Table

Present state A	Present state B	$T_a = A + B$	$T_\beta = A' + B$	Next state A(t + 1)	Next state B(t + 1)
0	0	0	1	0	1
0	1	1	1	1	0
1	0	1	0	0	0
1	1	1	1	0	0

Each row is obtained by evaluating the two T inputs from the present state and toggling each flip-flop whose T input is 1. From 00: $T_a = 0$ so A holds at 0, while $T_\beta = 1$ so B toggles to 1 — next state 01. From 01: both T inputs are 1, so A toggles to 1 and B toggles to 0 — next state 10. From 10: $T_a = 1$ so A toggles to 0, and $T_\beta = 0$ so B holds at 0 — next state 00. From 11: both T inputs are 1, so both flip-flops toggle — next state 00. The simplified equations give the same table: A'B is 1 only for present state 01, and A'B' is 1 only for present state 00, which is exactly what the table shows.

The State Diagram and the Function of the Circuit

State diagram: 00 -> 01 -> 10 -> 00 ... (11 -> 00, dashed)



The state diagram of Figure 2 has one circle for each of the four states of the two flip-flops, with a directed arc from each present state to its next state: 00 → 01, 01 → 10, 10 → 00, and 11 → 00. Three of the states form a closed loop, 00 → 01 → 10 → 00, which repeats forever once entered; the remaining state, 11, is not part of the loop but leads into it in a single clock pulse (11 → 00) and never recurs. The function of the circuit is now clear: it is a modulo-3 counter (a divide-by-3 counter). Starting in state 00, the state advances through the counting sequence 00, 01, 10, 00, 01, 10, …, so the circuit cycles through three distinct states and returns to its starting state after every three clock pulses; a signal taken from the counter completes one full cycle for every three input clock cycles, i.e. the circuit divides the clock frequency by three. The unused state 11 is harmless because the circuit is self-correcting: even if the flip-flops power up in state

11, the very first clock pulse takes the counter to 00 and it thereafter follows the normal count sequence, with no possibility of becoming stuck in an unused loop.

Two remarks complete the explanation. First, the counting order 00, 01, 10 is the ordinary binary counting order with the final combination replaced by a return to zero, which is typical of counters built to a modulus that is not a power of two: a two-flip-flop binary counter has four states (modulus 4), and the feedback through the OR gates in this circuit removes one state from the cycle, reducing the modulus to 3. Second, because the T inputs are formed from both flip-flop outputs, the circuit is a synchronous counter — both flip-flops are clocked together and change simultaneously — not a ripple counter, so all state transitions in the table occur at the same clock edge.

Conclusion

With input equations $T_a = A + B$ and $T_\beta = A' + B$ read from Fig. P5.8, the next-state equations reduce to $A(t + 1) = A'B$ and $B(t + 1) = A'B'$, the state table shows the transitions 00 → 01 → 10 → 00 with 11 → 00, and the state diagram displays the same behaviour graphically. The circuit is a synchronous modulo-3 (divide-by-3) counter, self-correcting from its one unused state.

Question 3: Shifting a Register, and Serial Versus Parallel Transfer (20 Marks)

Question as printed: a) The contents of a four-bit register are initially 1100. The register is shifted six times to the right with the serial input being 110010. What is the content of the register after each shift? (10) b) What is the difference between serial and parallel transfer?

Explain how to convert serial data to parallel and parallel data to serial. What type of register is needed? (10)

Introduction

Shift registers are the link between the two great ways of moving binary information: serially, one bit at a time over a single line, and in parallel, all bits at once over a bundle of lines. Part (a) is a careful bookkeeping exercise on a single four-bit register — it checks that the student knows precisely which end a serial bit enters, which end the displaced bit leaves, and how the contents evolve shift by shift. Part (b) then asks for the general principles behind the exercise: the difference between serial and parallel transfer, and the registers that convert between the two forms.

Part (a): Six Right Shifts with Serial Input 110010

In a right shift, every bit moves one position towards the least significant (right-hand) end; the bit that falls off the right-hand end is lost (or read as the serial output), and the new bit at the most significant (left-hand) end is whatever is presented at the serial input. The serial input stream 110010 is applied in the order written — the first (leftmost) bit, 1, enters at the first shift, then the second bit, 1, at the second shift, and so on. The register starts at 1100. First shift (serial input 1): the bits move right, the last 0 drops off, and 1 enters at the left, giving 1110. Second shift (input 1): 1111. Third shift (input 0): the rightmost 1 drops off and 0 enters, giving 0111. Fourth shift (input 0): 0011. Fifth shift (input 1): 1001. Sixth shift (input 0): 0100. The final content of the register is therefore 0100, which consists of the last four serial bits that entered

(0, 1, 0, 0 in the order of the final four shifts) — a neat illustration of the fact that after four or more shifts none of the original content survives.

Shift number	Serial input bit	Bit shifted out (right end)	Register contents after shift
Initial	—	—	1100
1	1	0	1110
2	1	0	1111
3	0	1	0111
4	0	1	0011
5	1	1	1001
6	0	1	0100

A useful way to check the table is to follow one individual bit. The original most significant 1 of 1100 moves one place right at each shift: after one shift it stands second from the left (1110), after two shifts third (1111), after three shifts it is the rightmost bit (0111), and at the fourth shift it leaves the register — which is why the fourth content, 0011, contains only bits that arrived from the serial input and the second original bit. The bookkeeping is consistent.

Part (b): Serial and Parallel Transfer Compared

In parallel transfer, all the bits of a binary word are transferred simultaneously, each over its own line, in a single clock pulse. An n-bit word needs n lines (plus control and ground), but the

whole word arrives in one step, so parallel transfer is fast. Inside a computer — between registers, and between the processor and memory over the data bus — transfer is normally parallel for exactly this reason. In serial transfer, the bits of the word are sent one after another, in time order, over a single line, so an n-bit word needs n clock pulses and only one data line. Serial transfer is slower for a given clock rate, but it needs far less wiring, which is why it is preferred over longer distances and in interfaces such as USB and the traditional serial ports: one line (or one pair) is cheaper and more reliable to run between separate pieces of equipment than a wide parallel cable, and the saving in connectors and cabling outweighs the loss of speed.

Feature	Serial transfer	Parallel transfer
Data lines for an n-bit word	One	n (one per bit)
Clock pulses for an n-bit word	n	One
Speed for a given clock	Slower (one bit per pulse)	Faster (whole word per pulse)
Wiring / connector cost	Low	High
Typical use	Longer distances, external interfaces	Inside the computer, buses, register-to-register
Register type used	Shift register	Register with parallel load and parallel outputs

Converting Between the Two Forms

Serial-to-parallel conversion is the job of the shift register itself, used as a serial-in, parallel-out (SIPO) register. The serial bits are applied to the serial input and the register is clocked once per bit; after n clock pulses for an n-bit word, each flip-flop of the register holds one bit of the word, in order, and the complete word is simply read off the n flip-flop outputs simultaneously — that simultaneous reading is a parallel output. Part (a) demonstrated the mechanism without naming it: after the last four shifts, the register held 0100, the last four serial bits, available all at once on its four outputs. In practice a control signal marks the moment when the word is complete, and the parallel value is then transferred onward into an ordinary parallel register if it must be held while the next word is being shifted in.

Parallel-to-serial conversion needs a shift register with parallel-load capability — a parallel-in, serial-out (PISO) register, sometimes called a parallel-load shift register. The word to be sent is first loaded into all the flip-flops simultaneously through their parallel inputs, using a load (or shift/load) control signal in place of one shifting step; the control is then switched to shift mode and the register is clocked n times, so the bits leave one by one from the serial output at the end flip-flop, least (or most) significant bit first according to the shift direction. After n pulses the register is empty of the original word and ready to be loaded with the next. A universal (bidirectional, multifunction) shift register such as the well-known 74194 combines all of these facilities — parallel load, shift right, shift left, serial inputs at both ends and parallel outputs — and can therefore serve both conversion directions in a single device, which is why such registers are the standard answer to the question “what type of register is needed?”: a shift

register with parallel outputs for serial-to-parallel conversion, and a parallel-load shift register for parallel-to-serial conversion.

Conclusion

Six right shifts with serial input 110010 carry the register 1100 through 1110, 1111, 0111, 0011 and 1001 to a final content of 0100. Serial transfer moves a word one bit at a time over a single line in n clock pulses, while parallel transfer moves the whole word at once over n lines in one pulse; converting serial data to parallel form uses a serial-in, parallel-out shift register, and converting parallel data to serial form uses a parallel-load (parallel-in, serial-out) shift register.

Question 4: Design of a Serial 2's Complementer (20 Marks)

Question as printed: Design a serial 2's complementer with a shift register and a flip-flop. The binary number is shifted out from one side and its 2's complement is shifted into the other side of the shift register. (20)

Introduction

The 2's complement of a binary number was met in Assignment 1 as a parallel operation: invert all the bits and add 1. This question asks for the serial version of the same job, done with the modest hardware of one shift register and one extra flip-flop. The number sits in the shift register; bit by bit it is shifted out from one end, transformed on its way round, and shifted back into the other end, so that when the process finishes the register contains the 2's complement of the number it started with. The design is a small but complete example of a serial data-

processing unit: a datapath (the shift register) plus a tiny one-bit controller (the flip-flop) that remembers the one fact the transformation depends on.

The Serial Rule for 2's Complement

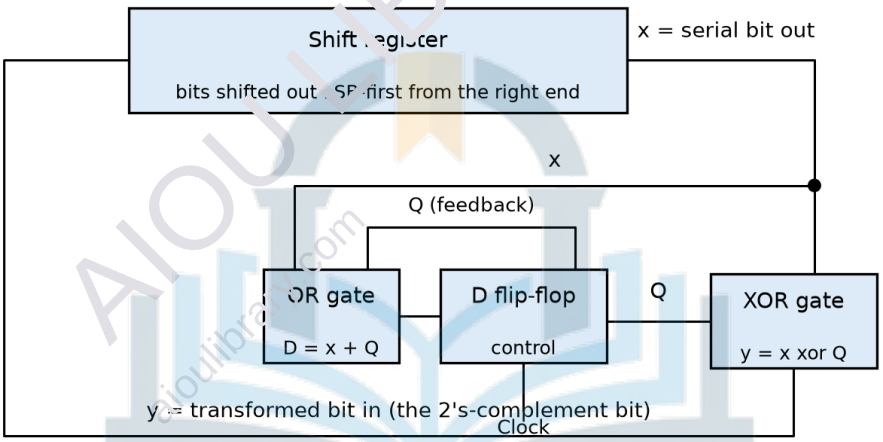
The whole design rests on a well-known property of the 2's complement when bits are processed from the least significant end. Scan the number from the LSB upwards: copy every bit unchanged up to and including the first 1 that is met, and complement every bit after that first 1. If the number contains no 1 at all (it is zero), every bit is copied and the result is zero, which is correct, since the 2's complement of 0 is 0. The rule follows from the meaning of "invert and add 1": adding 1 to the inverted number ripples a carry through the lowest zeros of the original number, turning the inverted bits there back into the original bits, and the carry dies exactly at the first 1, leaving all higher bits inverted. A serial circuit therefore needs to know only one thing about the bits it has already processed: whether the first 1 has been seen yet. One flip-flop is exactly enough to remember that one fact.

The Circuit

Let x be the bit currently leaving the right-hand (least significant) end of the shift register, and let Q be the output of the control flip-flop, initially cleared to 0, meaning "the first 1 has not yet been seen". The transformed bit y that is shifted into the left-hand end of the register is produced by an exclusive-OR gate: $y = x \oplus Q$. While Q = 0, y = x, so bits are copied, as the rule requires before the first 1. The flip-flop must change to 1 as soon as a 1 has been processed, and stay at 1 for the rest of the operation; its input is therefore D = x + Q, formed by an OR gate taking x and the fed-back Q. Once x = 1 occurs, D becomes 1 and Q is 1 from the next bit

onwards; the OR feedback then holds Q at 1 regardless of later bits. With Q = 1, the XOR gate gives y = x', so all subsequent bits are complemented, again exactly as the rule requires. Figure 2 shows the arrangement: the shift register in the middle, the serial path running out of the right end, through the XOR gate, and back into the left end, and the control flip-flop with its OR gate underneath.

Figure 2: Serial 2's complementer - shift register, XOR gate, OR gate and one D flip-flop (Question 4)



Present Q	Serial bit in x	Output bit y = x ⊕ Q	Next Q (D = x + Q)	Meaning
0	0	0	0	Still waiting for the first 1; copy the bit
0	1	1	1	This is the first 1; copy it, and switch to

				complementing
1	0	1	1	First 1 already seen; complement the bit
1	1	0	1	First 1 already seen; complement the bit

A Worked Example: Complementing 0110

Take the four-bit number 0110 (decimal 6), whose 2's complement is 1010. The control flip-flop starts at Q = 0 and the register is shifted right four times. Clock 1: the bit shifted out is the LSB, x = 0; y = 0 ⊕ 0 = 0 enters the left end, so the register becomes 0011, and Q stays 0. Clock 2: x = 1; y = 1 ⊕ 0 = 1 enters, giving 1001, and D = 1 + 0 = 1, so Q becomes 1. Clock 3: x = 1; now y = 1 ⊕ 1 = 0 enters, giving 0100, and Q remains 1. Clock 4: x = 0; y = 0 ⊕ 1 = 1 enters, giving 1010, and Q remains 1. After four clock pulses — one per bit — the register holds 1010, the 2's complement of the original 0110, and every original bit has been replaced in place, just as the question describes: the number was shifted out from one side and its complement shifted into the other side of the same register.

Clock pulse	Bit out x	Q (before pulse)	Bit in y = x ⊕ Q	Register after

				shift
Start	—	0	—	0110
1	0	0	0	0011
2	1	0	1	1001
3	1	1	0	0100
4	0	1	1	1010

Practical details complete the design. Four clock pulses are needed for a four-bit register, and in general n pulses for n bits, so a small counter or the system's control unit must count the pulses and stop (or flag completion) at the right moment. The control flip-flop must be cleared before each new number is processed; if it were left at 1 from a previous operation, every bit of the new number would be complemented from the very first bit and the result would be the 1's complement instead — the classic fault of this circuit, and a good illustration of why sequential circuits need a defined initial state. The circuit works for a register of any length without alteration, because the transformation is purely local in time: it never needs to look ahead to bits that have not yet emerged.

Conclusion

A shift register, one D flip-flop, one OR gate and one XOR gate make a complete serial 2's complementer. The flip-flop records whether the first 1 has been passed; the XOR gate copies bits while Q = 0 and complements them once Q = 1; the OR gate, D = x + Q, sets the flip-flop at

the first 1 and holds it thereafter. The worked example shows 0110 becoming 1010 in four clock pulses, with each transformed bit entering the register at the end opposite to the one the original bits leave from.

Question 5: Memory Cycle Timing Waveforms for Write and Read Operations (20 Marks)

Question as printed: Show the memory cycle timing waveforms for the write and read operations. Assume a CPU clock of 150 MHz and a memory cycle time of 20 ns. (20)

Transparency Note: The booklet names no specific set of control signals for these waveforms. The diagrams below therefore use the standard textbook signal set for a CPU–memory interface — Address, Chip Select / Memory Request (CS/MREQ, active low), Write (WR, active low) for the write cycle, Read / Output Enable (RD/OE, active low) for the read cycle, and the bidirectional Data bus — and every timing quantity is derived only from the two numbers the question supplies (150 MHz and 20 ns). The waveforms are schematic: they show the correct order and alignment of events within one 20 ns cycle, not analogue rise and fall times.

Introduction

A timing waveform diagram is the contract between a processor and its memory. It shows, on a common time axis, when the address becomes valid, when the control signals are asserted, and during which window the data must be present and stable for a transfer to succeed. This question fixes the two numbers that determine the whole geometry of the diagram: the CPU clock frequency, 150 MHz, and the memory cycle time, 20 ns. From these we first compute the clock period and the number of clock cycles spanned by one memory cycle, and then draw the

write cycle, in which data flows from the CPU into the memory, and the read cycle, in which data flows from the memory into the CPU.

The Timing Budget

The clock period is the reciprocal of the frequency: T = 1 / f = 1 / (150 × 10^6 Hz) = 6.67 ns (approximately). One memory cycle of 20 ns therefore spans 20 / 6.67 = 3 clock periods exactly (3 × 6.67 ns = 20 ns, using the unrounded period 1 / 150 MHz = 6.6 ns). In processor terminology, the memory cycle occupies three CPU clock cycles: the access is launched in the first clock period, and the CPU must in effect wait through the remaining periods until the memory has finished — equivalent to two wait states beyond the initial clock — because the memory cannot respond within a single 6.67 ns clock tick. This mismatch, a memory three times slower than the processor clock, is the normal situation in real systems and is the reason wait-state generation exists.

Quantity	Value	How obtained
CPU clock frequency	150 MHz	Given in the question
Clock period T	6.67 ns	T = 1 / f = 1 / (150 × 10^6)
Memory cycle time	20 ns	Given in the question
Clock periods per memory cycle	3	20 ns / 6.67 ns = 3
Wait states inserted	2	3 clock periods span the cycle; the first launches the access

The Write Cycle

Figure 3(a): Memory WRITE cycle - cycle time 20 ns, CPU clock 150 MHz (period 6.67 ns)

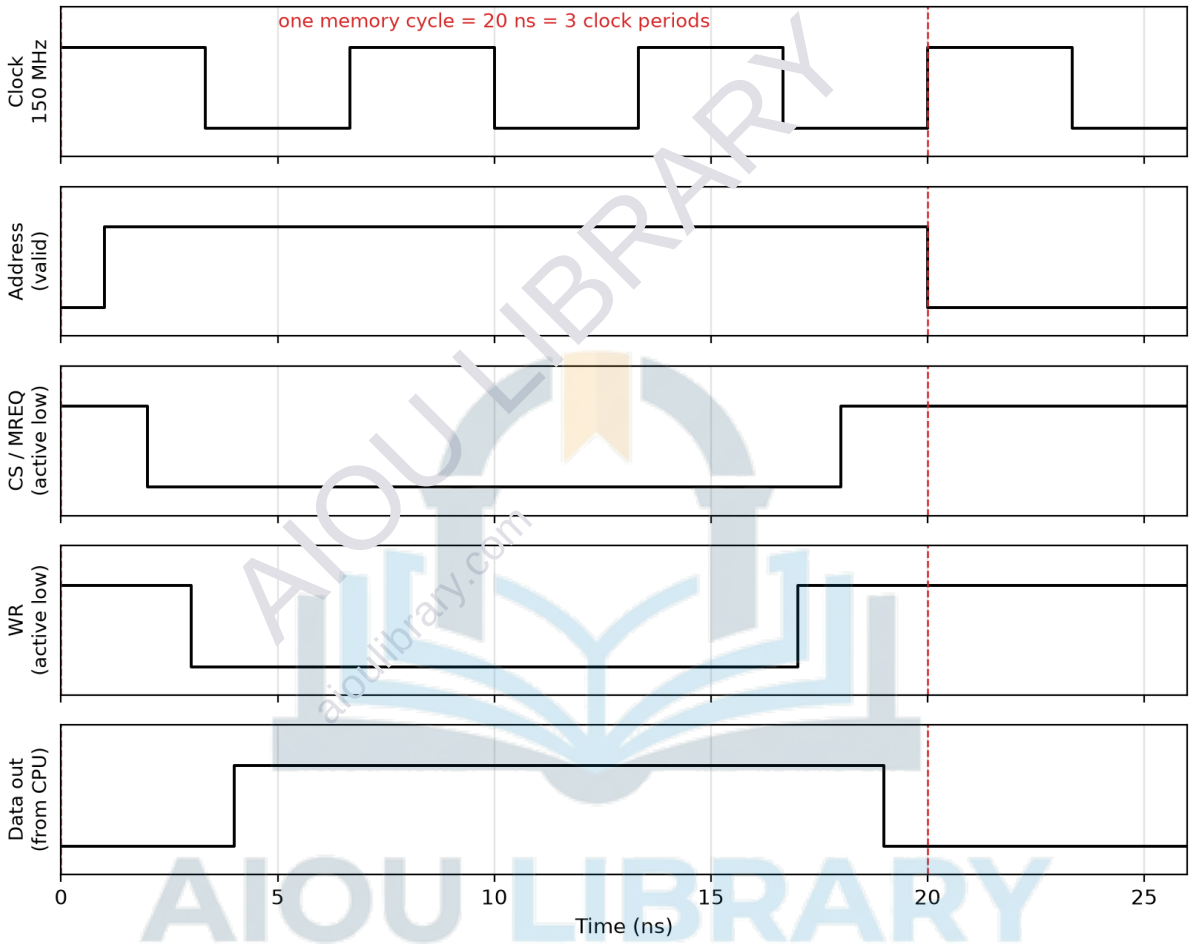


Figure 3(a) shows one write cycle on a time axis of nanoseconds, with the red dashed lines marking the start and end of the 20 ns memory cycle and the CPU clock drawn above for reference, ticking every 6.67 ns. The sequence of events is as follows. (1) The CPU places the address of the location to be written on the address bus; it becomes valid shortly after the start of the cycle (shown at about 1 ns) and is held for the whole cycle, because the memory's address decoding must remain stable while the write takes place. (2) The CPU asserts the chip-

select / memory-request signal (active low, shown falling at about 2 ns), which selects the memory device and tells it that a memory cycle, as opposed to an I/O cycle, is in progress. (3) The write signal WR is asserted (falls at about 3 ns); only when WR is low does the memory accept data. (4) The CPU places the data to be stored on the data bus; the data is valid from about 4 ns and is held until about 19 ns — crucially, it must be valid before WR rises again, since it is the rising edge of WR (at about 17 ns in the diagram) that strobes the data into the addressed cell. (5) CS/MREQ is released (about 18 ns) and the cycle closes at 20 ns, after which the address may change for the next cycle. The essential requirements the waveform expresses are that the address precedes the write strobe, the data is set up before the strobe ends, and everything is held stable until the strobe has finished its work.

AIOU LIBRARY
aioulibrary.com

The Read Cycle

Figure 3(b): Memory READ cycle - cycle time 20 ns, CPU clock 150 MHz (period 6.67 ns)

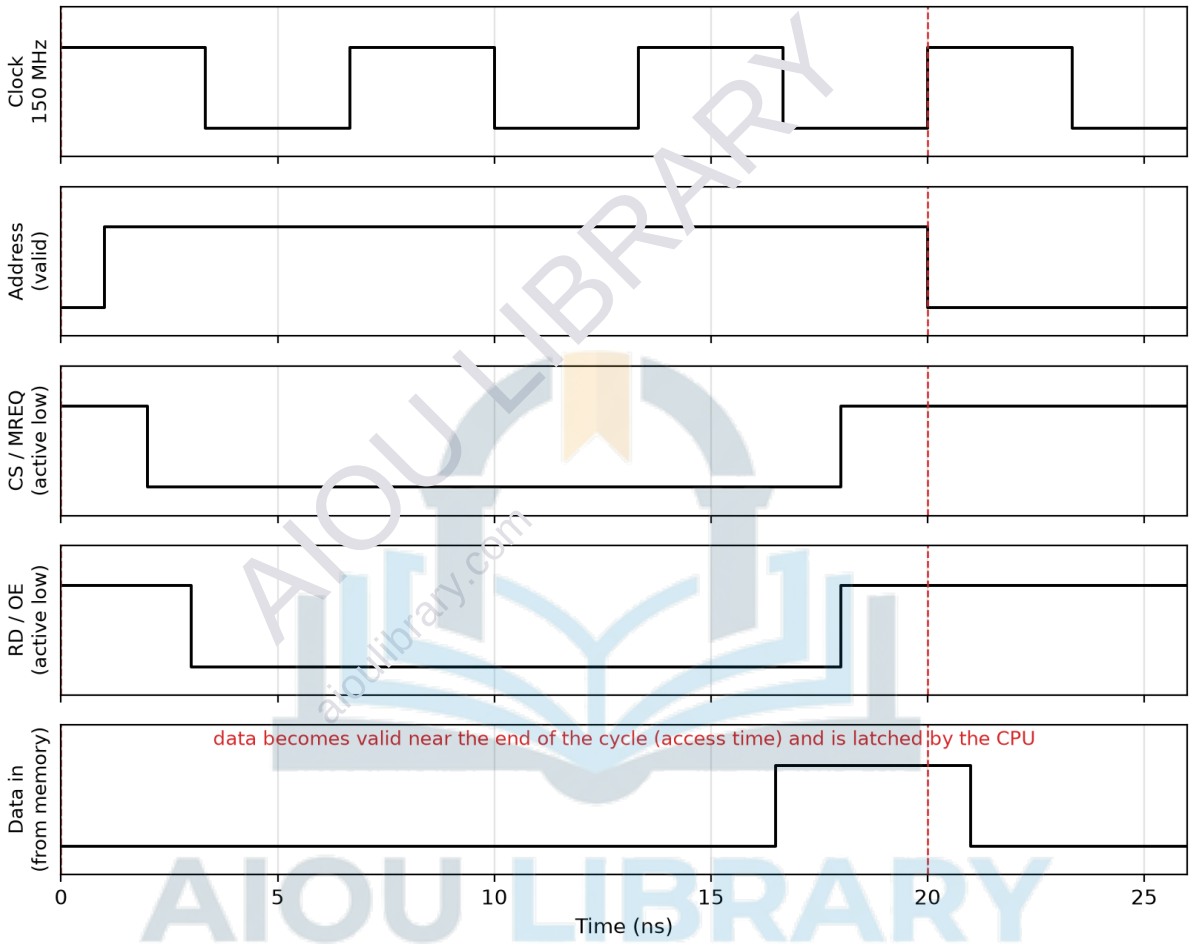


Figure 3(b) shows the read cycle over the same 20 ns window. (1) The CPU again issues the address first; it is valid from about 1 ns for the whole cycle. (2) CS/MREQ is asserted at about 2 ns to select the memory. (3) The read signal RD (equivalently, the memory's output enable, OE) is asserted at about 3 ns, instructing the selected memory to drive the data bus with the contents of the addressed location. (4) Unlike the write cycle, the data does not appear at once: the memory needs its access time to decode the address, sense the stored bits and drive them

onto the bus, so the data bus is shown invalid (low in the schematic) until late in the cycle, becoming valid at about 16.5 ns. (5) The CPU samples (latches) the data at the end of the cycle, at 20 ns, just before CS and RD are released; the interval between the address becoming valid and the data being ready is the access time, and it is this interval that the 20 ns cycle time must accommodate. The contrast between the two diagrams is the direction and timing of the data: in a write, the CPU is the source and the data is early and long-held; in a read, the memory is the source and the data is late, valid only near the end of the cycle.

Reading the two figures together also shows why the memory cycle time, rather than the clock period, governs the transfer rate. The clock ticks three times during each cycle, but no new address can be issued until the current cycle completes, so the fastest possible sequence of memory operations is one per 20 ns, i.e. 50 million accesses per second, even though the CPU is clocked at 150 MHz. If a faster memory with a 13.3 ns cycle time were fitted, the same diagrams would compress to two clock periods and one wait state; with a 6.67 ns memory, no wait states at all would be needed. The waveform method adapts unchanged — only the arithmetic of the timing budget changes.

Conclusion

With a 150 MHz clock the period is 6.67 ns, so the given 20 ns memory cycle spans exactly three clock periods, equivalent to two wait states. The write waveform shows address valid first, then chip-select, then the write strobe, with CPU data set up before the strobe ends and held until it does; the read waveform shows the same opening sequence, but with the data supplied by the memory and becoming valid only near the end of the cycle, when the CPU latches it. Together

the two diagrams specify completely when every signal in a memory transfer must change and how long it must remain stable.

AIOU LIBRARY
aioulibrary.com
AIOU LIBRARY