

SKRYPTY DLA SZKÓŁ WYŻSZYCH

POLITECHNIKA ŁÓDZKA

ZYGMUNT LESZCZYŃSKI

TEORIA UKŁADÓW LOGICZNYCH

ŁÓDŹ 1990

NAKŁADEM POLITECHNIKI ŁÓDZKIEJ



Skrypt jest przeznaczony dla studentów Wydziału Elektrycznego
kierunku elektronika



1472/W

Recenzent: doc. dr hab. Andrzej Materka

Redaktor: Małgorzata Horowska

Redaktor techniczny: Ewa Guzek

Korekta: Anna Waldhauser

WYDAWNICTWO POLITECHNIKI ŁÓDZKIEJ

93-005 Łódź, ul. Wólczańska 219

Nakład 300+46 egz. Ark. wyd. 17,7. Ark. druk. 19,0. Papier offset. kl. IV 71 g. 61X86

Druk ukończono w styczniu 1990 r. Zamówienie 181/87. Cena zł 1770,-

Wykonano w Zakładzie Poligraficznym PŁ, 93-005 Łódź, ul. Wólczańska 223

D-289/90



SPIS TREŚCI

1. WSTĘP.....	7
1.1. Uwagi ogólne.....	7
1.2. Ogólny opis układu logicznego.....	8
1.3. Układ pracy.....	11
2. ALGEBRA BOOLE'A.....	13
2.1. Uwagi ogólne.....	13
2.2. Definicja algebry Boole'a.....	13
2.3. Twierdzenia algebry Boole'a.....	16
2.4. Dwuelementowa algebra Boole'a.....	21
2.5. Geometryczna interpretacja dwuelementowej algebry Boole'a	22
2.6. Funkcje i formuły boolowskie.....	24
2.7. Postacie kanoniczne funkcji boolowskich.....	27
2.8. Postać normalna funkcji boolowskiej i jej dopełnienie....	32
2.9. Funkcje boolowskie dwóch zmiennych.....	35
2.10. Systemy funkcjonalnie pełne.....	38
2.11. Realizacje bramkowe funkcji logicznych.....	40
2.12. Minimalizacja formuł boolowskich.....	45
2.12.1. Heurystyczne znajdowanie postaci minimalnej for-	
muły boolowskiej.....	45
2.12.2. Algorytmiczne i tablicowe metody znajdowania po-	
staci minimalnych formuł boolowskich.....	51
2.13. Faktoryzacja funkcji logicznych.....	60
3. UKŁADY KOMBINACYJNE.....	67
3.1. Uwagi ogólne.....	67
3.2. Sumator jednobitowy.....	68
3.3. Komparator.....	71
3.4. Układy iteracyjne.....	73
3.4.1. Jednowymiarowe układy iteracyjne.....	74
3.4.2. Wielowymiarowe układy iteracyjne.....	81
3.5. Konwertery kodu.....	83
3.6. Multipleksery i demultipleksery.....	91
3.7. Optymalna realizacja układów wielowyjściowych.....	96
3.7.1. Metoda tabulacyjna minimalizacji zespołu funkcji..	96
3.7.2. Realizacja zespołu funkcji z zastosowaniem dekodera	103

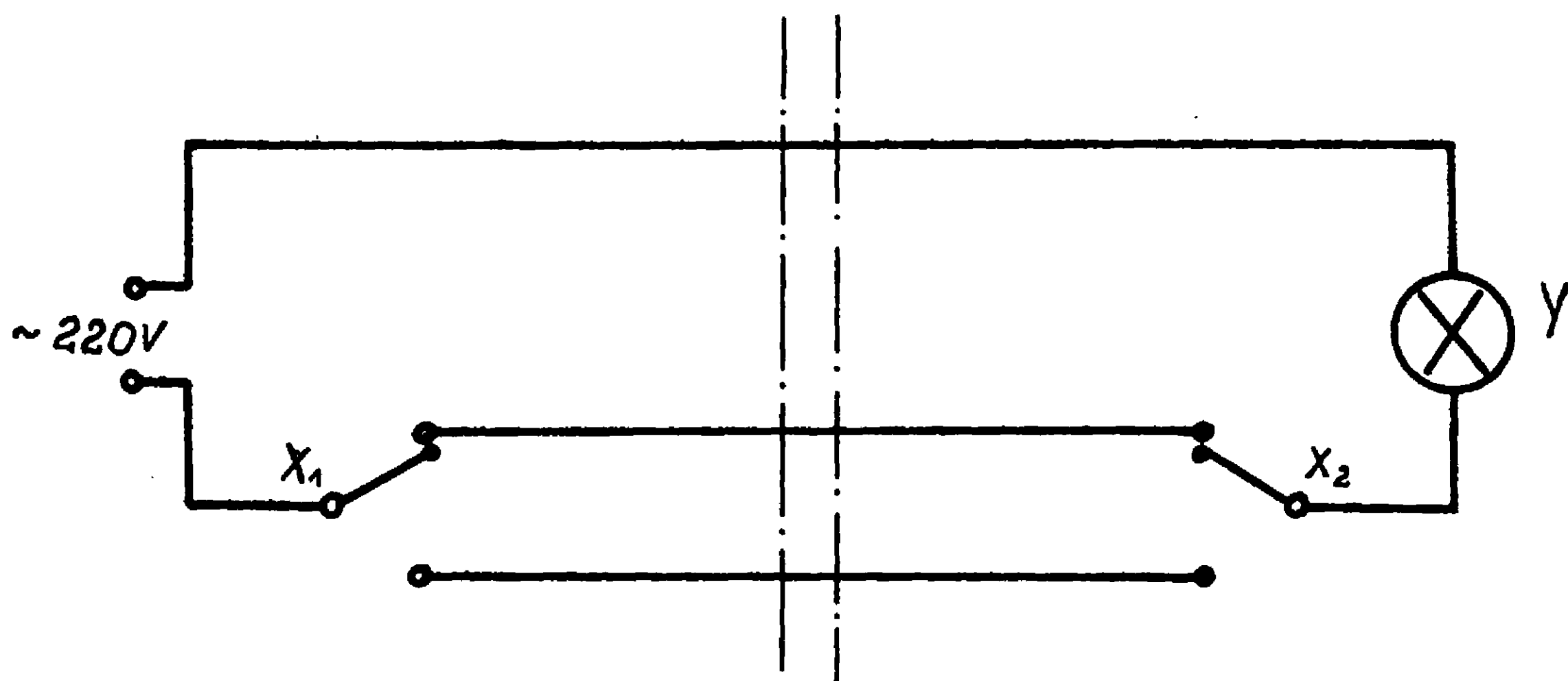
4.		
3.8.	Minimalizacja funkcji nie w pełni określonych.....	105
3.8.1.	Minimalizacja funkcji o małej nieokreśloności.....	105
3.8.2.	Minimalizacja funkcji silnie nieokreślonych.....	106
3.9.	Zjawisko hazardu.....	109
3.9.1.	Hazard statyczny.....	109
3.9.2.	Hazard dynamiczny.....	114
3.10.	Testowanie układów kombinacyjnych.....	116
3.10.1.	Warunki testowania układu.....	116
3.10.2.	Metoda różnic boolowskich.....	117
3.10.3.	Metoda pobudzania ścieżek.....	123
3.10.4.	Diagnostyka układów kombinacyjnych.....	126
4.	UKŁADY LOGICZNE W UJĘCIU TEORII AUTOMATÓW.....	130
4.1.	Uwagi ogólne.....	130
4.2.	Automaty Mealy'ego i Moore'a.....	131
4.3.	Podstawowe definicje.....	136
4.4.	Określenie automatu minimalnego.....	141
4.4.1.	Algorytmy wyznaczania relacji i zbiorów stanów niesprzecznych i nieodróżnialnych.....	141
4.4.2.	Procedura minimalizacji liczby stanów automatu pierwotnego.....	148
4.5.	Podobieństwo automatów Moore'a i Mealy'ego.....	159
5.	SYNCHRONICZNE UKŁADY SEKWENCYJNE.....	168
5.1.	Uwagi ogólne.....	168
5.2.	Przerzutniki.....	169
5.2.1.	Przerzutnik typu RS.....	170
5.2.2.	Przerzutnik typu D.....	175
5.2.3.	Przerzutnik typu JK.....	177
5.2.4.	Przerzutnik typu T.....	180
5.3.	Wyzwalanie przerzutników na zboczach impulsów zegarowych.....	182
5.4.	Analiza pracy synchronicznego układu sekwencyjnego.....	188
5.5.	Liczniki i rejestry.....	192
5.5.1.	Liczniki.....	192
5.5.2.	Rejestry.....	195
5.6.	Projektowanie synchronicznych układów sekwencyjnych.....	201
5.7.	Iterowanie w czasie.....	215
6.	ASYNCHRONICZNE UKŁADY SEKWENCYJNE.....	221
6.1.	Uwagi ogólne.....	221
6.2.	Analiza pracy asynchronicznych układów potencjałowych (statycznych).....	222

6.3. Synteza asynchronicznych układów potencjałowych.....	232
6.3.1. Uwagi ogólne.....	232
6.3.2. Przykłady syntezy asynchronicznych układów poten- cjałowych.....	236
6.3.3. Synteza z zastosowaniem przerzutników asynchroni- cznych.....	257
6.4. Asynchroniczne układy impulsowe (dynamiczne).....	259
7. WSTĘP DO SYSTEMÓW CYFROWYCH.....	269
7.1. Uwagi ogólne.....	269
7.2. Maszyna Turinga.....	269
7.3. Maszyna Поста.....	273
7.4. Maszyny uniwersalne.....	275
7.5. Laboratoryjny model programowalnego systemu cyfrowego....	279
7.5.1. Struktura systemu.....	279
7.5.2. Przykład wariantu sterowania.....	287
7.5.3. Przykłady oprogramowania.....	295
LITERATURA.....	303

1. WSTĘP

1.1. Uwagi ogólne

Układy logiczne są modelami takiej klasy urządzeń wykonywanych przez człowieka, dla której zarówno przyczyny (zmienne niezależne) jak i skutki (zmienne zależne) są wielkościami przyjmującymi wartości dyskretne. Ponadto zmiany wartości wielkości charakterystycznych odbywają się w dyskretnych punktach osi czasu. Dyskretyzacja zarówno wielkości wejściowych (przyczyn) jak i wyjściowych (skutków) oraz czasu, leży u podstaw definicji układu logicznego. W życiu codziennym spotykamy się często z urządzeniami, których pracę można rozpatrywać w kategoriach układu logicznego. Przykładem może być instalacja oświetleniowa w mieszkaniach. Efekt zapalania się żarówki (skutek) zależy jednoznacznie od położenia wyłącznika (przyczyny), przy czym zbiory eksploatacyjnych stanów zarówno żarówki jak i wyłącznika składają się z dwóch elementów (żarówka zgaszona lub zapalona; wyłącznik włączony lub wyłączony). Innym przykładem urządzenia, dla którego można stworzyć model układu logicznego, jest wyłącznik hotelowy, umożliwiający zgaszenie lub zapalenie żarówki Y poprzez niezależne przełączanie dwóch wyłączników (rys. 1.1) x_1 oraz x_2 .



Rys. 1.1. Schemat przełącznika hotelowego

O ile w poprzednim przykładzie można było w sposób łatwy określić stan pracy żarówki w zależności od stanu wyłącznika, o tyle tutaj problem jest nieco bardziej złożony: jeśli x_1 jest w górnym położeniu, załączenie Y następuje, gdy x_2 zajmuje górne położenie, wyłącznie - gdy dolne, ale gdy x_1 jest w położeniu dolnym - przełączenie x_2 w dolne położenie załącza, a w górne - wyłącza Y . Jeśli oznaczyć górne położenie wyłączników przez "1", dolne przez "0" oraz stan załączenia żarówki przez "1", wyłączenie przez "0", to pracę układu w sposób bardziej zwięzły i jednocześnie wyczerpująco opisuje tabela przedstawiona na rysunku 1.2. Tutaj zbiór wszystkich możliwych kombinacji wartości zmiennych wejściowych (x_1, x_2) zawiera cztery elementy (00, 01, 10, 11), natomiast zbiór kombinacji wyjściowych: dwa (0, 1). Symbole "1" oraz "0", choć zostały wybrane dowolnie, stanowią typowe oznaczenia stanów jakie charakteryzują zjawiska fizyczne występujące w urządzeniach, dla których można stworzyć model układu logicznego. Za pomocą tego typu symboliki dokonujemy translacji języka opisowego pracy urządzenia na bardziej uniwersalny i niezależny od realizacji fizycznej języka modelu, co z kolei pozwala na konstruowanie jednolitej teorii umożliwiającej zarówno działalność poznawczą jak i badania, których celem są konkretne zastosowania.

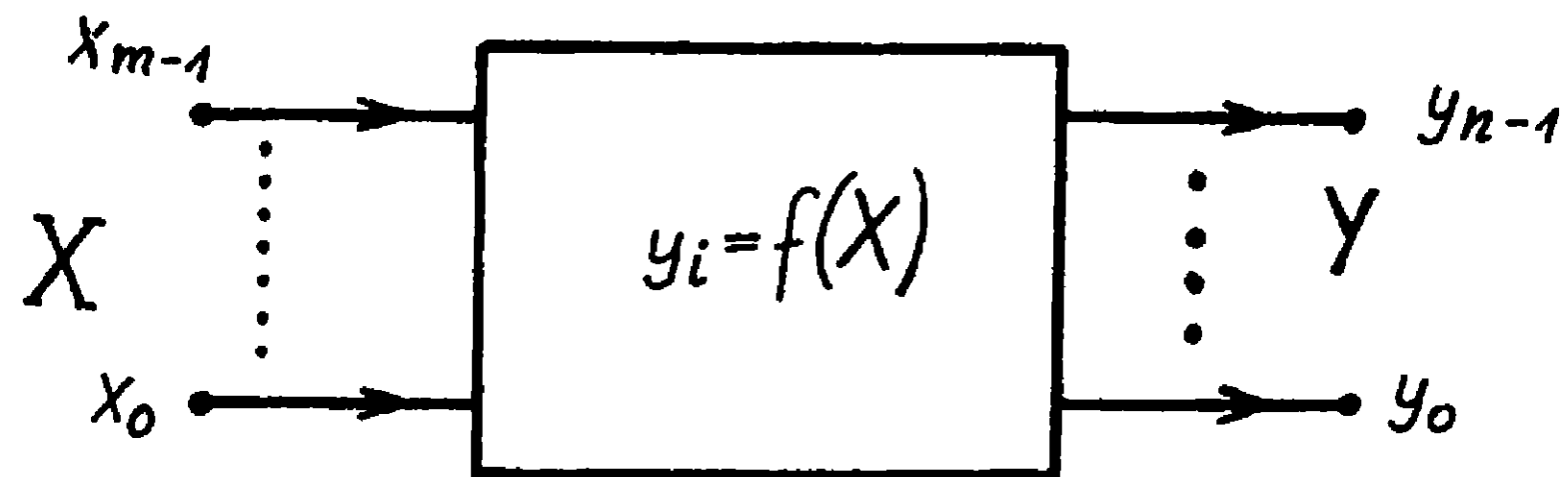
x_1	x_2	Y
0	0	1
0	1	0
1	0	0
1	1	1

Rys. 1.2. Tabelaryczny opis pracy układu przełącznika hotelowego

1.2. Ogólny opis układu logicznego

Omówione w punkcie 1.1 przykłady układów logicznych charakteryzują się jednoznacznym powiązaniem między wielkościami wejściowymi i wyjściem. Należą one do klasy układów logicznych, zwanych układami kombinacyjnymi. Ogólny schemat układu kombinacyjnego przedstawiono na rysunku 1.3. Układ posiada m wejść i n wyjść ($m, n = 1, 2, 3, \dots$), przy czym sygnały zarówno na poszczególnych wejściach jak i wyjściach mogą przyjmować jedynie dwie wykluczające się wartości, oznaczone symbolicznie jako "0" i "1". Sygnały te nazywamy sygnałami binarnymi, a odpowiednio końcówki układu wejściami lub wyjściami binarnymi. Dla pełnego opisu układu kombinacyjnego konieczne i zarazem wystarczające jest podanie dla każdego wyjścia y_i ($i = 0, 1, \dots, n-1$) wartości sygnału, ja-

kie ono przyjmie dla wszystkich mogących wystąpić kombinacji sygnałów na wejściach x_i ($i = 0, 1, \dots, m-1$). Do tego celu mogą służyć tzw. tabele wartości logicznych (rys. 1.4), analogiczne do tabeli przedstawionej na rysunku 1.2.



Rys. 1.3. Ogólny schemat kombinacyjnego układu logicznego

x_{m-1}	x_{m-2}	$\dots$	x_0	y_{n-1}	y_{n-2}	$\dots$	y_0
0	0	$\dots$	0	0	0	$\dots$	1
0	0	$\dots$	1	1	0	$\dots$	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
1	1	$\dots$	1	0	1	$\dots$	1

Rys. 1.4. Tabela wartości logicznych opisująca układ kombinacyjny

Inną formą opisu układu jest opis analityczny, przy czym szczególnie zwięzłą postać otrzymujemy traktując m -wejść dwuwartościowych jako jeden wektor m -wymiarowy X o maksimum 2^m wartościach oraz n -wyjść jako jeden wektor wyjściowy Y o maksimum 2^n wartościach. Przykładowo jeśli $m = 3$ oraz $n = 2$, to maksymalny zbiór wartości trójwymiarowego wektora X wynosi $2^3 = 8$

$$X = \{ \langle 000 \rangle, \langle 001 \rangle, \langle 010 \rangle, \langle 011 \rangle, \langle 100 \rangle, \langle 101 \rangle, \langle 110 \rangle, \langle 111 \rangle \},$$

natomiast dla dwuwymiarowego wektora Y otrzymamy następujący zbiór maksymalny

$$Y = \{ \langle 00 \rangle, \langle 01 \rangle, \langle 10 \rangle, \langle 11 \rangle \}.$$

Układ jest jednoznacznie określony, jeśli podamy odwzorowanie $X \rightarrow Y$. Odwzorowanie to można określić za pomocą tabeli wartości logicznych (rys. 1.4) lub też analitycznie wykorzystując do tego celu tzw. dwuelementową algebrę Boole'a opisaną w rozdziale 2. Odwzorowanie to definiuje jednocześnie funkcję wyjściową λ , która każdemu elementowi zbioru wartości wektora X przyporządkowuje określony element zbioru wartości wektora Y .

Uogólnieniem kombinacyjnego układu logicznego jest układ logiczny sekwencyjny. W układzie tym stan wyjść Y zależy nie tylko od aktualnej wartości wektora X , ale i od sekwencji zmian tego wektora, która stan aktualny poprzedziła. W układzie sekwencyjnym pojawia się więc dyskretna skala czasu, rozumiana jako zbiór kolejnych momentów zmian wartości tzw. wektora stanów wewnętrznych S , charakteryzującego historię układu. Liczba stanów wewnętrznych, niezbędnych do rejestracji wszystkich możliwych historii układu, warunkuje liczbę składowych binarnych wektora S zgodnie z relacją

$$u \leq 2^l,$$

gdzie: u - liczba stanów wewnętrznych,

l - liczba składowych binarnych wektora S .

Sekwencyjny układ logiczny jest jednoznacznie określony, jeśli podamy odwzorowanie wektorów X i $S \rightarrow Y$ oraz odwzorowanie wektorów X i $S \rightarrow S$. W pierwszym przypadku definiujemy funkcję wyjściową λ dla układu sekwencyjnego, w drugim - tzw. funkcję przejść δ , która nie występuje w układzie kombinacyjnym. Oczywiście zakładamy przy tym, że wszystkie zbiory są dane. Zachowanie się układu sekwencyjnego w okresie między dyskretnymi zmianami wektora stanu, może być utożsamione z zachowaniem układu kombinacyjnego, którego liczbę wejść m powiększono o l składowych wektora stanu. Uogólniony wektor wejściowy ma więc tutaj $m + l$ składowych. W konsekwencji procedura analizy i syntezy układu sekwencyjnego sprowadza się do metod charakterystycznych dla układu kombinacyjnego. Należy zauważyć, że układ kombinacyjny można uważać za przypadek szczególny układu sekwencyjnego, w którym $l = 0$.

Dyskretna skala czasu może być narzucona z zewnątrz przez tzw. zegar synchronizujący, bądź też układ wygenerowuje taką skalę sam. W pierwszym przypadku układ sekwencyjny nosi nazwę układu synchronicznego, w drugim - asynchronicznego.

1.3. Układ pracy

W rozdziale 2 omówiono podstawy teoretyczne algebry Boole'a, a szczególnie jej zastosowań w analizie i syntezie projektowej układów logicznych. Rozdział 3 poświęcono omówieniu zagadnień związanych z funkcjonowaniem i konstrukcją podstawowych kombinacyjnych układów logicznych, stosowanych powszechnie w systemach przełączających i cyfrowych. Rozdział 4 zawiera podstawy teoretyczne algebraicznej teorii automatów, ukierunkowane do potrzeb zastosowań w sekwencyjnych układach logicznych. Synchroniczne układy sekwencyjne omówiono w rozdziale 5, natomiast asynchroniczne - w rozdziale 6. W ostatnim przypadku główny nacisk położono na asynchroniczne realizacje potencjałowe, mające stosunkowo największe znaczenie w tej klasie układów. Rozdział 7 zawiera wiadomości wstępne o systemach cyfrowych. Omówiono w nim bliżej model laboratoryjny prostego programowalnego systemu liczącego.

Zgodnie z ogólnie przyjętą symboliką, uporządkowaną "n-tkę" elementów oznaczono nawiasami ostrokątnymi $\langle \dots \rangle$, np.

$$\langle a_{n-1}, \dots, a_1, a_0 \rangle.$$

Przy takim oznaczeniu, kolejność występowania elementów odgrywa istotną rolę. W punkcie 1.2 w ten sposób zapisano wartości wektorów wejściowych i wyjściowych. Jeśli nie budzi to wątpliwości, przecinki między poszczególnymi elementami ujętymi w nawiasy ostrokątne mogą być pomijane. W przypadkach oczywistych, również nawiasy mogą być pomijane. Dotyczy to zwłaszcza zapisów wagowych, tzn. takich, dla których z każdą pozycją jest związana waga charakterystyczna dla liczb binarnych. Na przykład zapis 1101 odpowiada liczbie dziesiętnej 13 i oznacza uporządkowaną "czwórkę" $\langle 1, 1, 0, 1 \rangle$.

Do oznaczania nazw wektorów zastosowano zwykłe duże litery. Nie wprowadza to wątpliwości, gdyż asortyment nazw jest ograniczony i dotyczy głównie wejścia X, wyjścia Y, stanów wewnętrznych S oraz wektora pomocniczego wzbudzeń Z. Nazewnictwo automatowe nie operuje pojęciami wektorów, ale nazwami alfabetów. Użycie tych samych symboli nie wprowadza jednak niejednoznaczności, gdyż nazewnictwo automatowe bezpośrednio tłumaczy się na nazewnictwo stosowane w układach logicznych. Przykładowo, w teorii automatów symbolem X oznacza się najczęściej alfabet wejściowy, który jest zbiorem elementów abstrakcyjnych zwanych literami wejściowymi. Alfabet ten w konwencji układu logicznego repre-

zentuje wartości wektora wejściowego X . To samo dotyczy wyjścia i stanów wewnętrznych, używanie więc tych samych nazw nie prowadzi do niejednoznaczności.

Zbiory są ujęte w nawiasy klamrowe $\{...\}$. W nawiasach tych mogą występować nazwy zbiorów, np. $\{Q_{\max}\}$, elementy tworzące zbiór, np. $\{a_{n-1}, \dots, a_0\}$ lub warunki definiujące zbiór, np. $\{a | W(a)\}$, gdzie $W(a)$ reprezentuje własności elementów a . W literaturze jest pewna niekonsekwencja w odniesieniu do nazw zbiorów - najczęściej, o ile nie budzi to wątpliwości, nawiasy klamrowe są opuszczane. I tak, zbiór liter wejściowych jest oznaczany literą X , zamiast $\{X\}$; zbiór liter wyjściowych Y zamiast $\{Y\}$ itp. W niniejszej pracy również występują powyższe odstępstwa oznaczeń.

Zastosowane nazewnictwo jest oparte na nazewnictwie anglosaskim. Wynika to z faktu, że wiodąca światowa literatura źródłowa używa tylko tego nazewnictwa i w tej sytuacji tworzenie odpowiedników językowych polskich jest niecelowe. W przypadku skrótów w nawiasach podano tworzące dany skrót pełne nazwy angielskie.

2. ALGEBRA BOOLE'A

2.1. Uwagi ogólne

Algebra Boole'a jest systemem dedukcyjnym, który opracował w roku 1854 matematyk i logik angielski George Boole. Stanowi ona ukoronowanie zapoczątkowanych już w starożytności prac nad usystematyzowaniem reguł wnioskowania. Aby zdefiniować dowolny system algebraiczny należy podać zbiór elementów rozważanych w algebrze wraz z elementami, zbiór operacji wykonywanych na elementach i relacji między nimi oraz zbiór aksjomatów charakteryzujących operacje oraz relacje. Algebra Boole'a może być definiowana w różny sposób. Największą popularność zdobyła definicja podana przez E.V. Huntingtona w 1904 r. oraz jej przypadek szczególny dotyczący algebry binarnej sieci przełączających, opracowany przez C.E. Shannon'a w 1938 r. Prace Shannon'a zapoczątkowały ogromny wzrost zainteresowania zastosowaniami algebry Boole'a do celów analizy i syntezy projektowej układów logicznych.

2.2. Definicja algebry Boole'a

Unowocześnioną wersję definicji algebry Boole'a podanej przez Huntingtona można sformułować w sposób przedstawiony poniżej.

Definicja 2.1. Algebrą Boole'a nazywamy zbiór elementów oznaczony zwyczajowo przez B , w którym:

- 1) istnieją co najmniej dwa różne elementy,
- 2) istnieją dwa elementy wyróżnione oznaczane najczęściej przez "0" i "1", przy czym zakładamy, że są one różne,
- 3) są określone dwa operatory dwuargumentowe oznaczone najczęściej symbolami sumy i iloczynu zapożyczonymi ze zwykłej algebry i zwane sumą logiczną $a + b$, iloczynem logicznym $a \cdot b$ ($a, b \in B$) oraz jeden operator jednoargumentowy zwany dopełnieniem oznaczany $\bar{a}$ (lub $\neg a$); elementy: $(a + b)$, $(a \cdot b)$, $\bar{a}$ należą do B , a więc w wyniku operacji sumy, iloczynu i dopełnienia nie wychodzimy poza zbiór B ,

4) jest określona relacja równoważności oznaczona "=" i spełniająca warunek: dla każdego elementu $a, b, c \in B$, jeśli $a = b$, to $\bar{a} = \bar{b}$ oraz $a + c = b + c$ i $a \cdot c = b \cdot c$,

5) operatory sumy logicznej, iloczynu logicznego i dopełnienia spełniają dla wszystkich elementów zbioru B następujące postulaty (aksjomaty) ($a, b, c \in B$):

A_1 - postulat przemienności: operatory sumy i iloczynu są przemienne, tzn.:

$$a + b = b + a \quad \text{oraz} \quad a \cdot b = b \cdot a,$$

A_2 - postulat rozdzielczości: operatory sumy i iloczynu są wzajemnie rozdzielne, tzn.:

$$a + (b \cdot c) = (a + b) \cdot (a + c)$$

oraz

$$a \cdot (b + c) = (a \cdot b) + (a \cdot c),$$

A_3 - postulat o elemencie identycznościowym: dla operatora sumy elementem identycznościowym jest element wyróżniony oznaczony "0", tzn.:

$$a + 0 = a,$$

dla operatora iloczynu elementem identycznościowym jest element wyróżniony oznaczony "1", tzn.:

$$a \cdot 1 = a,$$

A_4 - postulat o dopełnieniu: dla operatora sumy i iloczynu obowiązują zależności:

$$a + \bar{a} = 1 \quad \text{oraz} \quad a \cdot \bar{a} = 0,$$

$\bar{a}$ - nazywamy dopełnieniem.

Tak zdefiniowana algebra jest dualna względem operatorów dwuargumentowych, tzn. oznaczenia "+" i "." można traktować zamiennie, jeśli zmienimy również oznaczenia elementów wyróżnionych z "0" na "1" i z "1" na "0". Aksjomaty zdefiniowane dla jednego operatora przyjmują wówczas postać charakterystyczną dla drugiego. Dotyczy to również zależności boole'owskich w związku z czym twierdzenia prawdziwe dla danej postaci są również prawdziwe dla postaci dualnych, a nieprawdziwe

stają się nieprawdziwymi w wersji dualnej. Dualność jest podstawową zaletą podanej definicji algebry Boole'a.

Porównując podane aksjomaty z aksjomatami "zwykłej" algebry widzimy pełną analogię w odniesieniu do postulatów A_1 oraz A_3 (dlatego dla oznaczenia operatorów boole'owskich wygodnie jest wykorzystać klasyczne symbole sumy i iloczynu). Postulat rozdzielczości (A_2) w zwykłej algebrze jest spełniony tylko w odniesieniu do rozdzielania iloczynu względem sumy. Postulat o dopełnieniu (A_4) przypomina trochę postulat o elemencie odwrotnym, definiującym odejmowanie i dzielenie. W zbiorze aksjomatów podanej definicji algebry Boole'a nie występuje postulat o łączności, charakterystyczny dla algebry klasycznej. Daje się on tutaj wyprowadzić z innych aksjomatów, należąc tym samym do zbioru twierdzeń rozpatrywanej algebry.

Przykład 2.1. Przyjmijmy, że elementami zbioru B są wszystkie podzbiory zbioru dwuelementowego $\{a, b\}$; oznaczmy:

zbiór pełny: $\{a, b\} = 1$ (element wyróżniony),

podzbiór: $\{a\} = e_1$,

podzbiór: $\{b\} = e_2$,

zbiór pusty: $\{-\} = 0$ (element wyróżniony).

Można więc napisać:

$$B = \{1, e_1, e_2, 0\}.$$

Traktując sumę, iloczyn oraz dopełnienie zbiorów jako odpowiednie operatory algebry Boole'a, łatwo zauważyć, że zbiór B spełnia wymagania podane w definicji 2.1. Przykładowo:

$$\begin{aligned} \text{Aksjomat } A_3: \quad \{a\} \cap \{a, b\} &= \{a\} \quad \text{lub} \quad e_1 \cdot 1 = e_1, \\ \{a\} \cup \{-\} &= \{a\} \quad \text{lub} \quad e_1 + 0 = e_1, \end{aligned}$$

$$\begin{aligned} \text{Aksjomat } A_4: \quad \{a\} \cap \{b\} &= \{-\} \quad \text{lub} \quad e_1 \cdot e_2 = 0, \\ \{a\} \cup \{b\} &= \{a, b\} \quad \text{lub} \quad e_1 + e_2 = 1. \end{aligned}$$

Z ostatnich dwóch zależności wynika, że $\bar{e}_1 = e_2$.

2.3. Twierdzenia algebry Boole'a

W oparciu o definicję 2.1 algebry Boole'a, a w szczególności zawarte w niej aksjomaty można formułować i udowadniać twierdzenia tworzące rozpatrywaną teorię. Poniżej podano wraz z dowodami kilka twierdzeń istotnych zarówno z punktu widzenia poznawczego jak i aplikacyjnego.

Twierdzenie 2.1. W każdej algebrze Boole'a istnieją tylko dwa różne od siebie elementy wyróżnione.

Dowód. Zaprzeczmy tezie i załóżmy, że istnieją dwa elementy 0 oraz 0^* i 1 oraz 1^* ($0 \neq 0^*$ oraz $1 \neq 1^*$). wtedy zgodnie z definicją 2.1 można napisać

$$a \cdot 1 = a.$$

Podstawiając $a = 1^*$, otrzymamy

$$1^* \cdot 1 = 1^*.$$

Ale jednocześnie, z właściwości elementu wyróżnionego 1^* wynika, że

$$a \cdot 1^* = a.$$

Jeśli podstawić $a = 1$, to otrzymamy

$$1 \cdot 1^* = 1.$$

Z aksjomatu o przemienności (A_1) wynika że

$$1 \cdot 1^* = 1^* \cdot 1,$$

a więc musi być

$$1 = 1^*.$$

Postępując w sposób dualny wykazujemy, że

$$0 = 0^*.$$

Twierdzenie 2.2. Dla każdego elementu $a \in B$ (def. 2.1), istnieje jeden i tylko jeden element będący jego dopełnieniem.

Dowód. Z definicji 2.1 nie wynika bezpośrednio, że istnieje tylko jeden element $\bar{a}$ będący dopełnieniem a . Załóżmy więc, że element a ma dwa uzupełnienia: $\bar{a}$ i a^* . Można napisać:

$$\begin{aligned}
 a^* &= a^* + 0 = && \text{na podstawie } (A_3) \\
 &= a^* + (a \cdot \bar{a}) = && (A_4) \\
 &= (a^* + a) \cdot (a^* + \bar{a}) = && (A_2) \\
 &= (a + a^*) \cdot (a^* + \bar{a}) = && (A_1) \\
 &= 1 \cdot (a^* + \bar{a}) = && (A_4) \\
 &= 1 \cdot (\bar{a} + a^*) = && (A_1) \\
 &= (a + \bar{a}) \cdot (\bar{a} + a^*) = && (A_4) \\
 &= (\bar{a} + a) \cdot (\bar{a} + a^*) = && (A_1) \\
 &= \bar{a} + (a \cdot a^*) = && (A_2) \\
 &= \bar{a} + 0 = && (A_4) \\
 &= \bar{a} && (A_3)
 \end{aligned}$$

A więc $a^* = \bar{a}$, czyli mamy do czynienia tylko z jednym elementem będącym dopełnieniem a .

Twierdzenie 2.3

$$x + x = x \quad \text{oraz} \quad x \cdot x = x$$

Dowód:

$$\begin{aligned}
 x + x &= (x + x) \cdot 1 = && (A_3) \\
 &= (x + x) \cdot (x + \bar{x}) = && (A_4) \\
 &= x + (x \cdot \bar{x}) = && (A_2) \\
 &= x + 0 = && (A_4) \\
 &= x && (A_3)
 \end{aligned}$$

Dowód: $x \cdot x = x$ - na zasadzie dualności.

Twierdzenie 2.4

$$x + 1 = 1 \quad \text{oraz} \quad x \cdot 0 = 0$$

Dowód:

$$\begin{aligned}
 x + 1 &= 1 \cdot (x + 1) = & (A_1, A_3) \\
 &= (x + \bar{x}) \cdot (x + 1) = & (A_4) \\
 &= x + (\bar{x} \cdot 1) = & (A_2) \\
 &= x + \bar{x} = & (A_3) \\
 &= 1 & (A_4)
 \end{aligned}$$

Dowód: $x \cdot 0 = 0$ - na zasadzie dualności.Twierdzenie 2.5

$$1 + x = 1 \quad \text{oraz} \quad 0 \cdot x = 0$$

Dowód: na podstawie twierdzenia 2.4 oraz aksjomatu A_1 .Twierdzenie 2.6. Dopełnienie dopełnienia elementu a równa się elementowi a , tzn.

$$\overline{(\bar{a})} = a.$$

Dowód: na podstawie postulatów o przemienności (A_1), postulat o dopełnieniu można zapisać w postaci

$$\bar{a} + a = 1 \quad \text{oraz} \quad \bar{a} \cdot a = 0.$$

Wynika stąd, że dopełnieniem $\bar{a}$ jest a . Zgodnie z twierdzeniem 2.2 może być tylko jedno dopełnienie, więc:

$$\overline{(\bar{a})} = a.$$

Twierdzenie 2.7

$$x + (x \cdot y) = x \quad \text{oraz} \quad x \cdot (x + y) = x$$

Jest to tzw. twierdzenie o absorpcji.

$$\begin{aligned}
 \text{Dowód: } x + (x \cdot y) &= (x \cdot 1) + (x \cdot y) = & (A_3) \\
 &= x \cdot (1 + y) = & (A_2) \\
 &= x \cdot 1 = & (\text{Tw. 2.5}) \\
 &= x & (A_3)
 \end{aligned}$$

Dowód: $x \cdot (x + y) = x$ - na zasadzie dualności.

Twierdzenie 2.8

$$x + (\bar{x} \cdot y) = x + y \quad \text{oraz} \quad x \cdot (\bar{x} + y) = x \cdot y$$

$$\begin{aligned} \text{Dowód: } x + (\bar{x} \cdot y) &= (x + \bar{x}) \cdot (x + y) = && (A_2) \\ &= 1 \cdot (x + y) = && (A_4) \\ &= x + y && (A_1 \text{ i } A_3) \end{aligned}$$

Dowód: $x \cdot (\bar{x} + y) = x \cdot y$ - na zasadzie dualności.

Twierdzenie 2.9

$$(a + b) + c = a + (b + c) \quad \text{oraz} \quad (a \cdot b) \cdot c = a \cdot (b \cdot c)$$

Jest to prawo łączności.

$$\begin{aligned} \text{Dowód: } (a + b) + c &= (a + b) + [c + (b \cdot \bar{b})] = && (A_3 \text{ i } A_4) \\ &= (a + b) + [(b + c) \cdot (\bar{b} + c)] = && (A_1 \text{ i } A_2) \\ &= [(a + b) + (b + c)] \cdot [(a + b) + (\bar{b} + c)] \cdot (A_2) \end{aligned}$$

Analogicznie otrzymamy, że:

$$a + (b + c) = [(a + b) + (b + c)] \cdot [(a + \bar{b}) + (b + c)].$$

Na podstawie relacji równoważności (pkt. 4 definicji 2.1) oraz aksjomatu A_1 dla dowodu wystarczy wykazać, że

$$[(a + b) + (\bar{b} + c)] = [(a + \bar{b}) + (b + c)].$$

Jeśli wyrażenie powyższe jest prawdziwe, to zgodnie z relacją równoważności prawdziwe jest też wyrażenie następujące

$$[(a + b) + (\bar{b} + c)] \cdot b = [(a + \bar{b}) + (b + c)] \cdot b.$$

Stwierdzenie prawdziwości tego wyrażenia udowadnia jednocześnie tezę główną. Lewa strona wyrażenia daje się przekształcić w sposób następujący:

$$\begin{aligned}
[(a + b) + (\bar{b} + c)] \cdot b &= [(a + b) \cdot b] + [(\bar{b} + c) \cdot b] = & (A_1, A_2) \\
&= [(a \cdot b) + (b \cdot b)] + [(\bar{b} \cdot b) + (c \cdot b)] = & (A_2) \\
&= [(a \cdot b) + b] + [0 + (c \cdot b)] = & (A_1, A_4, \text{Tw.2.3}) \\
&= [(a + 1) \cdot b] + (c \cdot b) = & (A_1, A_2, A_3) \\
&= b + (c \cdot b) = & (\text{Tw.2.4}, A_1, A_3) \\
&= (b \cdot 1) + (b \cdot c) = & (A_1, A_3) \\
&= b \cdot (1 + c) = & (A_2) \\
&= 1 \cdot b & (\text{Tw.2.5}, A_1).
\end{aligned}$$

Zgodnie z relacją równoważności, można napisać

$$[(a + b) + (\bar{b} + c)] = 1.$$

Analogicznie można wykazać, że przekształcenie prawej strony daje taki sam rezultat, tzn.

$$[(a + \bar{b}) + (b + c)] = 1.$$

Lewa strona równa się prawej, a więc i teza główna jest udowodniona.

Dowód: $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ - na zasadzie dualności.

Twierdzenie 2.10

$$\overline{(a + b)} = (\bar{a} \cdot \bar{b}) \quad \text{oraz} \quad \overline{(a \cdot b)} = \bar{a} + \bar{b}$$

Są to prawa de Morgana.

Dowód: jeśli teza jest słuszna, to $\bar{a} \cdot \bar{b}$ jest dopełnieniem $(a + b)$. Wystarczy więc wykazać, że

$$(a + b) + (\bar{a} \cdot \bar{b}) = 1.$$

Otrzymamy wówczas:

$$\begin{aligned}
(a + b) + (\bar{a} \cdot \bar{b}) &= [(a + b) + \bar{a}] \cdot [(a + b) + \bar{b}] = & (A_2) \\
&= [(a + \bar{a}) + b] \cdot [a + (b + \bar{b})] = & (\text{Tw.2.9}, A_1) \\
&= (1 + b)(a + 1) = & (A_4) \\
&= 1 \cdot 1 = & (\text{Tw.2.4}, A_1) \\
&= 1. & (\text{Tw.2.3})
\end{aligned}$$

Dowód: $\overline{(a \cdot b)} = \bar{a} + \bar{b}$ - na zasadzie dualności.

2.4. Dwuelementowa algebra Boole'a

Dotychczasowe rozważania dotyczyły ogólnych własności algebr Boole'a, niezależnych od ich konkretnych realizacji. W tym punkcie zajmiemy się wariantem algebry dwuelementowej, mającym bezpośrednie zastosowanie w teorii układów logicznych. Zgodnie z definicją 2.1, określamy zbiór $B = \{0, 1\}$, przy czym $0 \neq 1$. Obydwa elementy są jednocześnie elementami wyróżnionymi. Operatory definiujemy za pomocą tabel przedstawionych na rysunku 2.1. Należy teraz sprawdzić czy są spełnione aksjomaty podane w punkcie 5 definicji 2.1. Aksjomat A_1 jest spełniony z uwagi na symetrię tabel. Aby zweryfikować aksjomat A_2 posłużmy się tabelą przedstawioną na rysunku 2.2.

a)

a	b	$a \cdot b$
0	0	0
0	1	0
1	0	0
1	1	1

b)

a	b	$a + b$
0	0	0
0	1	1
1	0	1
1	1	1

c)

a	$\bar{a}$
0	1
1	0

Rys. 2.1. Definicje operatorów: a) iloczynu, b) sumy, c) dopełnienia

Tabelę wypełniono zgodnie z definicjami operacji podanymi na rys. 2.2. Jak widać, rubryka $L_1 = P_1$ oraz $L_2 = P_2$, czyli aksjomat o wzajemnej rozdzielczości operatorów sumy i iloczynu jest spełniony. Aksjomat o elemencie identycznościowym A_3 sprawdzamy (rys. 2.1) dla $b = 1$ w przypadku iloczynu (tabela a) oraz $b = 0$ w przypadku sumy (tabela b). Z tabel definicyjnych wynika bezpośrednio zgodność z aksjomatem A_3 . Również bezpośrednio z tabel definicyjnych wynika spełnienie aksjomatu o dopełnieniu A_4 , bowiem:

$$0 \cdot 1 = 0, \quad 1 \cdot 0 = 0; \quad 0 + 1 = 1, \quad 1 + 0 = 1.$$

			L_1			P_1			L_2			P_2
a	b	c	$b \cdot c$	$a+(b \cdot c)$	$a+b$	$a+c$	$(a+b) \cdot (a+c)$	$b+c$	$a \cdot (b+c)$	$a \cdot b$	$a \cdot c$	$(a \cdot b)+(a \cdot c)$
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	1	0	0	0	0
0	1	0	0	0	1	0	0	1	0	0	0	0
0	1	1	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	0	0	0	0	0
1	0	1	0	1	1	1	1	1	1	0	1	1
1	1	0	0	1	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	1	1	1

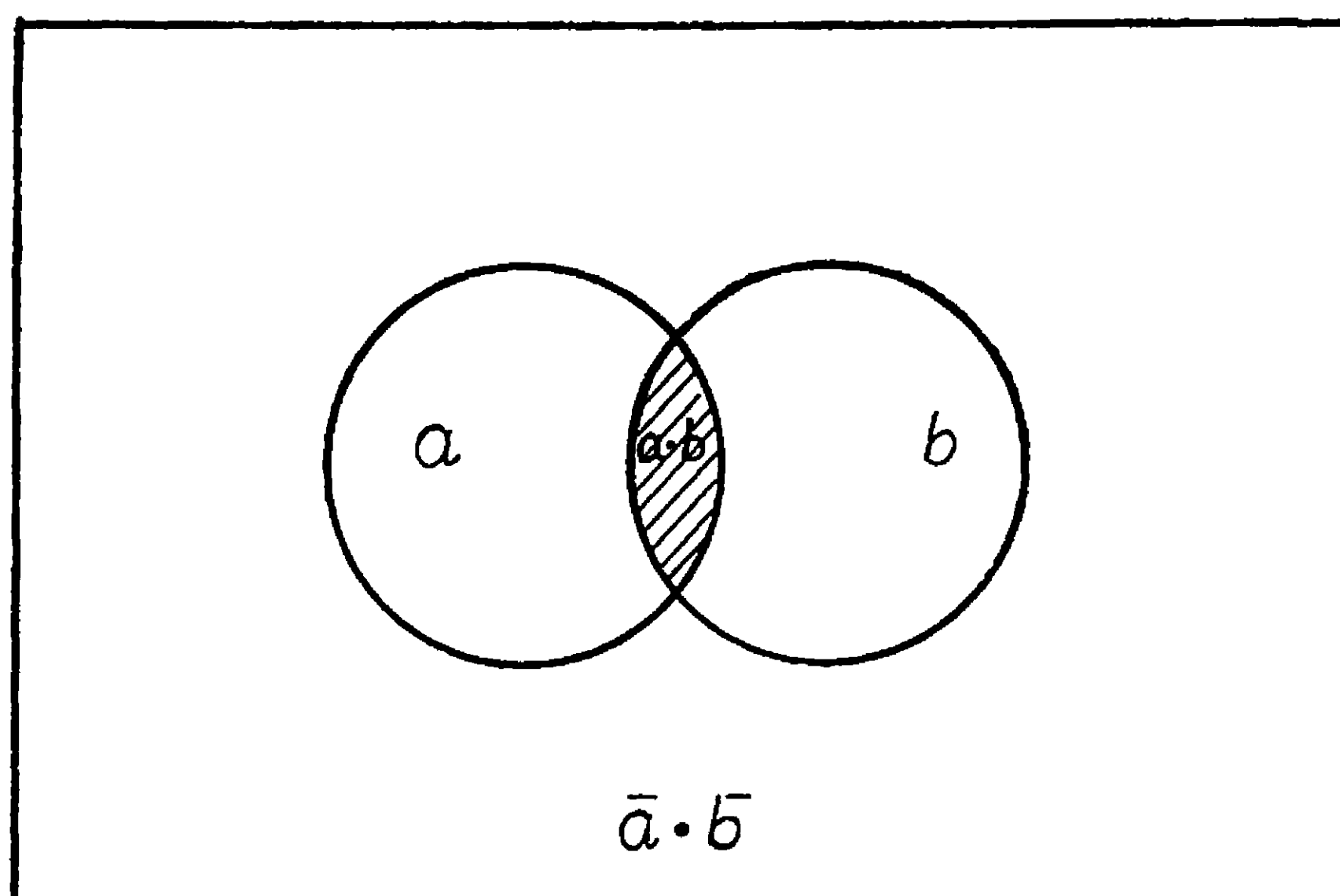
Rys. 2.2: Tabela dla weryfikacji aksjomatu o rozdzielczości operatorów sumy i iloczynu

Zdefiniowana powyżej algebra jest więc algebrą Boole'a, określoną na zbiorze $B = \{0, 1\}$. O ile nie zostanie to wyraźnie zaznaczone, dalsze rozważania dotyczyć będą głównie tej algebry.

2.5. Geometryczna interpretacja dwuelementowej algebry Boole'a

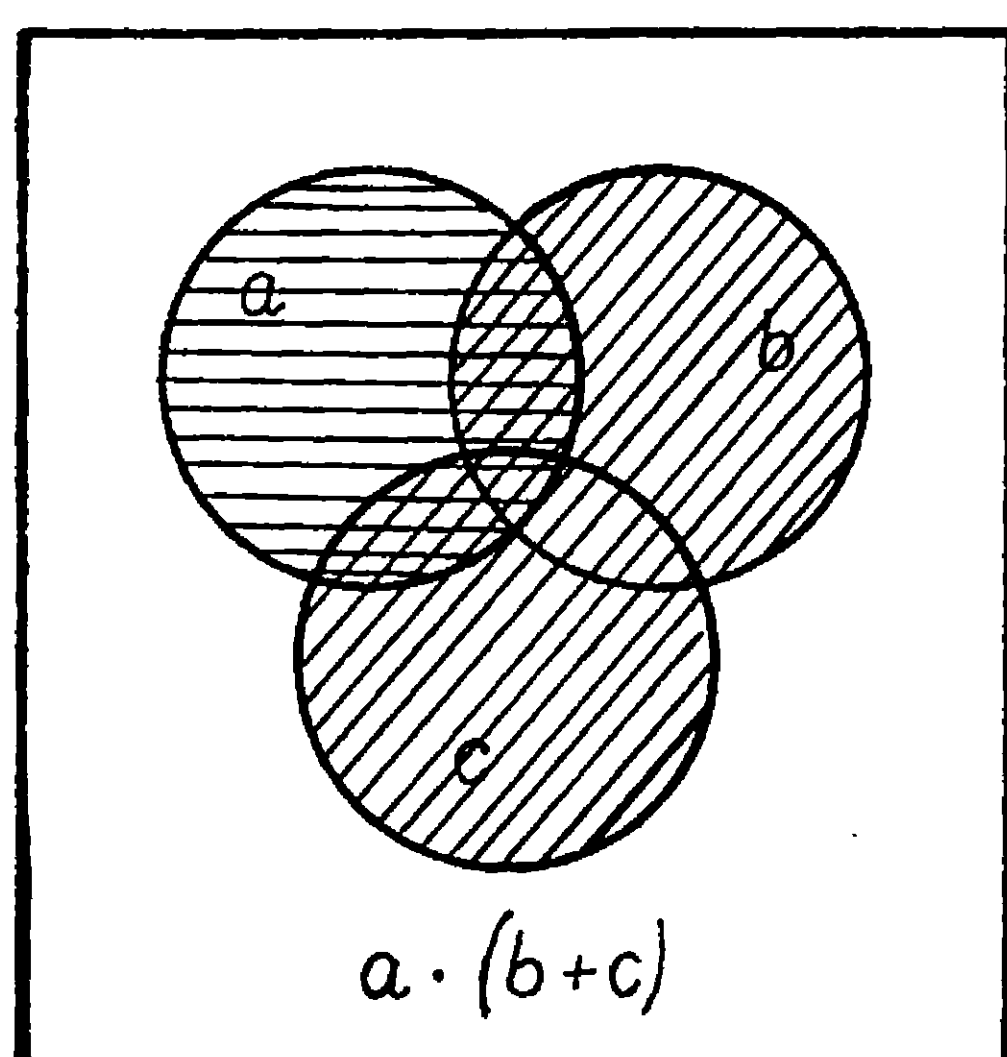
Omówione uprzednio zależności można w przypadku zdefiniowanej w punkcie 2.4 algebry dwuelementowej przedstawić w postaci tzw. wykresów Venna. Najczęściej przedstawiają one prostokąt z zaznaczonymi za pomocą okręgów obszarami, w których zmienne przyjmują wartość 1 i poza którymi ich wartość wynosi 0. Na rysunku 2.3 koło lewe wraz z okręgiem przedstawia obszar, dla którego $a = 1$, koło prawe obszar, dla którego $b = 1$. Poza okręgami wartości a i b są równe 0. Punkty wspólne dla obu kół, obszar zakreskowany, reprezentują obszar, w którym a i b są równe 1. Na rysunku 2.4 przedstawiono graficzną ilustrację prawa rozdzielczości operatora iloczynu względem operatora sumy. Na rysunku a) obszar reprezentujący $a = 1$ został zakreskowany poziomo, natomiast obszar $a + b = 1$ - skośnie. Obszar wspólny, a więc $a \cdot (a + b) = 1$ został zakreskowany łącznie. Na rysunku b) obszar $a \cdot b = 1$ oraz

$a \cdot c = 1$ łącznie tworzą obszar $(a \cdot b) + (a \cdot c) = 1$, identyczny z obszarem podwójnie zakreskowanym, przedstawionym na rysunku a). W ten sposób można ilustrować wszystkie zależności charakterystyczne dla zdefiniowanej w punkcie 2.4 algebry Boole'a.

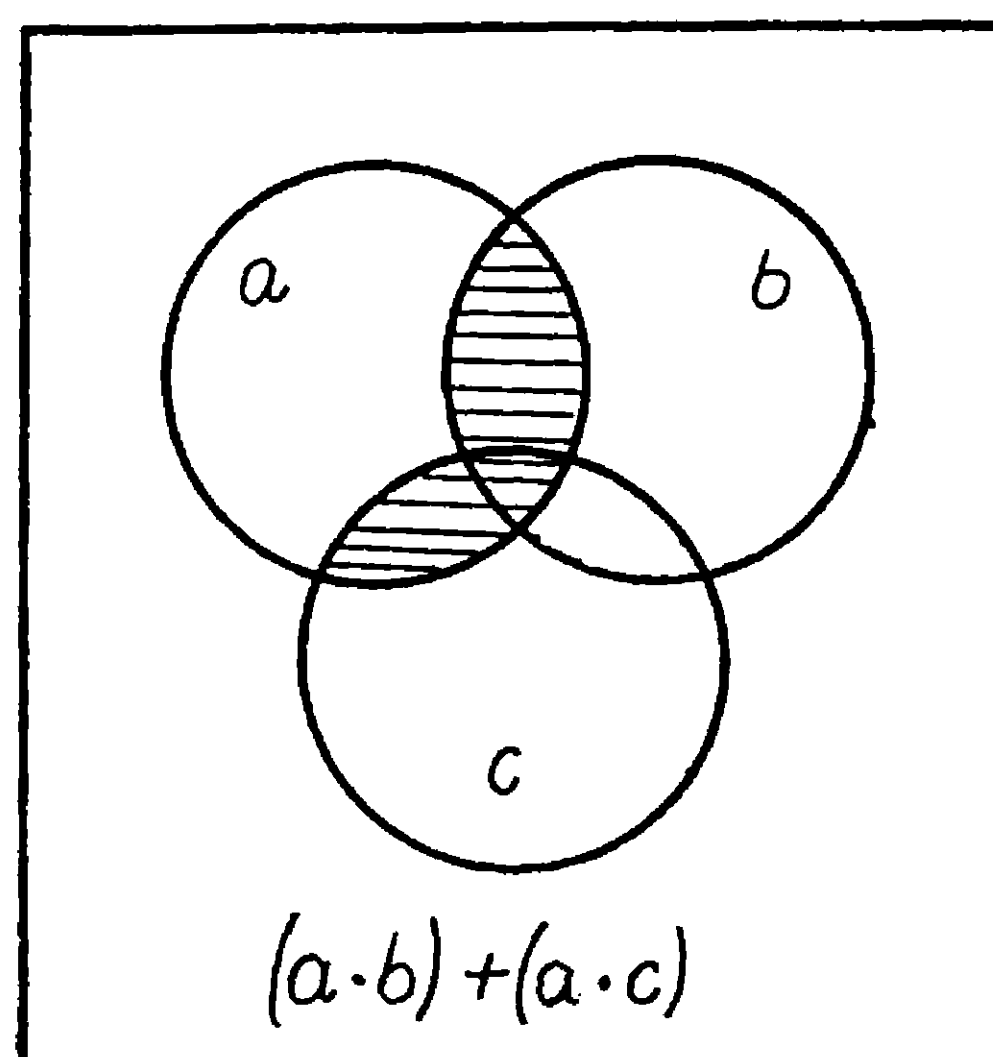


Rys. 2.3. Wykres Venna dla dwóch zmiennych

a)



b)



Rys. 2.4. Wykresy Venna (a,b) ilustrujące prawo rozdzielności

2.6. Funkcje i formuły boolowskie

Definicja funkcji w ujęciu relacyjnym jest przedstawiona jako pewien zbiór $\{F\}$ par uporządkowanych $\langle x, y \rangle$, który z kolei jest podzbiorem iloczynu kartezjańskiego zbiorów $\{X\} \times \{Y\}$, przy czym $x \in \{X\}$ i $y \in \{Y\}$. Podzbiór $\{F\}$ ma dwie podstawowe właściwości:

- dla każdego $x \in \{X\}$ istnieje $y \in \{Y\}$ takie, że $(x, y) \in \{F\}$,
- jeśli $(x, y_1) \in \{F\}$ i $(x, y_2) \in \{F\}$, to $y_1 = y_2$.

Często zamiast terminu funkcja używa się określenia odwzorowanie zbioru $\{X\}$ w zbiór $\{Y\}$. Zmienne x i y mogą w ogólnym przypadku być zmiennymi wielowymiarowymi (wektorami). Dla opisanego w punkcie 1.2 logicznego układu kombinacyjnego o m wejściach (rys. 1.3), zbiór możliwych wartości wektora wejściowego wynosi 2^m , przy czym poszczególne składowe wektora, a więc wejścia, mogą przyjmować wartość "0" lub "1". Zbiór wektorów wejściowych jest więc iloczynem kartezjańskim m - zbiorów: $\{0, 1\} \times \dots \times \{0, 1\} = \{0, 1\}^m$. Każde wyjście może przyjmować wartość 0 lub 1. Definicję funkcji boolowskiej dostosowanej do algebry opisanej w punkcie 2.4 można więc sformułować dla każdego wyjścia binarnego w sposób przedstawiony poniżej.

Definicja 2.2. Funkcją boolowską m zmiennych binarnych (funkcją boolowską binarną) nazywamy odwzorowanie zbioru

$$\{0, 1\} \times \dots \times \{0, 1\} = \{0, 1\}^m$$

w zbiór $\{0, 1\}$. Reguły zapisu odwzorowania mogą być bardzo różne. Jedną z nich jest omówiona uprzednio tablica wartości logicznych przedstawiona na rysunku 1.4. Inną formą jest zapis w postaci zdefiniowanej niżej formuły boolowskiej umieszczonej po prawej stronie znaku równości; po lewej stronie znaku "=" znajduje się symbol reprezentujący nazwę funkcji (wyjścia). Mówimy wówczas, że funkcja jest wyrażona w postaci analitycznej.

Definicja 2.3. Formułą boolowską nazywamy wyrażenie zbudowane ze stałych $\{0, 1\}$, oznaczeń zmiennych (literałów) oraz symboli operatorów sumy logicznej, iloczynu i dopełnienia.

Reguły kolejności wykonywania działań są podobne jak w klasycznej algebrze. W pierwszej kolejności wykonuje się operacje w nawiasach, a pierwszeństwo wykonywania operacji jest następujące:

- 1 - dopełnienie (negacja),
- 2 - mnożenie logiczne,
- 3 - suma logiczna.

Operacja dopełnienia, nazywana również operacją negacji, jest wykonywana przed wszystkimi pozostałymi operacjami. Mnożenie logiczne ma pierwszeństwo przed sumowaniem, co pozwala na zredukowanie liczby nawiasów. Jeśli to nie budzi wątpliwości znak mnożenia opuszcza się, jak w zwykłej algebrze. Przykładowo, formuła zapisu funkcji boolowskiej przedstawionej w tabelicy pokazanej na rysunku 1.2 będzie mieć postać

$$Y = \bar{x}_1 \cdot \bar{x}_2 + x_1 \cdot x_2. \quad (1)$$

Zapis ten oznacza, że $Y = 1$ wtedy, gdy $x_1 = 0$ i jednocześnie $x_2 = 0$ lub gdy $x_1 = 1$ i jednocześnie $x_2 = 1$. W pozostałych przypadkach $Y = 0$.

Jeśli funkcję (1) przekształcić zgodnie z prawem rozdzielności "+" względem ".", to otrzymamy

$$\begin{aligned} Y &= (\bar{x}_1 \cdot \bar{x}_2) + x_1 \cdot x_2 = \\ &= (\bar{x}_1 \cdot \bar{x}_2 + x_1) \cdot (\bar{x}_1 \cdot \bar{x}_2 + x_2). \end{aligned} \quad (2)$$

Wykorzystując postulat o przemienności (A_1) oraz stosując twierdzenie 2.8, można ostatecznie napisać

$$Y = (x_1 + \bar{x}_2) \cdot (\bar{x}_1 + x_2). \quad (3)$$

Stojące po prawej stronie znaku równości formuły boolowskie mają różną postać, jednak opisują tę samą funkcję. Porównując postać (1) funkcji Y z tabelicą podaną na rysunku 1.2, można zauważyć, że została ona utworzona bezpośrednio z tabeli dla zsumowanych "jedynek" funkcji (00, 11) wg algorytmu

$$(00; 11) \longleftrightarrow (\bar{x}_1 \cdot \bar{x}_2 + x_1 \cdot x_2),$$

w którym symbole wartości zmiennych wejściowych zostały zastąpione nazwami zmiennych występujących jako "jedynki" i ich dopełnieniami (negacjami) w tych miejscach, gdzie występują "zera". W ten sposób uzyskano zgodność formuły boolowskiej z opisem w tabeli. Analogiczny algorytm można utworzyć dla "zer" funkcji Y (01, 10)

$$(01 ; 10) \longleftrightarrow (x_1 + \bar{x}_2) (\bar{x}_1 + x_2).$$

W tym przypadku zeru odpowiada postać afirmacyjna (niezanegowana) zmiennej, natomiast jedynce - dopełnienie (negacja) zmiennej. Operatory "+" i "." dla obu postaci są stosowane zamiennie. Ostatni wynik jest formułą boolowską otrzymaną dla postaci (3) funkcji Y .

Omówiony algorytm postępowania ma charakter ogólny i daje się stosować do dowolnej tabeli wartości logicznych, której ogólną postać przedstawiono na rysunku 1.3. Wynika to bezpośrednio z prawa łączności (tw. 2.9) spełnionego dla operatorów sumy i iloczynu, co pozwala na tworzenie wieloargumentowych (bez używania nawiasów) sum oraz iloczynów oraz twierdzeń 2.3-2.5.

Innym przykładem funkcji boolowskiej może być następujące wyrażenie

$$F = a + \bar{b} \cdot c. \quad (4)$$

Zapis ten oznacza, że $F = 1$, gdy $a = 1$, lub gdy $b = 0$ i jednocześnie $c = 1$. Formuła boolowska stojąca po prawej stronie wyrażenia (4) nie ma bezpośredniego odpowiednika w tabeli wartości logicznych funkcji F , przedstawionej na rysunku 2.5, otrzymanego w wyniku przedstawionego uprzednio postępowania algorytmicznego. Dla zer funkcji otrzymamy bowiem

a	b	c	F
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

$$F = (a + b + c) \cdot (a + \bar{b} + c) \cdot (a + \bar{b} + \bar{c}),$$

natomiast dla jedynek:

$$F = \bar{a} \cdot \bar{b} \cdot c + a \cdot \bar{b} \cdot \bar{c} + a \cdot \bar{b} \cdot c + a \cdot b \cdot \bar{c} + a \cdot b \cdot c.$$

Do postaci (4) funkcji F można jednak dojść, stosując przekształcenia wynikające z aksjomatów i udowodnionych twierdzeń, np.:

Rys. 2.5. Tablica wartości logicznych funkcji (4)

$$\begin{aligned}
 F &= (a + b + c) \cdot (a + \bar{b} + c)(a + \bar{b} + \bar{c}) = \\
 &= (a + b + c) \cdot [(a + \bar{b}) + c \cdot \bar{c}] = (a + b + c)(a + \bar{b}) = \\
 &= a + (b + c) \cdot \bar{b} = a + \bar{b} \cdot c;
 \end{aligned}$$

lub:

$$\begin{aligned}
 F &= \bar{a} \cdot \bar{b} \cdot c + a \cdot \bar{b} \cdot \bar{c} + a \cdot \bar{b} \cdot c + a \cdot b \cdot \bar{c} + a \cdot b \cdot c = \\
 &= \bar{a} \cdot \bar{b} \cdot c + a \cdot \bar{b} + a \cdot b = \bar{a} \cdot \bar{b} \cdot c + a = a + \bar{b} \cdot c.
 \end{aligned}$$

Należy zwrócić uwagę na fakt, że choć otrzymywane w poszczególnych etapach przekształceń formuły boolowskie są równoważne, to jednak najprostszą postacią funkcji F jest postać (4). Postać ta nosi nazwę postaci minimalnej funkcji w tym sensie, że do jej zapisu użyto najmniejszej liczby symboli zmiennych lub ich dopełnień. Zmienną lub jej dopełnienie (negację) określamy często jedną nazwą: literał. Postać minimalną funkcji boolowskiej można więc określić jako postać zawierającą najmniejszą liczbę literałów. Jak wynika z przykładu na ogół nie jest to postać, którą można bezpośrednio otrzymać z tablicy wartości logicznych.

Łatwo zauważyć, że istnieje nieskończenie wiele formuł boolowskich, za pomocą których można zapisać analitycznie tę samą funkcję, jednak znaczenie użytkowe mają tylko pewne formy szczególne, które można nazwać ogólnie postaciami wyróżnionymi funkcji. Jedną z nich jest wspomniana postać minimalna, drugą - postać otrzymana bezpośrednio z tablic wartości logicznych wg omówionego uprzednio algorytmu.

2.7. Postacie kanoniczne funkcji boolowskich

Jak już wspomniano, z faktu, że operatory "+" i "." spełniają prawo łączności wynika bezpośrednio możliwość tworzenia wieloliterałów, zarówno iloczynów jak i sum logicznych. Zmienne w tych wyrażeniach mogą występować bądź w postaci afirmacyjnej bądź też zanegowanej. Wygodnie jest w związku z tym wprowadzić dwie definicje.

Definicja 2.4. Pełnym iloczynem nazywamy iloczyn złożony ze wszystkich zmiennych, przy czym literał reprezentujący każdą ze zmiennych występuje tylko jeden raz.

Definicja 2.5. Pełną sumą nazywamy sumę złożoną ze wszystkich zmiennych, przy czym literał reprezentujący każdą ze zmiennych występuje tylko jeden raz.

Pełny iloczyn przyjmuje wartość 1 tylko dla jednej kombinacji wartości zmiennych. Z kolei pełna suma, tylko dla jednej kombinacji wartości zmiennych przyjmuje wartość 0. Dla n zmiennych można utworzyć 2^n różnych pełnych iloczynów lub pełnych sum. Suma wszystkich pełnych iloczynów jest zawsze równa jedności, iloczyn wszystkich pełnych sum - zeru.

indeks	$x_{n-1}, \dots, x_1, x_0$	Minterm	Symbol	Maksterm	Symbol
0	0 ... 0 0	$\bar{x}_{n-1} \cdots \bar{x}_1 \cdot \bar{x}_0$	m_0	$x_{n-1} + \cdots + x_1 + x_0$	M_0
1	0 ... 0 1	$\bar{x}_{n-1} \cdots \bar{x}_1 \cdot x_0$	m_1	$x_{n-1} + \cdots + x_1 + \bar{x}_0$	M_1
2	0 ... 1 0	$\bar{x}_{n-1} \cdots x_1 \cdot \bar{x}_0$	m_2	$x_{n-1} + \cdots + \bar{x}_1 + x_0$	M_2
3	0 ... 1 1	$\bar{x}_{n-1} \cdots x_1 \cdot x_0$	m_3	$x_{n-1} + \cdots + \bar{x}_1 + \bar{x}_0$	M_3
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2^n - 1$	1 ... 1 1	$x_{n-1} \cdots x_1 \cdot x_0$	$m_{(2^n-1)}$	$\bar{x}_{n-1} + \cdots + \bar{x}_1 + \bar{x}_0$	$M_{(2^n-1)}$

Rys. 2.6. Tablica uporządkowanych kombinacji wartości i oznaczeń zmiennych boolowskich

Wymienione własności pełnych sum i iloczynów logicznych mają charakter twierdzeń, których prawdziwość ilustruje tabela pokazana na rysunku 2.6. Pełne iloczyny zostały tam zapisane w ten sposób, że poszczególnym ciągom zero-jedynkowym reprezentującym uporządkowane względem indeksów wartości zmiennych x_i ($i = 0, \dots, n-1$), są przypisane tak samo uporządkowane iloczyny tych zmiennych, przy czym jeśli pozycji o i -tym indeksie odpowiada wartość 1, zmienna występuje w postaci afirmacyjnej, jeśli 0 - w postaci dopełnienia (negacji). Przy takim uporządkowaniu ciągi zero-jedynkowe reprezentują takie wartości poszczególnych argumentów pełnego iloczynu, dla których przyjmuje on wartość 1. Z właściwości operatora iloczynu (rys. 2.1) wynika, że dla każdego innego ciągu, wartość iloczynu wynosi 0. Uporządkowany w sposób wyżej wymieniony pełny iloczyn nosi nazwę mintermu. Do oznaczenia

mintermu stosuje się małą literę m z indeksem reprezentującym odpowiednik dziesiętny, liczby binarnej utworzonej przez uporządkowany ciąg zer i jedynek reprezentujących wartości poszczególnych zmiennych.

W sposób analogiczny zostały uporządkowane pełne sumy. Występuje tutaj jednak różnica w przyporządkowaniu wartościom poszczególnych zmiennych, postaci zmiennych w pełnej sumie: wartości 0 jest przyporządkowana afirmacja zmiennej, wartości 1 - jej dopełnienie. W konsekwencji ciągi zero-jedynkowe reprezentują takie wartości poszczególnych argumentów pełnej sumy, dla których przyjmuje ona wartość 0. Z właściwości operatora sumy (rys. 2.1) wynika, że dla każdego innego ciągu, wartość sumy wynosi 1. Uporządkowana w wyżej wymieniony sposób pełna suma nosi nazwę makstermu. Do oznaczenia makstermu stosuje się dużą literę M z indeksem ustalonym analogicznie jak dla odpowiedniego mintermu. Reasumując powyższe rozważania można sformułować kolejne definicje.

Definicja 2.6. Mintermem nazywamy pełny iloczyn (def. 2.4) uporządkowany w sposób przedstawiony na rysunku 2.6.

Definicja 2.7. Makstermem nazywamy pełną sumę (def. 2.5) uporządkowaną w sposób przedstawiony na rysunku 2.6.

Bezpośrednio z definicji funkcji boolowskiej binarnej (def. 2.2) oraz z twierdzeń 2.3-2.5 wynikają poniższe twierdzenia.

Twierdzenie 2.11. Każdą funkcję boolowską binarną można przedstawić w postaci sumy mintermów

$$f = \sum_i m_i,$$

przy czym wskaźnik i dotyczy wszystkich tych i tylko tych mintermów, dla których wartość funkcji równa się 1.

Twierdzenie 2.12. Każdą funkcję boolowską binarną można przedstawić w postaci iloczynu makstermów

$$f = \prod_i M_i,$$

przy czym wskaźnik i dotyczy wszystkich tych i tylko tych makstermów, dla których wartość funkcji równa się 0.

W obydwu twierdzeniach posłużono się symbolami sumy $\sum$ i iloczynu $\prod$ zapożyczonymi ze zwykłej algebry. Twierdzenia te stanowią bardziej formalne ujęcie rezultatu uzyskanego w sposób heurystyczny w punkcie 2.6 przy omawianiu sposobów zapisu odwzorowań leżących u podstaw definicji 2.2. Postacie funkcji określone w twierdzeniach 2.11 i 2.12 noszą nazwę postaci kanonicznych funkcji boolowskich.

Przykład 2.2. Funkcja boolowska

$$F = x y + \bar{x} z, \quad (1)$$

równa się jedności wtedy, gdy $x = 1$ i jednocześnie $y = 1$ lub wtedy, gdy $x = 0$ i jednocześnie $z = 1$. W pozostałych przypadkach równa się

ona 0. Funkcja ta nie jest dana w postaci kanonicznej, gdyż nie jest sumą pełnych iloczynów. Sprowadzić ją można do postaci kanonicznej za pomocą tablicy wartości logicznych bądź też za pomocą manipulacji algebraicznych. Tablica funkcji f jest przedstawiona na rysunku 2.7. Z tablicy tej bezpośrednio wynika, że

	x	y	z	f
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1

$$f = m_1 + m_3 + m_6 + m_7 = \\ + \bar{x} \bar{y} z + \bar{x} y z + x y \bar{z} + x y z, \quad (2)$$

bądź też

$$f = M_0 \cdot M_2 \cdot M_4 \cdot M_5 = \\ = (x + y + z)(x + \bar{y} + z)(\bar{x} + y + z)(\bar{x} + y + \bar{z}). \quad (3)$$

Rys. 2.7. Tablica wartości logicznych funkcji $f = xy + \bar{x}z$

Regułą porządkową jest tutaj kolejność alfabetyczna liter użytych do oznaczania zmiennych. Jak już powiedziano postać kanoniczną można otrzymać również w wyniku manipulacji algebraicznych przeprowadzonych w oparciu o udowodnione twierdzenia i reguły wynikające z aksjomatów, na przykład

$$f = x y + \bar{x} z = x y (z + \bar{z}) + \bar{x} z (y + \bar{y}) = \\ = xyz + xy\bar{z} + \bar{x}yz + \bar{x}\bar{y}z.$$

Po uporządkowaniu kolejności mintermów uzyskujemy postać (2) funkcji. Aby uzyskać postać (3) iloczynu sum, korzystając z prawa rozdzielczości, przekształćmy funkcję (1) nieco inaczej:

$$\begin{aligned}
f &= x y + \bar{x} z = (x y + \bar{x})(x y + z) = \\
&= (\bar{x} + y)(x + z)(y + z) = \\
&= (\bar{x} + y + z \bar{z})(x + z + y \bar{y})(y + z + x \bar{x}) = \\
&= (\bar{x} + y + z)(\bar{x} + y + \bar{z})(x + y + z)(x + \bar{y} + z)(x + y + z)(\bar{x} + y + z) = \\
&= (x + y + z)(x + \bar{y} + z)(\bar{x} + y + z)(\bar{x} + y + \bar{z}).
\end{aligned}$$

Można zauważyć, że jeśli znamy nazwy zmiennych oraz regułę porządkową, to zapis postaci kanonicznych można jeszcze bardziej uprościć, poprzez ograniczenie się do wyszczególnienia jedynie indeksów występujących mintermów bądź makstermów. Zapis będzie miał wtedy postać:

$$f(x_{n-1}, \dots, x_0) = \sum (i) \quad (4)$$

lub

$$f(x_{n-1}, \dots, x_0) = \prod (j), \quad (5)$$

gdzie: i, j - reprezentuje indeksy odpowiednio mintermów bądź makstermów opisujących funkcję.

Postacie kanoniczne funkcji podanej w przykładzie 2.2 będą wówczas następujące:

$$f(x, y, z) = \sum (1, 3, 6, 7) \quad (6)$$

lub

$$f(x, y, z) = \prod (0, 2, 4, 5). \quad (7)$$

Jeśli nazwy zmiennych nie są istotne, to postacie (4) i (5) można jeszcze bardziej uprościć, podając w nawiasie za nazwą funkcji jedynie liczbę zmiennych n :

$$f(n) = \sum (j), \quad (8)$$

$$f(n) = \prod (j). \quad (9)$$

Zapis (4), (5) lub (8) i (9) pozwala na proste przejście od jednej postaci kanonicznej do drugiej. Wystarczy bowiem zauważyć, że indeksy występujące w jednej postaci nie występują w drugiej, przy czym z defi-

nicji funkcji wynika, że łącznie w obu postaciach musi wystąpić 2^n indeksów. Ilustracją powyższych wniosków są postacie (6) i (7) funkcji omawianej w przykładzie 2.2.

2.8. Postać normalna funkcji boolowskiej i jej dopełnienie

Definicja 2.8. Postacie, w których formuła boolowska określająca funkcję jest wyłącznie sumą iloczynów zmiennych bądź też iloczynem ich sum jest nazywana postacią normalną funkcji. Postacie kanoniczne, jak również postacie przytoczone w dotychczasowych przykładach, są postaciami normalnymi funkcji. Nie jest postacią normalną następujący zapis funkcji

$$F = a \cdot c + b \cdot (c + \bar{d}),$$

bowiem występuje tu suma, zarówno iloczynu zmiennych a i c jak i iloczynu zmiennej b i sumy $(c + \bar{d})$.

Postać normalna iloczynu sum jest określona jako koniunkcyjna postać normalna, np. funkcja

$$f = (a + \bar{b})(c + d) \bar{a}$$

jest zapisana w koniunkcyjnej postaci normalnej (KPN).

Postać normalna sumy iloczynów jest określona jako alternatywna lub niekiedy - dysjunkcyjna postać normalna, np. funkcja

$$f = a b c + a \bar{c} + b$$

jest zapisana w alternatywnej (dysjunkcyjnej) postaci normalnej (APN).

Wartości funkcji boolowskiej są elementami zbioru, na którym została określona algebra. Zgodnie z aksjomatem A_4 definicji 2.1 musi istnieć dopełnienie każdej wartości funkcji, a więc również funkcja, którą można nazwać dopełnieniem funkcji pierwotnej.

Dopełnienie funkcji w sposób ogólny można znaleźć posługując się rozwinięciem praw de Morgana (tw. 2.10) na większą liczbę zmiennych. Można to zrobić formułując poniższe twierdzenia.

Twierdzenie 2.13. Dopełnienie iloczynu wielu zmiennych jest równe sumie dopełnień poszczególnych literałów

$$\overline{(x_{n-1} \cdots x_1 x_0)} = \bar{x}_{n-1} + \bar{x}_{n-2} + \cdots + \bar{x}_1 + \bar{x}_0.$$

Dowód:

$$\overline{(x_{n-1} \dots x_1 x_0)} = \overline{(x_{n-1} \dots x_1)} x_0 = \quad (\text{Tw. 2.9})$$

$$= \overline{x_{n-1} \dots x_2 x_1} + \bar{x}_0 = \overline{(x_{n-1} \dots x_2)} x_1 + \bar{x}_0 = \quad (\text{Tw. 2.10})$$

$$= \overline{x_{n-1} \dots x_2} + \bar{x}_1 + \bar{x}_0 = \quad (\text{jw. + relacje równoważne})$$

$$= \dots = \bar{x}_{n-1} + \dots + \bar{x}_1 + \bar{x}_0.$$

Twierdzenie 2.14. Dopełnienie sumy wielu zmiennych jest równe iloczynowi dopełnień poszczególnych literałów

$$\overline{(x_{n-1} + \dots + x_1 + x_0)} = \bar{x}_{n-1} \dots \bar{x}_1 \bar{x}_0.$$

Dowód: na zasadzie dualności względem twierdzenia 2.13.

Wykorzystując powyższe twierdzenie można w wyniku manipulacji algebraicznych otrzymać analityczną postać funkcji będącej dopełnieniem funkcji pierwotnej. Jeśli funkcja dana jest w postaci normalnej, przekształcenie za pomocą uogólnionych praw de Morgana jest dokonywane w sposób szczególnie prosty.

Przykład 2.3. Wykorzystując uogólnione twierdzenie de Morgana, wyznaczyć dopełnienie dla postaci (1) funkcji z punktu 2.7 oraz jej postaci kanonicznych (2) i (3). Otrzymamy

$$\bar{f} = \overline{(xy + \bar{x}z)} = \overline{(xy)} \overline{(\bar{x}z)} = (\bar{x} + \bar{y}) \cdot (x + \bar{z}).$$

Dla postaci kanonicznej sumy:

$$\begin{aligned} \bar{f} &= \overline{(m_1 + m_3 + m_6 + m_7)} = \bar{m}_1 \bar{m}_3 \bar{m}_6 \bar{m}_7 = \\ &= (x + y + \bar{z}) \cdot (x + \bar{y} + \bar{z}) \cdot (\bar{x} + \bar{y} + z) \cdot (\bar{x} + \bar{y} + \bar{z}) = \\ &= M_1 M_3 M_6 M_7 = \prod (1, 3, 6, 7). \end{aligned}$$

dla postaci kanonicznej iloczynu:

$$\begin{aligned} \bar{f} &= \overline{(M_0 M_2 M_4 M_5)} = \bar{M}_0 + \bar{M}_2 + \bar{M}_4 + \bar{M}_5 = \bar{x} \bar{y} \bar{z} + \bar{x} y \bar{z} + \\ &+ x \bar{y} \bar{z} + x \bar{y} z = m_0 + m_2 + m_4 + m_5 = \sum (0, 2, 4, 5). \end{aligned}$$

Porównując rezultaty otrzymane dla postaci kanonicznych z wyrażeniami (2) i (3) z punktu 2.7 można zauważyć, że postać kanoniczną sumy mintermów przechodzi w postać iloczynu makstermów o tych samych indeksach. Rezultat ten jest naturalną konsekwencją uogólnionych twierdzeń de Morgana bowiem jak wynika z tabeli pokazanej na rysunku 2.6, mintermy są dopełnieniami odpowiednich makstermów i odwrotnie, w związku z czym zgodnie z (4) można napisać

$$\bar{f}(x_{n-1}, \dots, x_0) = \overline{\sum (i)} = \prod (i). \quad (1)$$

Podobnie dla postaci kanonicznej iloczynu sum, funkcję będącą dopełnieniem otrzymujemy zmieniając iloczyn makstermów na sumę mintermów o tych samych indeksach; można to zgodnie z (5) symbolicznie zapisać

$$\bar{f}(x_{n-1}, \dots, x_0) = \overline{\prod (j)} = \sum (j). \quad (2)$$

Przykład 2.4. Dana funkcja $f = (w, x, y, z) = \sum (0, 2, 3, 8, 10)$. Określmy obie postacie kanoniczne funkcji będącej jej dopełnieniem. Kanoniczna postać koniunkcyjna składa się z iloczynów makstermów o indeksach uzupełniających indeksy mintermów, czyli

$$f(w, x, y, z) = \prod (1, 4, 5, 6, 7, 9, 11, 12, 13, 14, 15).$$

Najwyższa wartość indeksu wynosi $2^n - 1 = 15$, gdyż $n = 4$. Stąd dla funkcji będącej dopełnieniem f otrzymamy

$$\bar{f}(w, x, y, z) = \prod (0, 2, 3, 8, 10),$$

oraz

$$\bar{f}(w, x, y, z) = \sum (1, 4, 5, 6, 7, 9, 11, 12, 13, 14, 15).$$

Jeśli funkcja dana jest w postaci tablicy wartości logicznych, to zgodnie z definicją (rys. 2.1c), jej dopełnienie uzyskujemy zamieniając zera funkcji na jedynki i jedynki na zera. Ilustruje to przykład podany na rysunku 2.8. Bezpośrednio z tablicy można wypisać dowolne postacie kanoniczne funkcji lub ich dopełnienia:

$$F = \bar{x} \bar{y} z + x \bar{y} \bar{z} + x \bar{y} z + x y z = \sum (1, 4, 5, 7),$$

$$\bar{F} = \bar{x} \bar{y} \bar{z} + \bar{x} y \bar{z} + \bar{x} y z + x y \bar{z} = \sum (0, 2, 3, 6)$$

lub:

$$\begin{aligned}
 F &= (x + y + z)(x + \bar{y} + z) \\
 &\quad (x + \bar{y} + \bar{z})(\bar{x} + \bar{y} + z) = \\
 &= \prod (0, 2, 3, 6), \\
 \bar{F} &= (x + y + \bar{z})(\bar{x} + y + z) \\
 &\quad (\bar{x} + y + \bar{z})(\bar{x} + \bar{y} + \bar{z}) = \\
 &= \prod (1, 4, 5, 7).
 \end{aligned}$$

Gdy funkcja dana jest w postaci innej niż normalna, to dla znalezienia dopełnienia można ją przekształcić do postaci normalnej i dalej zastosować zilustrowaną już procedurę, bądź też przekształcać ją bezpośrednio stosując prawa de Morgana.

	x	y	z	F	$\bar{F}$
0	0	0	0	0	1
1	0	0	1	1	0
2	0	1	0	0	1
3	0	1	1	0	1
4	1	0	0	1	0
5	1	0	1	1	0
6	1	1	0	0	1
7	1	1	1	1	0

Rys. 2.8. Tablica wartości logicznych funkcji F i jej dopełnienia $\bar{F}$

Przykład 2.5. Znaleźć za pomocą manipulacji algebraicznych dopełnienie następującej funkcji

$$f = a c + b(c + \bar{d}).$$

Stosując bezpośrednio prawa de Morgana, otrzymamy

$$\begin{aligned}
 \bar{f} &= \overline{[a c + b(c + \bar{d})]} = \overline{[a c]} \cdot \overline{[b(c + \bar{d})]} = \\
 &= (\bar{a} + \bar{c}) \cdot [\bar{b} + \overline{(c + \bar{d})}] = \\
 &= (\bar{a} + \bar{c}) \cdot (\bar{b} + \bar{c} \cdot d).
 \end{aligned}$$

2.9. Funkcje boolowskie dwóch zmiennych

Dwuwymiarowy wektor zmiennych wejściowych binarnych może przyjmować cztery różne wartości

$$X = \{00, 01, 10, 11\}.$$

Wartościom tym na wyjściu można przypisać wartości ze zbioru $\{0, 1\}$ na $2^4 = 16$ sposobów (ogólnie na 2^{2^n} sposobów). Innymi słowy, dla dwóch

argumentów można zdefiniować 16 różnych funkcji boolowskich. Funkcje te przedstawiono w tabeli wartości logicznych pokazanej na rysunku 2.9. Niektóre z nich mają charakter operatorów dwuargumentowych, inne jednoargumentowych oraz stałych. Wymieńmy je wg kolejności wzrastania indeksów:

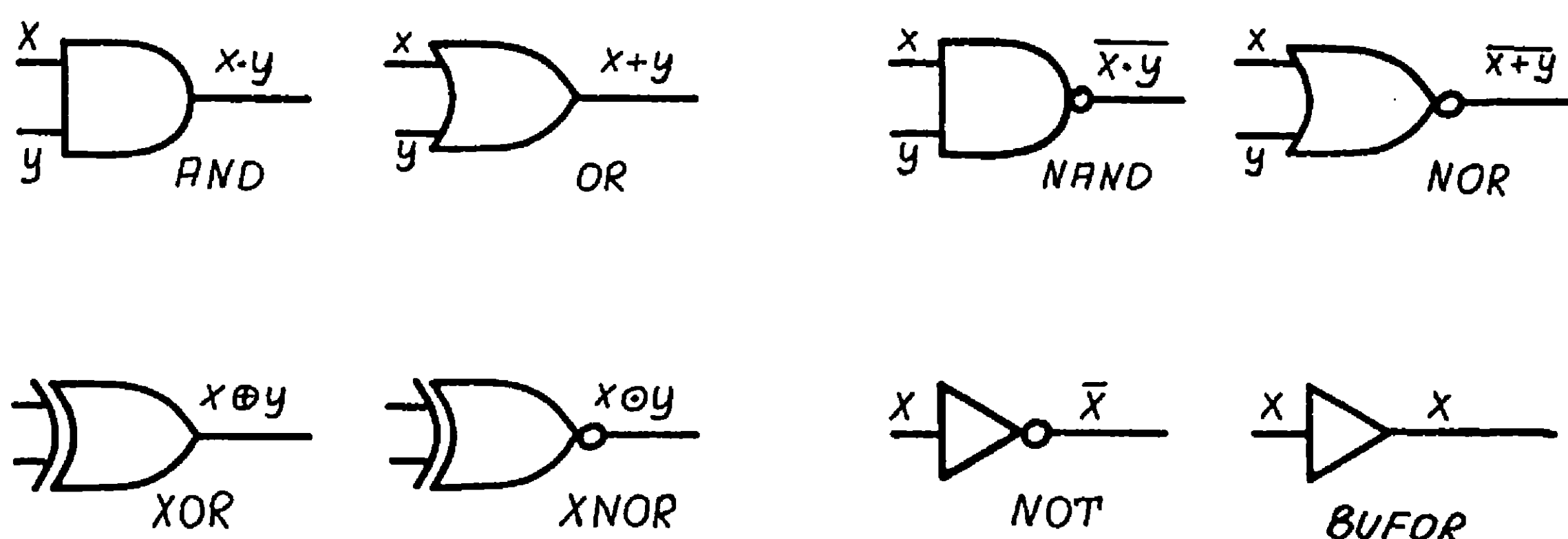
- f_0 - stała: $f_0 = 0$
- f_1 - iloczyn logiczny (koniunkcja) : $f_1 = x \cdot y$ (lub $f_1 = x \wedge y$),
- f_2 - funkcja zakazu: $f_2 = x \cdot \bar{y} = x / y$,
- f_3 - zmienna x : $f_3 = x$,
- f_4 - funkcja zakazu: $f_4 = \bar{x} \cdot y = y / x$,
- f_5 - zmienna y : $f_5 = y$,
- f_6 - operator ALBO (XOR): $f_6 = \bar{x} \cdot y + x \cdot \bar{y} = x \oplus y$,
- f_7 - suma logiczna (alternatywa) : $f_7 = x + y$ (lub $f_7 = x \vee y$),
- f_8 - negacja sumy NOR: $f_8 = \bar{x} \cdot \bar{y} = \overline{(x + y)} = x \downarrow y$,
- f_9 - równoważność (XNOR): $f_9 = \bar{x} \cdot \bar{y} + x \cdot y = x \odot y$,
- f_{10} - negacja y : $f_{10} = \bar{y}$,
- f_{11} - implikacja: $f_{11} = x + \bar{y} = x \leftarrow y$,
- f_{12} - negacja x : $f_{12} = \bar{x}$,
- f_{13} - implikacja: $f_{13} = \bar{x} + y = x \rightarrow y$,
- f_{14} - negacja iloczynu NAND: $f_{14} = \overline{x \cdot y} = x \uparrow y$,
- f_{15} - stała 1 : $f_{15} = 1$.

$x \ y$	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0 0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0 1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1 0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1 1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Symbol lub operacja	0	$\cdot$	$/$	x	$/$	y	$\oplus$	$+$	$\downarrow$	$\odot$	$\bar{y}$	$\leftarrow$	$\bar{x}$	$\rightarrow$	$\uparrow$	1

Rys. 2.9. Tabela wartości logicznych funkcji dwuargumentowych

Nazwy funkcji pochodzą od określeń stosowanych w rachunku zdań, gdzie operator sumy jest nazwany alternatywą zmiennych zdaniowych, iloczyn koniunkcją, a dopełnienie - negacją. Stałą logiczną 1 określa się zwykle jako "prawda", a stałą 0 jako "fałsz". Dla takich określeń zdefiniowana w punkcie 2.4 algebra Boole'a staje się algebrą rachunku zdań. Istotną rolę w procesie dowodzenia twierdzeń wielu teorii sformalizowanych (i nie tylko) odgrywa implikacja, określana dla funkcji f_{11} jako $x \rightarrow y$, a dla funkcji f_{13} jako $x \leftarrow y$. Dla f_{11} (f_{13}) implikacja jest fałszem (0) tylko w jednym przypadku: gdy $y(x)$ jest zdaniem prawdziwym (1), a $x(y)$ fałszywym (0). Implikację f_{11} określa spójnik zdaniowy: jeżeli y to x ; implikację f_{13} : jeżeli x to y . Dopełnieniem implikacji jest funkcja zakazu f_2 lub f_4 . W odróżnieniu od rachunku zdań, implikacja w teorii układów logicznych nie odgrywa większej roli. To samo dotyczy operacji zakazu, natomiast istotne są pozostałe operatory dwuargumentowe, a więc obok omawianych już iloczynu i sumy logicznej operator negacji iloczynu, negacji sumy, operator ALBO i równoważności.

Operator negacji iloczynu jest najczęściej oznaczany symbolem NAND (ang. Not-AND); natomiast operator negacji sumy NOR (ang. Not-OR). Operator ALBO (f_6) jest podobny do operatora sumy, z wyjątkiem przypadku $x = y = 1$. Stąd jego nazwa angielska: EXCLUSIVE-OR lub w skrócie: XOR. Czasem operator ten określa się jako nierównoważność, różnicę symetryczną, alternatywę z wykluczeniem lub sumę modulo 2. Operator równoważności jest dopełnieniem operatora XOR i przyjmuje wartość 1 tylko wtedy, gdy oba argumenty są sobie równe.



Rys. 2.10. Symbole graficzne i nazwy operatorów dwu i jednoargumentowych

Na rysunku 2.10 przedstawiono najczęściej stosowane symbole graficzne i nazwy reprezentujące operatory istotne z punktu widzenia układów logicznych. Symbole te przyjęto tradycyjnie nazywać bramkami logicznymi, z uwagi na dwustanowy (włącz-wyłącz) charakter pracy urządzeń opisywanych przez układy logiczne. Nazwy bramek iloczynu i sumy logicznej zaczerpnięto z angielskiego nazewnictwa spójników zdaniowych: i - AND, lub - OR. Bramkę dopełnienia nazywa się najczęściej bramką negacji i stąd określenie NOT. Bramkę negacji nazywa się również inwerterem. Na rysunku zaznaczono bramkę bufora, który nie zmienia wartości logicznej zmiennej boolowskiej, ale wzmacnia sygnał fizyczny, który ona reprezentuje.

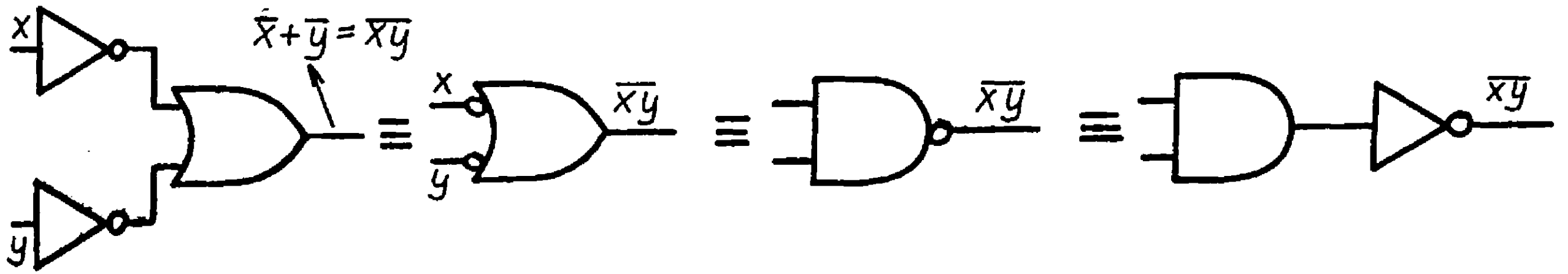
2.10. Systemy funkcjonalnie pełne

Z tabeli pokazanej na rysunku 2.9 wynika, że poza operatorami dwuargumentowymi wymienionymi w definicji 2.1 algebry Boole'a, istnieją również inne operatory dwuargumentowe. Powstaje pytanie czy można dla różnych ich zestawów tworzyć systemy funkcjonalnie pełne, tzn. takie, które pozwolą tworzyć poprawne formuły boolowskie, zawierające obok nazw zmiennych i elementów 0,1, symbole odpowiednio dobranych operatorów. Przykładami systemów funkcjonalnie pełnych są, obok podanych w definicji 2.1, systemy utworzone przez następujące operatory:

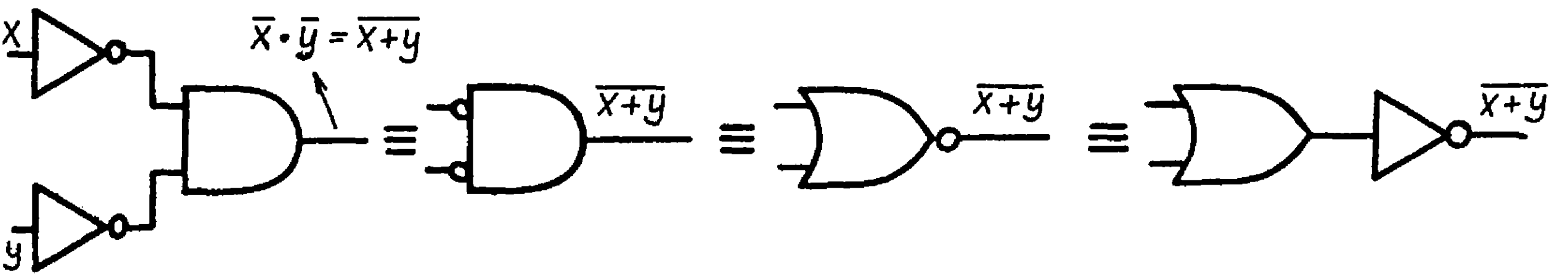
- | | |
|-------------------------|---------------------------------------|
| a) NAND; | g) ALBO (EXCLUSIVE-OR), AND, stała 1; |
| b) NOR; | h) (EXCLUSIVE-OR), OR, stała 1; |
| c) implikacja, stała 0; | i) równoważność, AND, stała 0; |
| d) Implikacja, NOT; | j) równoważność, OR, stała 0; |
| e) zakaz, stała 1; | k) AND, NOT; |
| f) zakaz, NOT; | l) OR, NOT. |

Istnieją metody formalne pozwalające określać systemy funkcjonalnie pełne (np. [3,5]) oraz metody pozwalające stwierdzić czy mamy z takim systemem do czynienia. Najbardziej naturalnym testem jest sprawdzenie, czy w oparciu o dane operatory daje się skonstruować operatory wymienione w definicji 2.1, a więc: AND, OR, NOT. W przypadku a) i b) otrzymamy, stosując symbole bramek (rys. 2.10 i rys. 2.11), schematy pokazane na rysunku 2.12. Wynika z nich, że bramki NAND oraz NOR tworzą samodzielnie systemy funkcjonalnie pełne, gdyż operatory AND, OR, NOT dają się składać tylko z bramek NAND lub tylko z bramek NOR. W podobny sposób można przeprowadzić testy dla pozostałych przypadków.

a)



b)

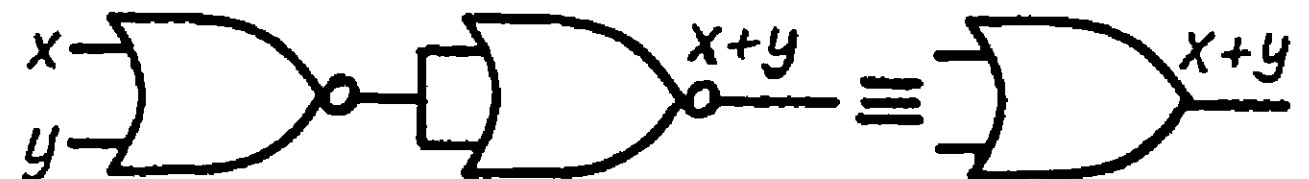
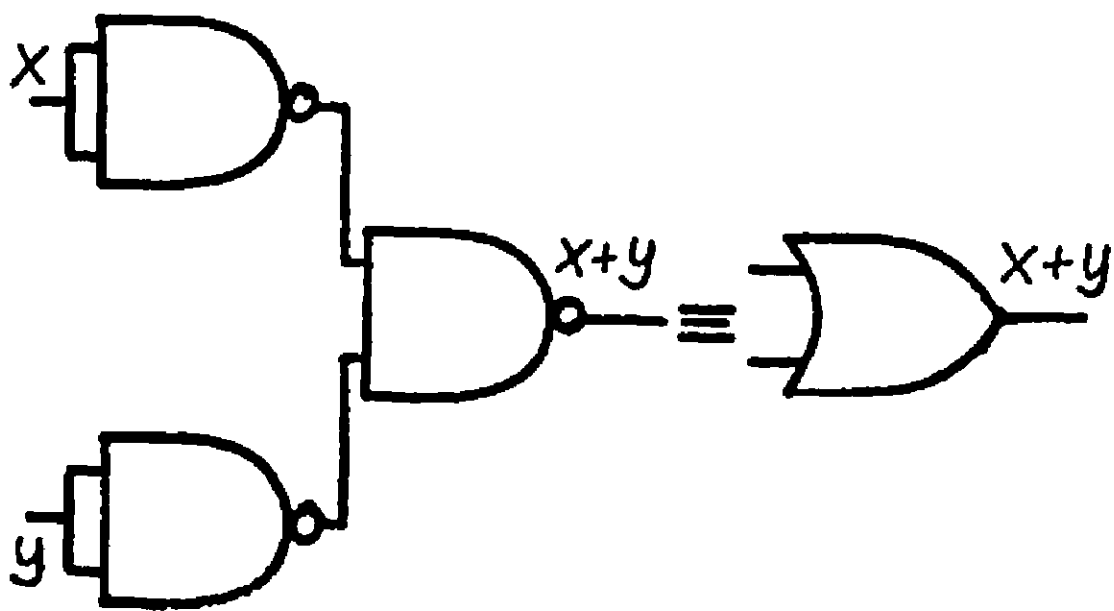
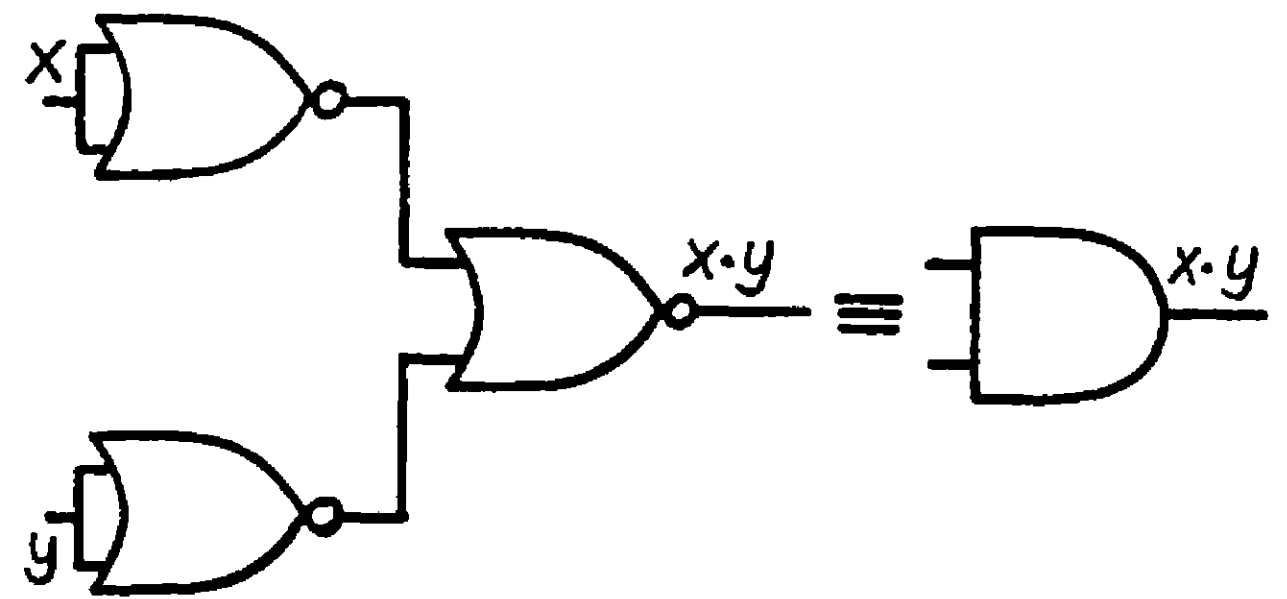
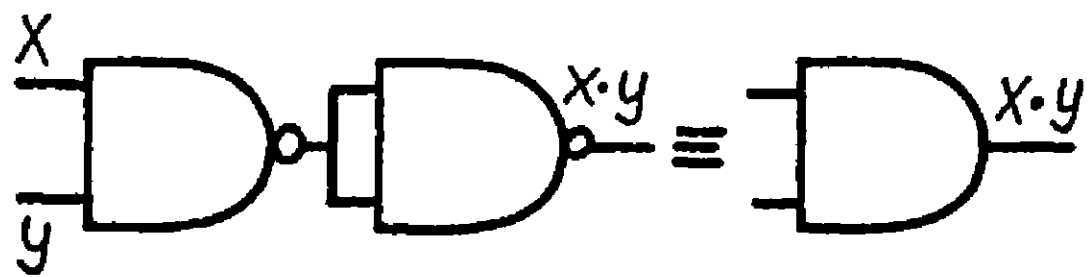


Rys. 2.11. Schematy operacyjnie równoważne dla bramek NAND oraz NOR

a)



b)

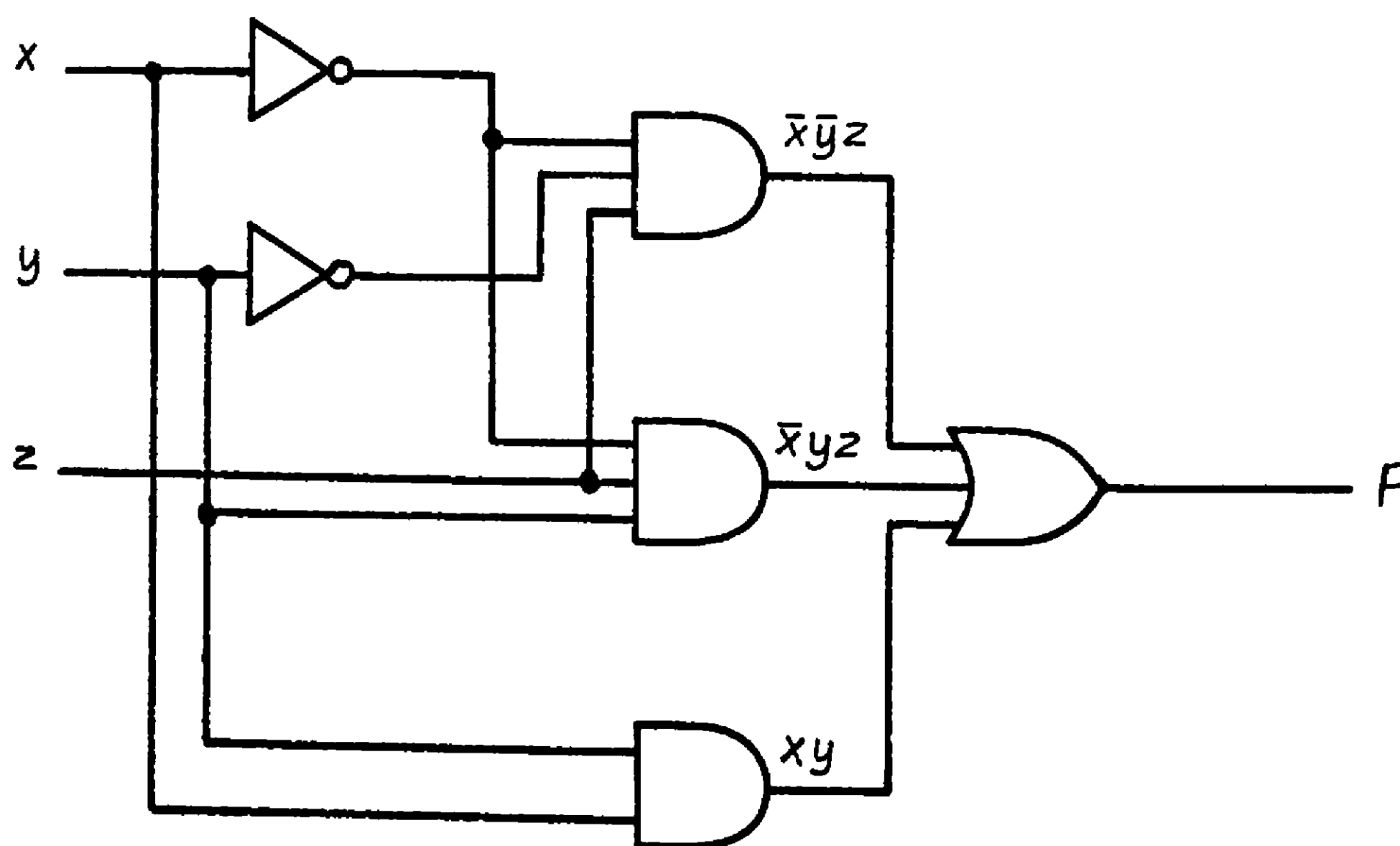


Rys. 2.12. Testy zamkniętości systemów NAND (a) oraz NOR (b)

2.11. Realizacje bramkowe funkcji logicznych

Symbole graficzne operacji logicznych pokazane na rysunkach 2.10 i 2.11 mogą być wykorzystywane do graficznego przedstawiania formuł boolowskich reprezentujących określone funkcje. Operatory łączne i przemienne, tzn. takie, które spełniają prawo łączności i przemienności mogą być reprezentowane bezpośrednio przez bramki wieloweściowe. Należą do nich zarówno operatory sumy, jak i iloczynu logicznego. Przykładowo na rys. 2.13 przedstawiono schemat logiczny funkcji

$$F = \bar{x} \cdot \bar{y} \cdot z + \bar{x} \cdot y \cdot z + x \cdot y. \quad (1)$$



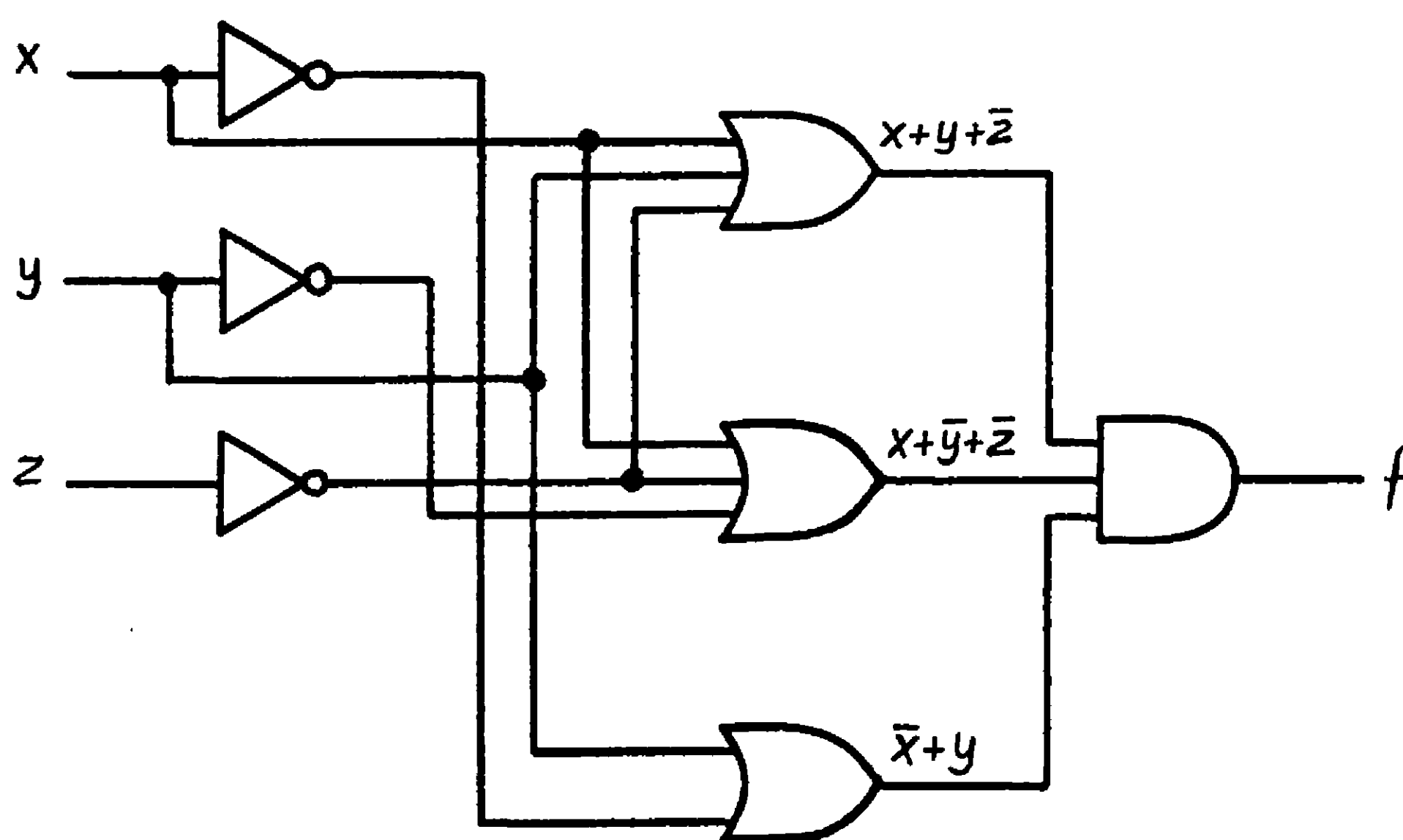
Rys. 2.13. Schemat logiczny funkcji $F = \bar{x} \cdot \bar{y} \cdot z + \bar{x} \cdot y \cdot z + x \cdot y$

Można tu wyróżnić trzy poziomy bramek: na poziomie pierwszym jest trójwejściowa bramka sumy, na poziomie drugim są bramki iloczynu: dwie trójwejściowe i jedna dwuwejściowa. Sygnały wyjściowe tych bramek są bezpośrednio podawane na wejście bramki poziomu pierwszego. Poziom trzeci składa się z jednowejściowych bramek negacji, które mogą wystąpić lub nie - zależnie od postaci realizowanej funkcji. Układ logiczny reprezentuje model urządzenia, przy czym najczęściej (szczególnie w elektronice) poszczególne bramki mają bezpośrednie odwzorowanie w blokach realizowanych fizycznie. W tej sytuacji sygnał fizyczny reprezentowany przez zmienne jest w skończonym czasie propagowany poprzez poszczególne poziomy do wyjścia poziomu pierwszego. Im mniej tych poziomów, tym szybsza propagacja, co zwykle jest bardzo pożądane.

Jak wynika z definicji 2.8 (pkt. 2.8), każda postać normalna funkcji boolowskiej daje realizację co najwyżej trójpoziomową, przy czym na poziomie trzecim są tylko inwertery (bramki negacji), na drugim - bramki iloczynu, jeśli na poziomie pierwszym jest bramka sumy, bądź też bramki sumy, jeśli na poziomie pierwszym jest bramka iloczynu. Funkcja (1) jest przykładem alternatywnej postaci normalnej (APN), natomiast na rysunku 2.14 przedstawiono schemat logiczny funkcji

$$f = (x + y + \bar{z})(x + \bar{y} + \bar{z})(\bar{x} + y), \quad (2)$$

danej w koniunkcyjnej postaci normalnej (KPN).



Rys. 2.14. Schemat logiczny funkcji $F = (x + y + \bar{z})(x + \bar{y} + \bar{z})(\bar{x} + y)$

Wśród wymienionych w punkcie 2.10 systemów funkcjonalnie pełnych na szczególną uwagę zasługują systemy jednorodne, złożone tylko z bramek NAND lub NOR. Operatory te jednak są wprowadzicie przemienne (wynika to z symetrii względem x, y tablicy wartości logicznych funkcji f_8 i f_{14}), lecz nie są łączne, tzn.:

$$x \uparrow (y \uparrow z) \neq (x \uparrow y) \uparrow z,$$

$$x \downarrow (y \downarrow z) \neq (x \downarrow y) \downarrow z,$$

bowiem

$$\overline{x \cdot (y \cdot z)} = \bar{x} + \overline{(y \cdot z)} = \bar{x} + y \cdot z,$$

$$\overline{(x \cdot y) \cdot z} = \overline{(x \cdot y)} + \bar{z} = x \cdot y + \bar{z}.$$

i analogicznie:

$$\overline{x + \overline{(y + z)}} = \bar{x} \overline{\overline{(y + z)}} = \bar{x} (y + z)$$

$$\overline{\overline{(x + y)} + z} = \overline{\overline{(x + y)}} \cdot \bar{z} = (x + y) \bar{z}.$$

Nie daje się więc bezpośrednio z definicji zrealizować trój i więcej wejściowych bramek NAND oraz NOR. Wady tej nie posiadają systemy k) i l) podane w punkcie 2.10, tzn. AND-NOT oraz OR-NOT, z uwagi na omówione już właściwości operatorów sumy i iloczynu logicznego. Jednocześnie systemy te dla dwóch argumentów utożsamiają się z operatorami NAND i NOR (rys. 2.11); przyjęto więc tej nazwy używać zarówno dla wielowejściowej negacji iloczynu

AND-NOT $\longrightarrow$ NAND,

jak i negacji sumy

OR-NOT $\longrightarrow$ NOR.

Trój i więcej wejściowe bramki NAND oraz NOR są więc realizacją odpowiednio negacji iloczynu bądź negacji sumy, a nie wielowejściową realizacją operatorów NAND czy też NOR zdefiniowanych w punkcie 2.9.

Operatory zakazu oraz implikacji, tzn. funkcje f_2 , f_4 , f_{11} , f_{13} , podane w tabeli pokazanej na rysunku 2.9, nie są przemienne, w związku z czym nie znajdują zastosowania w układach logicznych i nie konstruuje się ich odpowiedników fizycznych. Powszechne zastosowanie natomiast, szczególnie w układach cyfrowych, mają bramki EXCLUSIVE-OR (XOR) oraz ich negacje XNOR. Przemienność tych operatorów wynika z symetrii tablic wartości logicznych względem argumentów (rys. 2.9), natomiast weryfikację prawa łączności dla bramek XOR przedstawiono w tabeli wartości logicznych pokazanej na rysunku 2.15. Tę formę weryfikacji można określić jako dowód poprzez wyliczenie. Dla dwuelementowej algebry Boole'a rozmiary tablic wypadają niezbyt duże i jest to forma często stosowana. Z uwagi na jednoargumentowy charakter operacji dopełnienia, nie ma potrzeby przeprowadzania podobnego dowodu dla operatora XNOR, gdyż spełnienie prawa łączności przez funkcję pociąga za sobą spełnienie go także dla dopełnienia tej funkcji.

Dla dwóch argumentów funkcja EXCLUSIVE-OR (XOR) przyjmuje wartość 1, jeśli jeden z nich przyjmuje wartość 1. Z tablicy pokazanej na rysunku 2.15 wynika, że dla trzech argumentów funkcja XOR = 1, gdy jeden lub trzy argumenty przyjmują wartość 1. Rozwijając tę prawidłowość

rekurencyjnie, dochodzimy do wniosku, że wieloargumentowa funkcja XOR równa się jedności, jeśli nieparzysta liczba argumentów jest równa jedności

$$x_{n-1} \oplus x_{n-2} \oplus \dots \oplus x_0 = 1, \quad (3)$$

gdy

$$\sum_{i=0}^{n-1} x_i = k, \quad \text{gdzie: } k = 1, 3, 5, \dots \leq n-1.$$

a b c	$(a \oplus b)$	$(a \oplus b) \oplus c$	$(b \oplus c)$	$a \oplus (b \oplus c)$
0 0 0	0	0	0	0
0 0 1	0	1	1	1
0 1 0	1	1	1	1
0 1 1	1	0	0	0
1 0 0	1	1	0	1
1 0 1	1	0	1	0
1 1 0	0	0	1	0
1 1 1	0	1	0	1

Rys. 2.15. Weryfikacja prawa łączności dla operatora XOR

W wyniku analogicznego rozumowania dochodzimy do wniosku, że funkcja równoważności XNOR = 1, gdy parzysta liczba argumentów równa się zeru; tzn.

$$x_{n-1} \odot x_{n-2} \odot \dots \odot x_0 = 1. \quad (4)$$

gdy

$$\sum_{i=0}^{n-1} \bar{x}_i = l, \quad \text{gdzie } l = 0, 2, 4, \dots \leq n-1.$$

W zapisie wyrażeń (3) i (4) zrezygnowano z nawiasów z uwagi na dowolność wyboru kolejnych argumentów dla dokonania między nimi operacji $\oplus$ lub $\odot$. Jednocześnie z wyrażeń tych wynika, że gdy liczba argumentów n jest nieparzysta, to funkcje XOR i XNOR są sobie równoważne, gdyż jeśli parzysta liczba argumentów przyjmie wartość 0, to jednocześnie nieparzysta ich liczba musi być równa 1. Na przykład dla trzech zmiennych mamy

$$x_2 \oplus x_1 \oplus x_0 = x_2 \odot x_1 \odot x_0. \quad (5)$$

dla pięciu zmiennych

$$x_4 \oplus x_3 \oplus x_2 \oplus x_1 \oplus x_0 = x_4 \odot x_3 \odot x_2 \odot x_1 \odot x_0 \text{ itd. (6)}$$

Tym samym dla parzystej liczby argumentów funkcje XOR i XNOR wzajemnie się dopełniają.

Powyższe rozważania można rekurencyjnie uogólnić na łańcuchy operacji $\oplus$ i $\odot$, które wygodnie jest uporządkować do postaci

$$x_{n-1} \oplus x_{n-2} \oplus \dots \oplus x_k \odot x_{k-1} \odot \dots \odot x_0. \quad (7)$$

Łańcuchy powyższe są równoważne jednolitej strukturze z operacjami $\oplus$, gdy liczba operacji równoważnościowych $\odot$ jest parzysta lub też dopełniają tę strukturę, gdy liczba operacji $\odot$ jest nieparzysta. Analogicznie, parzysta liczba operacji $\oplus$ czyni łańcuch równoważnym strukturze jednolitej z operatorami $\odot$, natomiast liczba nieparzysta operatorów $\oplus$ - dopełnia ten sam łańcuch połączony operatorami równoważności $\odot$. Przykładowo, dla trzech zmiennych

$$\overline{a \oplus b \oplus c} = a \odot b \oplus c = a \oplus b \odot c,$$

dla pięciu zmiennych (6)

$$\begin{aligned} x_4 \oplus x_3 \oplus x_2 \oplus x_1 \oplus x_0 &= x_4 \oplus x_3 \oplus x_2 \odot x_1 \odot x_0 \\ &= \overline{x_4 \oplus x_3 \oplus x_2 \oplus x_1 \odot x_0} \\ &= \overline{x_4 \oplus x_3 \odot x_2 \odot x_1 \odot x_0}, \end{aligned}$$

lub

$$\begin{aligned} x_4 \odot x_3 \odot x_2 \odot x_1 \odot x_0 &= x_4 \odot x_3 \odot x_2 \oplus x_1 \oplus x_0 \\ &= \overline{x_4 \odot x_3 \odot x_2 \odot x_1 \oplus x_0} \\ &= \overline{x_4 \odot x_3 \oplus x_2 \oplus x_1 \oplus x_0}. \end{aligned}$$

Operatory XOR oraz XNOR mogą tworzyć systemy funkcjonalnie pełne z bramkami sumy lub iloczynu oraz negacji (pkt. 2.10), w związku z czym nie są tak uniwersalne jak operatory NOR czy NAND. Jednak z uwagi na wymienione już własności, użyteczne są w różnego rodzaju cyfrowych układach liczących, np. w procesie kontroli parzystości liczby zer lub jedynek w ciągach binarnych.

2.12. Minimalizacja formuł boolowskich

Minimalizacja formuł boolowskich jest zagadnieniem ogólnym, a jego warianty szczegółowe zależą od przyjętych kryteriów minimalizacji. Najczęściej spotykanym kryterium jest minimalizacja liczby literałów w formule. Doprowadza to w konsekwencji do najprostszych realizacji bramkowych określonej funkcji boolowskiej. Minimalizację taką można przeprowadzać stosując przekształcenia wynikające z dotychczas poznanych twierdzeń i aksjomatów.

2.12.1. Heurystyczne znajdowanie postaci minimalnej formuły boolowskiej

Postępowanie to zilustruje poniższy przykład.

Przykład 2.6. Znaleźć postać minimalną funkcji opisanej następującą formułą

$$F = \bar{w} \bar{y} + w \bar{x} \bar{y} \bar{z} + w \bar{x} \bar{y} z + w x y z + w x y \bar{z}.$$

Stosując prawo rozdzielczości, przemienności i łączności operatorów (wyciąganie przed nawias) oraz aksjomat A_4 definicji 2.1 i twierdzenie 2.8 można napisać

$$\begin{aligned} F &= \bar{y}(\bar{w} + w \bar{x} \bar{z}) + w \bar{x} \bar{y}(\bar{z} + z) + w x y (\bar{z} + z) = \\ &= \bar{w} \bar{y} + \bar{x} \bar{y} \bar{z} + w \bar{x} \bar{y} + w x y. \end{aligned}$$

Właściwie teraz nie widać bezpośrednio dalszych możliwości minimalizacji, choć docelowa formuła minimalna, którą podamy a priori, ma postać

$$F = \bar{w} \bar{y} + \bar{x} \bar{y} + w x y.$$

Przed dokonaniem przekształceń redukujących, sprowadźmy więc funkcję F do postaci kanonicznej sumy iloczynów (lub iloczynu sum)

$$\begin{aligned} F &= \bar{w} \bar{x} \bar{y} \bar{z} + \bar{w} \bar{x} \bar{y} z + \bar{w} x \bar{y} \bar{z} + \bar{w} x \bar{y} z + w \bar{x} \bar{y} \bar{z} + \\ &+ w \bar{x} \bar{y} z + w x y \bar{z} + w x y z = \sum (0, 1, 4, 5, 8, 9, 14, 15). \end{aligned}$$

Redukcja liczby literałów polega na tzw. sklejeniu tych mintermów, które różnią się tylko na jednej pozycji, przy czym zgodnie z twierdze-

niem 2.3, każdy z mintermów może być użyty dowolną ilość razy. Na poszczególnych iloczynach wykonujemy zgodnie z aksjomatem A_4 operację zwaną operacją sklejaną

$$A x_i + A \bar{x}_i = A, \quad (1)$$

(lub dualnie dla sum: $(A + x_i)(A + \bar{x}_i) = A$), przy czym A reprezentuje część wspólną dla obu iloczynów. Iloczyny (sumy) różniące się tylko na jednej pozycji nazywamy iloczynami (sumami) sąsiednimi. Dla ułatwienia poszukiwań mintermów sąsiednich, uporządkujemy więc w grupach te mintermy, które mają jednakową liczbę zmiennych w postaci afirmacyjnej (i zanegowanej):

Grupa 0: $\bar{w} \bar{x} \bar{y} \bar{z}$ - brak zmiennych w postaci afirmacyjnej,

Grupa 1: $\bar{w} \bar{x} \bar{y} z$ - po jednej zmiennej w postaci afirmacyjnej,

$$\bar{w} x \bar{y} \bar{z}$$

$$w \bar{x} \bar{y} \bar{z}$$

Grupa 2: $\bar{w} x \bar{y} z$ - po dwie zmienne w postaci afirmacyjnej,

$$w \bar{x} \bar{y} z$$

Grupa 3: $w x y \bar{z}$ - trzy zmienne w postaci afirmacyjnej,

Grupa 4: $w x y z$ - cztery zmienne w postaci afirmacyjnej.

Jest oczywiste, że mintermów sąsiednich można szukać jedynie w grupach o liczbie zmiennych w postaci afirmacyjnej, różniących się dokładnie o 1, a więc np. między grupą 0 i grupą 2 nie ma możliwości znalezienia mintermów spełniających zależność (1), gdyż mintermy z tych grup różnią się na dwóch pozycjach. Między grupą 0, zawierającą zawsze tylko jeden minterm, a grupą 1 można dokonać tyle sklejeń typu (1) ile jest mintermów w tej grupie, przy czym w wyniku sklejenia otrzymamy wyrażenie zredukowane o ten literał, który występuje w jednej grupie w postaci afirmacyjnej, w drugiej - zanegowany. Ze sklejenia mintermu zerowego z mintermami grupy 1 otrzymamy ostatecznie następujące iloczyny:

$$\begin{array}{l} \hline \bar{w} \bar{x} \bar{y} \bar{z} \\ \hline \bar{w} \bar{x} \bar{y} z \\ \bar{w} x \bar{y} \bar{z} \\ w \bar{x} \bar{y} \bar{z} \end{array} \Rightarrow \begin{array}{l} \bar{w} \bar{x} \bar{y} - \\ \bar{w} - \bar{y} \bar{z} \\ - \bar{x} \bar{y} \bar{z} \end{array} \iff \begin{array}{l} \bar{w} \bar{x} \bar{y} \\ \bar{w} \bar{y} \bar{z} \\ \bar{x} \bar{y} \bar{z} \end{array}$$

przy czym pozycje sklejanego wg reguły (1) zaznaczono kreskami. Suma iloczynów prawej kolumny zawiera pełną informację o wszystkich mintermach, z których została wygenerowana. Ponieważ nie zawsze tak być

musi, zaznaczamy dowolnym (V) znakiem mintermy reprezentowane w nowo-otrzymanej kolumnie. Przejdźmy teraz do sklejania mintermów sąsiednich, grupy 1 i grupy 2:

$$\begin{array}{l}
 \vee \bar{w} \bar{x} \bar{y} z \\
 \vee \bar{w} x \bar{y} \bar{z} \\
 \vee w \bar{x} \bar{y} \bar{z} \\
 \hline
 \vee \bar{w} x \bar{y} z \\
 \vee w \bar{x} \bar{y} z
 \end{array}
 \Rightarrow
 \begin{array}{l}
 \bar{w} - \bar{y} z \\
 - \bar{x} \bar{y} z \\
 \bar{w} x \bar{y} - \\
 w \bar{x} \bar{y} -
 \end{array}
 \Leftrightarrow
 \begin{array}{l}
 \bar{w} \bar{y} z \\
 \bar{x} \bar{y} z \\
 \bar{w} x \bar{y} \\
 w \bar{x} \bar{y}
 \end{array}$$

I w tym przypadku wszystkie mintermy obu grup są reprezentowane przez kolumnę otrzymaną w wyniku sklejenia mintermów sąsiednich, co zaznaczono symbolicznie V. Jeśli zestawimy uzyskane w wyniku sklejenia kolumny, to dalej można szukać iloczynów sąsiednich w analogiczny sposób, z tym, że skleją się oczywiście tylko te, które mają kreski na tych samych pozycjach i różnią się tylko na którejś z pozostałych pozycji. Otrzymamy więc:

$$\begin{array}{l}
 \vee \bar{w} \bar{x} \bar{y} - \\
 \vee \bar{w} - \bar{y} \bar{z} \\
 \vee - \bar{x} \bar{y} \bar{z} \\
 \hline
 \vee \bar{w} - \bar{y} z \\
 \vee - \bar{x} \bar{y} z \\
 \vee \bar{w} x \bar{y} - \\
 \vee w \bar{x} \bar{y} -
 \end{array}
 \Rightarrow
 \begin{array}{l}
 \bar{w} - \bar{y} - \\
 - \bar{x} \bar{y} - \\
 \bar{w} - \bar{y} - \\
 - \bar{x} \bar{y} -
 \end{array}
 \Leftrightarrow
 \begin{array}{l}
 \bar{w} \bar{y} \\
 \bar{x} \bar{y}
 \end{array}$$

Spośród iloczynów wyjściowych, wszystkie są reprezentowane w kolumnie wygenerowanej w wyniku sklejeń (1).

W analogiczny sposób dokonujemy sklejenia grupy 2 i grupy 3:

$$\begin{array}{l}
 \bar{w} x \bar{y} z \\
 \vee w \bar{x} \bar{y} \bar{z} \\
 \hline
 w x y \bar{z}
 \end{array}
 \Rightarrow (-) \quad \text{brak możliwości sklejeń}$$

oraz grupy (3) i (4):

$$\begin{array}{l}
 \vee w x y \bar{z} \\
 \hline
 \vee w x y z
 \end{array}
 \Rightarrow w x y -$$

Żestawmy łącznie uzyskane rezultaty:

$$\begin{array}{l}
 \begin{array}{c}
 \overline{V} \overline{w} \overline{x} \overline{y} \overline{z} \\
 \overline{V} \overline{w} \overline{x} \overline{y} z \\
 \overline{V} \overline{w} x \overline{y} \overline{z} \\
 \overline{V} \overline{w} x \overline{y} z \\
 \overline{V} \overline{w} x y \overline{z} \\
 \overline{V} \overline{w} x y z \\
 \overline{V} w x y \overline{z} \\
 \overline{V} w x y z
 \end{array}
 \Rightarrow
 \begin{array}{c}
 \overline{V} \overline{w} \overline{x} \overline{y} - \\
 \overline{V} \overline{w} - \overline{y} \overline{z} \\
 \overline{V} - \overline{x} \overline{y} \overline{z} \\
 \overline{V} \overline{w} - \overline{y} z \\
 \overline{V} - \overline{x} \overline{y} z \\
 \overline{V} \overline{w} x \overline{y} - \\
 \overline{V} w \overline{x} \overline{y} - \\
 \underline{\underline{A w x y -}}
 \end{array}
 \Rightarrow
 \begin{array}{c}
 \overline{w} - \overline{y} - \\
 - \overline{x} \overline{y} - \\
 \overline{w} - \overline{y} - \\
 - \overline{x} \overline{y} -
 \end{array}
 \Rightarrow
 \begin{array}{c}
 B \overline{w} - \overline{y} - \\
 C - \overline{x} \overline{y} -
 \end{array}
 \end{array}$$

Iloczyny, które dalej nie dały się sklejać oznaczono literami A,B,C. Reprezentują one niewątpliwie wszystkie mintermy funkcji F, a więc można napisać

$$F = A + B + C = w x y + \overline{w} \overline{y} + \overline{x} \overline{y}.$$

	$\overline{w}\overline{x}\overline{y}\overline{z}$	$\overline{w}\overline{x}\overline{y}z$	$\overline{w}x\overline{y}\overline{z}$	$\overline{w}x\overline{y}z$	$w\overline{x}\overline{y}\overline{z}$	$w\overline{x}\overline{y}z$	$wxy\overline{z}$	$wxyz$	
wxy							X	X	*
$\overline{w}\overline{y}$	X	X	X	X					*
$\overline{x}\overline{y}$	X	X			X	X			*
	*	*	*	*	*	*	*	*	

Rys. 2.16. Weryfikacja pokrycia mintermów funkcji
 $f(w,x,y,z) = \sum (0,1,4,5,8,9,14,15)$

Otrzymana postać jest bardziej skrócona w porównaniu z formułą kanoniczną, jednak przedstawiony algorytm postępowania gwarantuje wprawdzie, że wszystkie mintermy są reprezentowane, lecz nie ma pewności czy nie są reprezentowane z nadmiarem. W wyniku sklejania bowiem elementów grupy k z elementami grupy k + 1 mogą zostać wygenerowane iloczyny reprezentujące te mintermy grupy k, które reprezentują już iloczyny wygenerowane w wyniku sklejanie grupy k z grupą k - 1. Aby więc otrzymać postać minimalną należy zweryfikować otrzymany rezultat i wy-

eliminować nadmiarowe składniki formuły opisującej funkcję. Można tego dokonać w oparciu o tabelę pokazaną na rysunku 2.16. Jak wynika z tabeli do pełnego pokrycia mintermów funkcji F są potrzebne wszystkie składniki skróconej formuły boolowskiej uzyskanej w wyniku generowania kolejnych skłajeń. Iloczyn pokrywa te mintermy, w skład których on wchodzi. Wyłania się przy tym reguła wynikająca zresztą bezpośrednio z zależności (1), że iloczyn o literał mniejszy od wymiaru mintermu pokrywa dwa mintermy, o dwa literały - cztery i ogólnie o n literałów mniejszy pokrywa 2^n mintermów funkcji wyjściowej.

Operowanie literałami w rozpatrzonym przykładzie jest trochę niewygodne. Jeśli więc przyjąć konwencję porządkową: wagi wg kolejności alfabetycznej w, x, y, z i afirmacji zmiennej przypisać 1, a negacji 0, to podane w przykładzie 2.6 kolumny grup można zapisać następująco:

<u>w x y z</u>		<u>w x y z</u>		<u>w x y z</u>		<u>w x y z</u>
V 0 0 0 0		V 0 0 0 -		0 - 0 -	$\Rightarrow$	B 0 - 0 -
V 0 0 0 1		V 0 - 0 0		- 0 0 -		C - 0 0 -
V 0 1 0 0	$\Rightarrow$	V - 0 0 0	$\Rightarrow$	0 - 0 -		
V 1 0 0 0		V 0 - 0 1		- 0 0 -		
V 0 1 0 1		V - 0 0 1		0 - 0 -		
V 1 0 0 1		V 0 1 0 -		- 0 0 -		
V 1 1 1 0		V 1 0 0 -				
V 1 1 1 1		A 1 1 1 -				

W analogiczny sposób można zmodyfikować tabelę pokazaną na rysunku 2.16. Postać końcową doszyfrujemy zgodnie z przyjętą konwencją

$$F = (1 \ 1 \ 1 \ -) + (0 \ - \ 0 \ -) + (- \ 0 \ 0 \ -) = w \ x \ y + \bar{w} \ \bar{y} + \bar{x} \ \bar{y}.$$

Dalsze uproszczenie zapisu kolumn można uzyskać nadając poszczególnym-wynikającym z konwencji porządkowej - pozycjom, wag charakterystycznych dla zapisu liczby binarnej i następnie przedstawienia jej odpowiednika dziesiętnego. Sprowadza się to do zastąpienia mintermu - jego indeksem. Trzeba jednak przy podziałach na grupy wiedzieć ile jedynek w zapisie binarnym miałyby liczba charakteryzująca indeks mintermu. Liczbę tę przyjęto nazywać liczbą indeksową. Tak więc np. liczba indeksowa mintermu 0 wynosi zero, mintermów 1,4,8 - 1, mintermów 5,9 -

2 itd. Grupy tworzymy dla mintermów o tych samych liczbach indeksowych. Aby znaleźć mintermy sąsiednie dla sąsiadujących ze sobą grup, wystarczy zauważyć, że gwarancje na to, by różnica dotyczyła tylko jednej pozycji dają dwa warunki:

- 1) różnica między indeksami mintermów musi być całkowitą potęgą liczby 2, tzn. 1, 2, 4, 8 itd.,
 - 2) liczba w grupie o większej liczbie indeksowej musi być większa.
- Dla funkcji podanej w przykładzie 2.6, można zatem kolumny sklejaną zapisać następująco:

$$f(w, x, y, z) = \sum (0, 1, 4, 5, 8, 9, 14, 15)$$

$$(0)(1)(1)(2)(1)(2)(3)(4)$$

(w nawiasach podano liczby indeksowe),

<u>V 0</u>	V 0, 1 (1)		
V 1	V 0, 4 (4)	0, 1, 4, 5 (1, 4)	
V 4	V 0, 8 (8)	0, 1, 8, 9 (1, 8)	B 0, 1, 4, 5 (1, 4)
<u>V 8</u>	V 1, 5 (4)	0, 4, 1, 5 (4, 1)	C 0, 1, 8, 9 (1, 8)
V 5	V 1, 9 (8)	0, 8, 1, 9 (8, 1)	
V 9	V 4, 5 (1)		
V 14	V 8, 9 (1)		
<u>V 15</u>	<u>A 14, 15 (1)</u>		

Drugą kolumnę tworzymy z pierwszej pisząc ją i zachowując kolejność: liczba grupy niższej, liczba grupy wyższej i podając w nawiasie różnicę między tymi liczbami. Różnica ta reprezentuje wagę pozycji, na której występuje "kreska" w zapisie binarnym. Trzecią kolumnę tworzymy z drugiej, tworząc uporządkowane zestawy par sklejanych tylko wtedy, gdy w nawiasach kolumny drugiej występują te same wagi, a różnica na poszczególnych pozycjach jest potęgą liczby 2. Zapis wyniku sklejenia tworzymy poprzez zestawienie uporządkowanych par liczbowych, przy czym w nawiasie podajemy różnicę liczb z odpowiadających sobie pozycji. Obowiązuje również ta sama zasada, że liczba z niższej grupy indeksowej musi być mniejsza od znajdującej się na odpowiadającej jej pozycji liczby z grupy indeksowej wyższej. W wyniku sklejenia par, w nawiasach będą dwie liczby reprezentujące wagi, na których w ich odpowied-

nikach binarnych znajdują się "kreski". Procedurę tę należy kontynuować dalej aż do wyczerpania wszystkich możliwości sklejeń (np. czwarta kolumna zawierałaby uporządkowane dwie czwórki liczb i trzy liczby wagowe w nawiasach). Skróconą postać funkcji F określamy analogicznie jak poprzednio, uwzględniając specyfikę przyjętej konwencji

$$F = [14, 15 (1)] + [0, 1, 4, 5 (1, 4)] + [0, 1, 8, 9 (1, 8)] = \\ = (1 \ 1 \ 1 \ -) + (0 \ - \ 0 \ -) + (- \ 0 \ 0 \ -).$$

Zapis ten oznacza, że biorąc dowolną z liczb podaną w nawiasach prostokątnych i pisząc jej odpowiednik binarny z "kreską" na pozycji odpowiadającej liczbie wagowej, otrzymujemy ciąg zero-jedynkowy, reprezentujący odpowiednie postacie literałów. Ostatecznie można napisać

$$F = w \times y + \bar{w} \bar{y} + \bar{x} \bar{y}.$$

	0	1	4	5	8	9	14	15	
14, 15 (1)							X	X	*
0, 1, 4, 5, (1, 4)	X	X	X	X					*
0, 1, 8, 9 (1, 8)	X	X			X	X			*
	*	*	*	*	*	*	*	*	

Rys. 2.17. Odpowiednik dziesiętny tablicy weryfikacyjnej pokazanej na rys. 2.16

Tablica weryfikacyjna dla zapisu dziesiętnego jest szczególnie prosta. Przedstawiono ją na rysunku 2.17. Uporządkowane ciągi liczb otrzymane w wyniku skiejenia, reprezentują bezpośrednio indeksy mintermów przez nie pokrywanych.

2.12.2. Algorytmiczne i tablicowe metody znajdowania postaci minimalnych formuł boolowskich

Rozważania przeprowadzone w poprzednim podpunkcie pozwalają na określenie algorytmu postępowania w procesie poszukiwania minimalnej formuły określającej funkcje boolowską. Przed jego określeniem, wygodnie jest wprowadzić poniższe definicje.

Definicja 2.9. Implikantem funkcji F nazywamy funkcję f tych samych zmiennych taką, że jeśli $f = 1$, to i $F = 1$, tzn. $f \rightarrow F$. Implikantem prostym nazywamy taki implikant, który przestaje być implikantem, jeśli wyeliminujemy z niego dowolny literał.

Przykładowo implikantami funkcji

$$F = w \times y + \bar{w} \bar{y} + \bar{x} \bar{y},$$

są m.in. wyrażenia: $w \times y$, $\bar{w} \bar{y}$, $\bar{x} \bar{y}$, $(w \times y + \bar{w} \bar{y})$, $(\bar{w} \bar{y} + \bar{x} \bar{y})$, $(w \times y + \bar{w} \bar{y} + \bar{x} \bar{y})$. Spośród tych wyrażeń implikantami prostymi są iloczyny: $w \times y$; $\bar{w} \bar{y}$, $\bar{x} \bar{y}$. Szczególnym przypadkiem implikantu jest funkcja sama dla siebie. Mintermy funkcji są jednocześnie jej implikantami.

Definicja 2.10. Implicentem funkcji H nazywamy funkcję h tych samych zmiennych taką, że jeśli $h = 0$, to i $H = 0$, tzn. $\bar{h} \rightarrow \bar{H}$. Implicentem prostym nazywamy taki implicent, który przestaje być implicentem, jeśli wyeliminujemy z niego dowolny literał.

Przykładowo, implicentami funkcji dualnej do podanej poprzednio

$$H = (w + x + y)(\bar{w} + \bar{y})(\bar{x} + \bar{y}),$$

są m.in. wyrażenia: $(w + x + y)$, $(\bar{w} + \bar{y})$, $(\bar{x} + \bar{y})$, $(w + x + y)(\bar{w} + \bar{y})$, $(\bar{w} + \bar{y})(\bar{x} + \bar{y})$, $(w + x + y)(\bar{x} + \bar{y})$, $(w + x + y)(\bar{w} + \bar{y})(\bar{x} + \bar{y})$. Spośród tych wyrażeń implicentami prostymi są sumy: $(w + x + y)$, $(\bar{w} + \bar{y})$, $(\bar{x} + \bar{y})$. Szczególnym przypadkiem implicentu jest funkcja sama dla siebie. Makstermy funkcji są jednocześnie jej implicentami.

Procedura poszukiwań minimalnej formuły boolowskiej dla danej funkcji sprowadza się do przedstawienia jej albo w postaci sumy implikantów prostych (jak w p. 2.12.1) albo iloczynu prostych implicentów, przy czym algorytm postępowania obejmuje kolejne etapy.

A. Sprowadzenie funkcji do postaci kanonicznej.

B. Utworzenie pierwszej kolumny poprzez podział indeksów mintermów lub makstermów na grupy indeksowe o jednakowej liczbie jedynek w zapisie binarnym indeksu (rys. 2.6). Grupy umieszczamy wg kolejności wzrastania (tzn. 0,1,2,3 - jedynek, itd.). Jeśli jakaś grupa nie występuje należy to zaznaczyć.

C. Drugą kolumnę tworzymy z pierwszej poprzez sklejanie wszystkich iloczynów lub sum sąsiednich wybieranych z kolejnych grup indeksowych. Reguły sklejanie dla zapisu dziesiętnego są następujące:

- sklejać można tylko wyrażenia z kolejnych grup indeksowych (tzn.: $0 \div 1$, $1 \div 2$, $2 \div 3$, $3 \div 4$ itd.),
- różnica między sklejanymi liczbami musi być całkowitą potęgą 2, (tzn. 1, 2, 4, 8 itd.),
- liczba o indeksie niższym musi być mniejsza od liczby o indeksie wyższym.

Wynik zapisujemy w postaci pary uporządkowanej, przy czym w nawiasie umieszczamy różnice obu liczb. Różnica ta reprezentuje wagę pozycji, na której zmienna została wyeliminowana (sklejona). Rezultat sklejenia zaznaczamy w kolumnie pierwszej. Nie sklejone mintermy lub makstermy są implikantami lub implicantami prostymi.

D. N-tą kolumnę tworzymy z kolumny N-1 w analogiczny sposób jak w etapie C, z tym że porównujemy uporządkowane ciągi liczb (pary, czwórki, ósemki itd.) i różnice między znajdującymi się na odpowiadających sobie pozycjach liczbami umieszczamy w nawiasach w dowolnej kolejności. Sklejać można tylko te ciągi liczb, które w nawiasach mają dokładnie te same liczby wagowe. Rezultaty sklejeń zaznaczamy w kolumnie N-1. Nie zaznaczone wyrażenia są implikantami lub implicantami prostymi. Procedurę powtarzamy aż do wyczerpania wszystkich możliwości sklejeń.

E. Z otrzymanych implikantów lub implicantów prostych eliminujemy te, które są zbędne dla pokrycia wszystkich mintermów lub makstermów funkcji minimalizowanej. Można to zrobić w oparciu o tabelę analogiczną do pokazanej na rysunku 2.17.

F. Deszyfrujemy wynik, tzn. przechodzimy od zapisu indeksowego do zapisu algebraicznego, przy czym zmienne znajdujące się na pozycjach zaznaczonych liczbami wagowymi są wyeliminowane.

Przedstawiony algorytm jest często określany jako tabulacyjna metoda poszukiwania skróconej postaci funkcji boolowskiej. Podstawy tej metody zostały sformułowane po raz pierwszy dla zapisu literałowego przez Quine'a w roku 1952 [3,6]. W roku 1956 McCluskey [3,6] opracował wersję metody Quine'a, wykorzystującą dziesiętny zapis indeksów mintermów, w związku z czym omówiona metoda tabulacyjna nosi nazwę metody Quine'a McCluskey'a.

Przykład 2.7. Znaleźć metodą tabelacyjną minimalną formułę funkcji

$$F = (w + \bar{y})(\bar{w} + x + \bar{y})(\bar{w} + \bar{x} + y + z)(\bar{w} + \bar{x} + y + \bar{z})$$

Sprowadźmy funkcję do postaci kanonicznej iloczynu sum. W tym celu wyrażenia w nawiasach należy rozwinąć do sum czteroliterałowych:

$$(w + \bar{y}) = (w + \bar{y} + x \bar{x}) = (w + x + \bar{y}) (w + \bar{x} + \bar{y}),$$

$$(w + x + \bar{y} + z \bar{z}) = (w + x + \bar{y} + z) (w + x + \bar{y} + \bar{z}) = M_2 \cdot M_3,$$

$$(w + \bar{x} + \bar{y} + z \bar{z}) = (w + \bar{x} + \bar{y} + z) (w + \bar{x} + \bar{y} + \bar{z}) = M_6 \cdot M_7,$$

$$(\bar{w} + x + \bar{y} + z \bar{z}) = (\bar{w} + x + \bar{y} + z) (\bar{w} + x + \bar{y} + \bar{z}) = M_{10} \cdot M_{11},$$

$$(\bar{w} + \bar{x} + y + z) (\bar{w} + \bar{x} + y + \bar{z}) = M_{12} \cdot M_{13},$$

Ostatecznie można napisać

$$\begin{aligned} F(w,x,y,z) &= M_2 \cdot M_3 \cdot M_6 \cdot M_7 \cdot M_{10} \cdot M_{11} \cdot M_{12} \cdot M_{13} = \\ &= \prod (2,3,6,7,10,11,12,13). \end{aligned}$$

W wyniku podziału na grupy indeksowe oraz sklejanie sąsiednich impli-
centów, otrzymamy następujące kolumny:

<u>V 2</u>	V 2,3 (1)			
V 3	V 2,6 (4)	$\Rightarrow$	$\Rightarrow$	$\Leftrightarrow$
V 6	V 2,10 (8)			
V 10	V 3,7 (4)		2,3,6,7 (1,4)	B 2,3,6,7 (1,4)
V 12	V 3,11 (8)		2,3,10,11 (1,8)	C 2,3,10,11(1,8)
<u>V 7</u>	V 6,7 (1)			
V 11	V 10,11 (1)			
V 13	A 12,13 (1)			

Ostatecznie otrzymujemy zaszyfrowaną postać skróconą złożoną z impli-
centów prostych A,B,C

$$F \longleftrightarrow [12,13(1)] [2,3,6,7(1,4)] [2,3,10,11(1,8)].$$

Z tabeli impli-
centów prostych przedstawionej na rysunku 2.18 wy-
nika, że są one wszystkie potrzebne. Postać końcową funkcji otrzymamy,
przechodząc najpierw na zapis binarny

$$F \longleftrightarrow [110-] [0 - 1 -] [- 0 1 -].$$

Korzystając z tablicy pokazanej na rysunku 2.6 oraz dostosowując nazwy zmiennych do nazw podanych w przykładzie, otrzymamy ostatecznie

$$F = (\bar{w} + \bar{x} + y)(w + \bar{y})(x + \bar{y}).$$

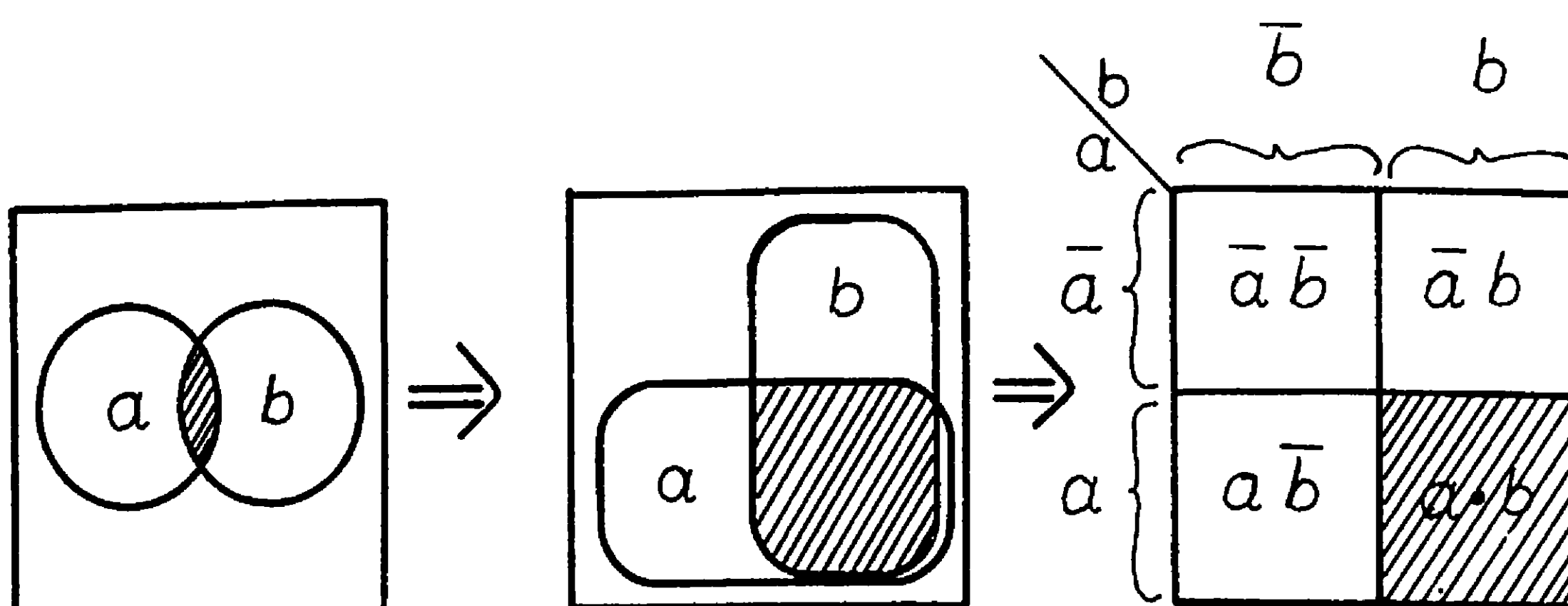
	2	3	6	7	10	11	12	13	
12, 13 (1)							X	X	*
2, 3, 6, 7 (1,4)	X	X	X	X					*
2, 3, 10, 11 (1,8)	X	X			X	X			*
	*	*	*	*	*	*	*	*	

Rys. 2.18. Tablica implicentów prostych dla funkcji z przykładu 2.7

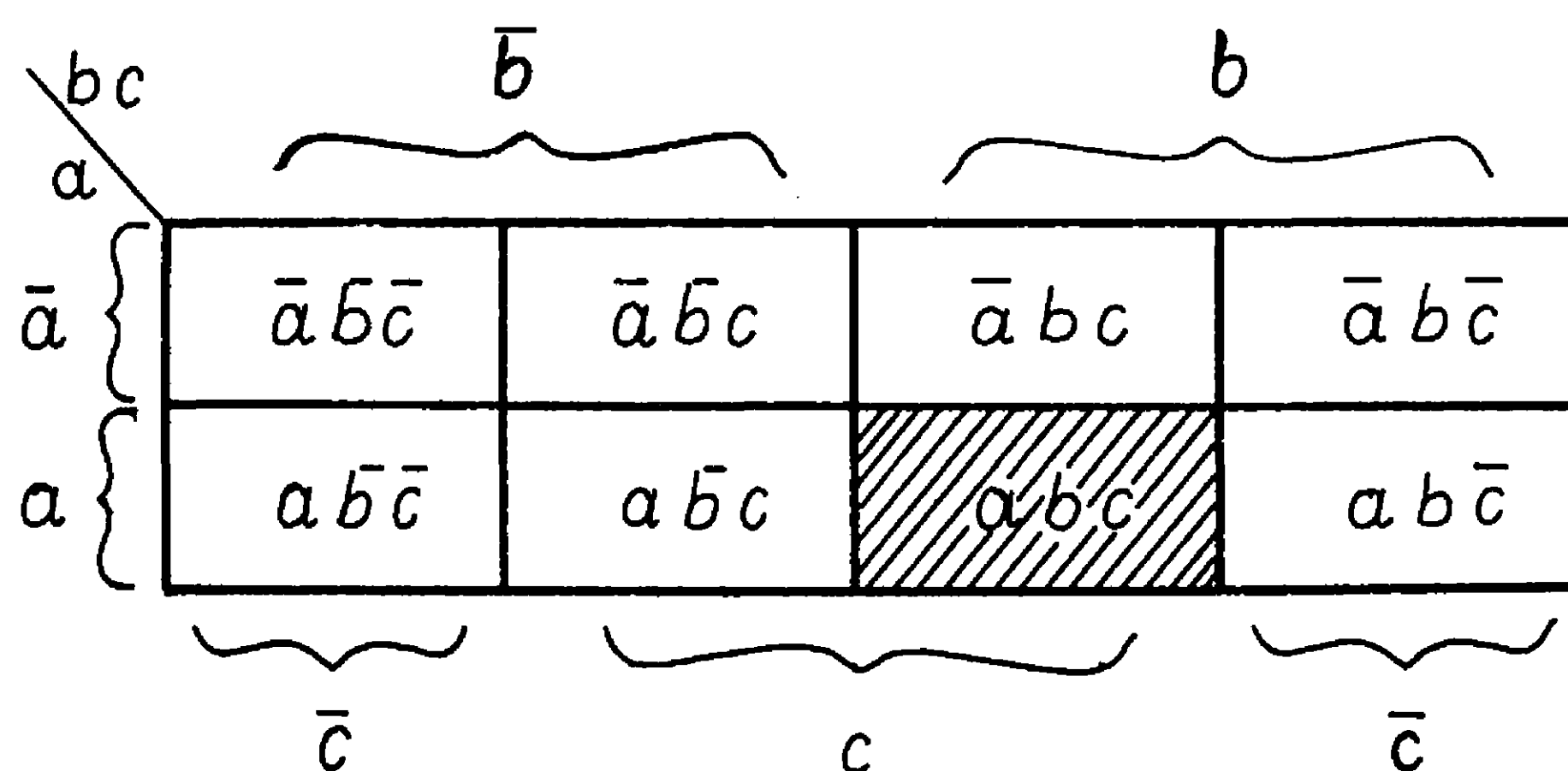
Metoda tabulacyjna Quine'a McCluskey'a jest metodą ogólną i algorytmiczną, a więc nadaje się bezpośrednio do obliczeń komputerowych. Dla małej liczby zmiennych, w podręcznym zastosowaniu nie jest ona zbyt wygodna i bardziej użyteczne są metody intuicyjne bazujące na tzw. tablicach Veitch'a i Karnaugh'a. Metoda zastosowana po raz pierwszy przez Veitch'a w 1952 r. wykorzystuje wykresy Venn'a (p. 2.5), przekształcone topologicznie do diagramów w sposób przedstawiony na rysunku 2.19. Każde pole diagramu reprezentuje obszar, odpowiadający jednemu mintermowi, przy czym mintermy z obszarów granicznych są mintermami sąsiednimi, tzn. różnią się tylko na jednej pozycji. Diagram Veitch'a dla trzech zmiennych przedstawiono na rysunku 2.20. Tworząc analogicznie diagramy dla większej liczby zmiennych należy zachować warunek sąsiedniości odpowiadających poszczególnym obszarom mintermów, co umożliwia wielokrotne przeprowadzanie operacji sklejan charakterystycznej dla metody tabulacyjnej Quine'a i McCluskey'a. W odróżnieniu jednak od metody tabulacyjnej, która jest metodą algorytmiczną, tutaj wyszukiwanie iloczynów sąsiednich jest przeprowadzane intuicyjnie, niejako poprzez spojrzenie "z góry" na topologiczny obraz funkcji.

Przykład 2.8. Znaleźć w oparciu o diagramy Veitch'a minimalną postać funkcji boolowskiej opisanej wyrażeniem

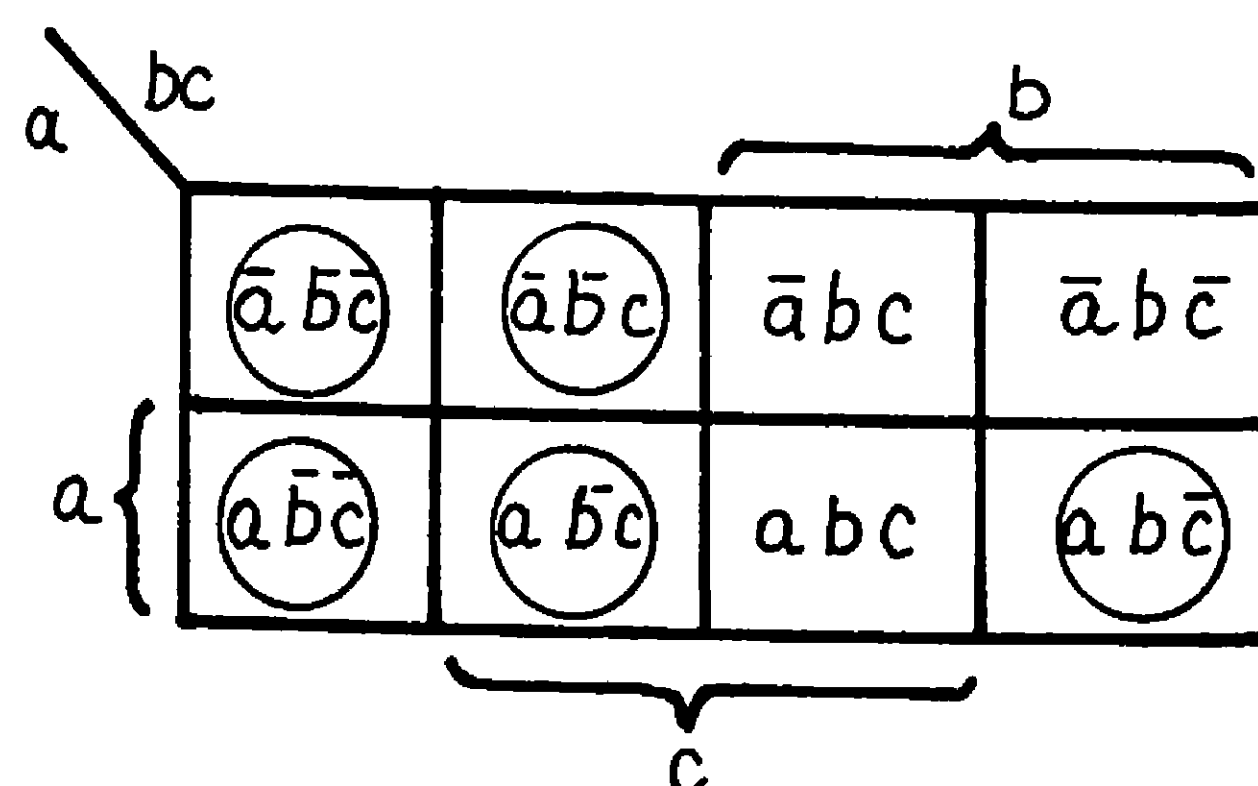
$$f(a,b,c) = \bar{a} \bar{b} \bar{c} + \bar{a} \bar{b} c + a \bar{b} \bar{c} + a \bar{b} c + a b \bar{c}.$$



Rys. 2.19. Etapy topologicznych przekształceń wykresów Venna do diagramów Veitcha



Rys. 2.20. Diagram Veitcha dla trzech zmiennych



Rys. 2.21. Funkcja z przykładu 2.8 wpisana do diagramu Veitcha

Wykorzystajmy w tym celu tabelę pokazaną na rysunku 2.20, przy czym mintermy należące do funkcji zaznaczmy kółkami (rys. 2.21). Łącząc pary pionowych sąsiednich mintermów funkcji f otrzymujemy następujące implikanty reprezentujące po dwa mintermy:

$$\bar{b} \bar{c}, \quad \text{dla} \quad \bar{a} \bar{b} \bar{c} \quad \text{i} \quad a \bar{b} \bar{c},$$

$$\bar{b} c, \quad \text{dla} \quad \bar{a} \bar{b} c \quad \text{i} \quad a \bar{b} c.$$

Łatwo zauważyć, że implikanty te są wobec siebie również sąsiednie, w związku z czym można zredukować kolejny literał otrzymując w wyniku $\bar{b}$. A więc cztery parami sąsiednie mintermy mogą być rozpatrywane łącznie jako tzw. "czwórka" reprezentowana tylko przez jeden implikant, w tym przypadku $\bar{b}$. Pozostaje jeszcze problem mintermu $a b \bar{c}$, ale wystarczy zauważyć, że obszary leżące na brzegach są również wobec siebie sąsiednie, a więc minterm ten skleja się z mintermem $a \bar{b} \bar{c}$, generując w rezultacie implikant $a \bar{c}$. Ostatecznie otrzymamy następującą postać minimalną funkcji

$$f = \bar{b} + a \bar{c}.$$

Przy pewnej wprawie wynik ten można uzyskać bardzo szybko, podczas gdy metoda tabulacyjna wymaga dość żmudnych konstrukcji poszczególnych kolumn złożonych z wygenerowywanych implikantów.

Znaczne usprawnienie w posługiwaniu się diagramami Veitch'a wprowadził w roku 1953 Karnaugh [5,6], stosując zamiast opisów literałowych, konwencję zero-jedynkową podaną w tabeli pokazanej na rysunku 2.6. Diagramy z tą konwencją oznaczeń obszarów zmiennych noszą nazwę diagramów lub tablic Karnaugh. Dla dwóch i trzech zmiennych diagramy te przedstawiono na rysunku 2.22. Na rysunku 2.23 wpisano funkcję z przykładu 2.8, przy czym symbolami 1 oznaczono te kratki, w których znajdują się mintermy rozpatrywanej funkcji, natomiast symbolami 0 - mintermy, dla których funkcja przyjmuje wartość zero. Ciągi zer i jedynek charakteryzujące obszary, w których poszczególne zmienne przyjmują postać afirmacyjną lub zanegowaną są tak dobrane, by zmienna dla sąsiadujących ze sobą krutek następowała tylko na jednej pozycji. To samo dotyczy krutek znajdujących się na brzegach, tak jakby diagram był rozciętą powierzchnią walca. Na rysunku 2.24 przedstawiono tablicę Karnaugh dla czterech zmiennych. Jest ona dwukrotnie większa niż tablica dla trzech zmiennych. Oprócz wymienionych już właściwości cha-

rakterystycznych dla tablicy trzech zmiennych, warunek sąsiedniości spełniają również kratki znajdujące się na brzegu górnym i dolnym. Tak więc wszystkie kratki znajdujące się na przeciwległych bokach różnią się tylko na jednej pozycji. W tablicę pokazaną na rysunku 2.25 wpisano funkcje

$$f(w, x, y, z) = \sum (0, 1, 2, 4, 5, 8, 9, 10, 12, 13).$$

a \ b	0	1
	$\bar{a}\bar{b}$	$\bar{a}b$
0	$\bar{a}\bar{b}$	$\bar{a}b$
1	$a\bar{b}$	ab

a \ bc	00	01	11	10
	$\bar{a}\bar{b}\bar{c}$	$\bar{a}\bar{b}c$	$\bar{a}bc$	$\bar{a}b\bar{c}$
0	$\bar{a}\bar{b}\bar{c}$	$\bar{a}\bar{b}c$	$\bar{a}bc$	$\bar{a}b\bar{c}$
1	$a\bar{b}\bar{c}$	$a\bar{b}c$	abc	$ab\bar{c}$

Rys. 2.22. Tablica Karnaugh dla 2 i 3 zmiennych

Osiem jedynek utworzonych z parami sąsiednich mintermów, może być zastąpione jednym implikantem, który w tym przypadku jest y . Dwa mintermy znajdujące się w dolnym i górnym rogu prawej części diagramu można połączyć w parę, ake korzystniej jest je połączyć w "czwórkę" z jedynekami znajdującymi się w pozostałych rogach. Rozpatrywana funkcja przyjmie ostatecznie postać

a \ bc	00	01	11	10
	$\bar{a}\bar{b}\bar{c}$	$\bar{a}\bar{b}c$	$\bar{a}bc$	$\bar{a}b\bar{c}$
0	1	1	0	0
1	1	1	0	1

$$f = \bar{y} + \bar{x} \bar{z}.$$

Rys. 2.23. Funkcja z przykładu 2.8 wpisana do tablicy Karnaugh

Grupy ośmiu i czterech jedynek są implikantami prostymi rozpatrywanej funkcji.

W taki sam sposób można łączyć w pary czwórki i ósemki itd. zera funkcji, z tym, że postać zminimalizowana jest wówczas koniunkcyjną postacią normalną. Dla zer rozpatrywanej funkcji można utworzyć dwie czwórki, a więc uwzględniając konwencje zero-jedynkową dla makstremów (rys. 2.6) otrzymamy

$$f = (\bar{y} + \bar{z})(\bar{x} + \bar{y}).$$

Zauważmy, że obie postacie minimalne różnią się liczbą literałów. Alternatywna postać (APN) zawiera mniej literałów niż postać koniunkcyjna (KPN), a więc zagadnienie minimalizacji jest rozwiązywane albo w klasie APN albo KPN.

wx \ yz				
	00	01	11	10
00	$\bar{w}\bar{x}\bar{y}\bar{z}$			
01				
11				
10				

Rys. 2.24. Tablica Karnaugh dla czterech zmiennych

wx \ yz				
	00	01	11	10
00	1	1	0	1
01	1	1	0	0
11	1	1	0	0
10	1	1	0	1

Rys. 2.25. Tablica Karnaugh funkcji $f(w,x,y,z) = \sum (0,1,2,4,5,8,9,10,12,13)$

$x_4 \ x_3$ \ $x_2 \ x_1 \ x_0$								
	000	001	011	010	110	111	101	100
00								
01		1	1			1	1	
11								
10								

Rys. 2.26. Tablica Karnaugh dla pięciu zmiennych

Tablice Karnaugh można tworzyć dla dowolnej liczby zmiennych. Na przykład tablicę dla pięciu zmiennych otrzymujemy ze złożenia dwóch tablic czterech zmiennych, dla których piąta zmienna raz występuje w postaci afirmacyjnej (1), raz w postaci dopełnienia (0). W tablicy pokazanej na rysunku 2.26 rolę piątej zmiennej spełnia x_2 . Konstrukcja tablicy jest taka, że sąsiednimi kratkami są nie tylko omówione poprzednio, ale również te, które leżą symetrycznie względem podwójnej środkowej linii. Można więc np. cztery jedynki pokazane na rysunku połączyć w jedną czwórkę, otrzymując implikant: $\bar{x}_4 \ x_3 \ x_0$. Dla tablicy pięciu zmiennych, zera lub jedynki mogą, obok omówionych uprzednio konfiguracji, tworzyć grupy sąsiednie "szesnastkowe". Tablica dla

sześciu zmiennych utworzona przez zestawienie dwóch tablic dla pięciu zmiennych, jest mało użyteczna, gdyż korzystanie z niej wymaga wielkiej wprawy i właściwie nie mamy pewności co do poprawności wyniku końcowego. Tak więc metoda tablic Karnaugh, bardzo użyteczne dla trzech, czterech ewentualnie pięciu zmiennych, nie znajduje zastosowania przy większej ich liczbie. Dzieje się tak z uwagi na jej intuicyjny charakter - jest to właściwie metoda prób i błędów, a rozmiary tablicy w odniesieniu do liczby zmiennych n rosną 2^n razy, co dla $n > 5$ czyni bardzo trudnym kojarzenie grup jedynek lub zer tworzących implikanty lub implikanty proste.

2.13. Faktoryzacja funkcji logicznych

Otrzymane w wyniku omówionych w punkcie 2.12 metod, minimalne formuły boolowskie są postaciami normalnymi (p. 2.8), a sama procedura została przeprowadzona również w klasie formuł normalnych. Jeśli zrezygnować z tego ograniczenia, to możliwe jest dalsze zmniejszenie liczby literałów. W szczególnie prosty sposób dokonuje się tego poprzez zastosowanie aksjomatu rozdzielczości operatorów, a więc poprzez wyciąganie przed nawias. Przykładowo, funkcja

$$f = a b + a \bar{c} + d$$

nie daje się dalej minimalizować w postaci normalnej. Jeśli zmienną a wyciągniemy przed nawias otrzymana formuła będzie zawierać o jeden literał mniej:

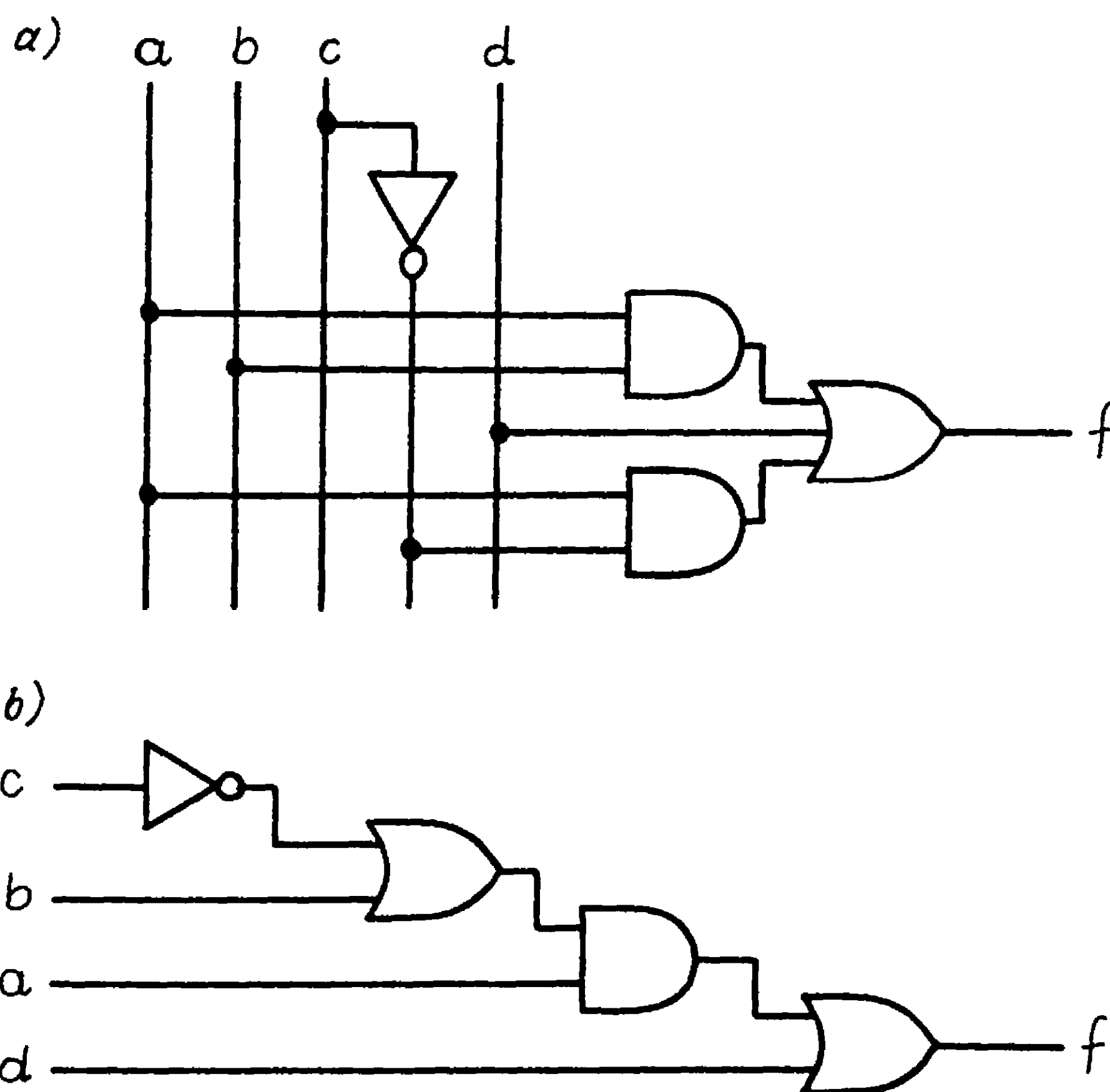
$$f = a(b + \bar{c}) + d.$$

Otrzymana formuła nie jest już jednak postacią normalną. Na rysunku 2.27 przedstawiono schemat logiczny obu postaci funkcji. Jeśli jako kryterium kosztu minimalizacji wprowadzić następującą parę liczb uporządkowanych

$$\langle \text{liczba końcówek wejściowych, liczba bramek} \rangle, \quad (1)$$

przy czym wagę większą ma liczba końcówek, to układ b) jest tańszy, gdyż koszt wynosi tu $\langle 7, 4 \rangle$, a dla układu a) $\langle 8, 4 \rangle$. Jednak ceną jaką się za to płaci jest wzrost liczby poziomów bramek, przez który musi

przejsć sygnał wejściowy do wyjścia. Zmienna c jest podawana na wyjście w układzie b) przez cztery, natomiast w układzie a) - przez trzy poziomy bramek. Formuły normalne zapewniają generalnie najmniejszą możliwą liczbę poziomów propagacji sygnału wejściowego na wyjście. Ma to bardzo duże znaczenie użytkowe z uwagi na szybkość pracy układu, w związku z czym zwiększenie liczby poziomów bramek często nie jest wskazane nawet, jeśli zmniejsza to wskaźnik kosztu (1). Jeśli jednak szybkość pracy układu nie jest parametrem krytycznym, przedstawiona w przykładzie manipulacja algebraiczna może okazać się użyteczną.



Rys. 2.27. Realizacje bramkowe funkcji $f = ab + a\bar{c} + d(a,b)$

Wyciąganie przed nawias wspólnego członu nie gwarantuje uzyskania formuły absolutnie minimalnej. Wykazał to w roku 1952 Nurkhardt [5]. Abhyankar podał w roku 1959 [5] algorytm postępowania, gwarantującego znalezienie absolutnie minimalnej formuły boolowskiej, jednak ma on znaczenie raczej teoretyczne, gdyż liczba niezbędnych elementarnych operacji logicznych N , tylko w ostatnim etapie obliczeń, mieści się w granicach

$$2^{2^{2^{n-1}}} + 1 \leq N \leq 2^{2^{2^n}}, \quad (2)$$

gdzie n - liczba zmiennych.

Jeśli $n = 2$, dla N otrzymamy

$$2^5 \leq N \leq 2^{16},$$

natomiast dla $n = 3$

$$2^{17} \leq N \leq 2^{256}.$$

Dla $n = 4$ liczba obliczeń osiąga wielkość trudną do osiągnięcia, nawet dla nowoczesnych maszyn liczących:

$$2^{257} \leq N \leq 2^{65536}.$$

Postęp w dziedzinie technologii półprzewodników, a w szczególności konstruowanie układów średniej i wielkiej skali integracji zdecydowanie zmniejszyły zainteresowanie poszukiwaniem formuł absolutnie minimalnych. Najczęściej formułę boolowską zminimalizowaną lub nie, poddaje się przekształceniom algebraicznym, dostosowując jej formę do wymagań aplikacyjnych. Postępowanie to określa się ogólną nazwą: faktoryzacja. W wyniku faktoryzacji można np. uzyskać postać funkcji, dającą się realizować wyłącznie w oparciu o bramki dwuwejściowe. Przykładowo, po minimalizacji uzyskano następującą postać funkcji

$$f = a \bar{c} + \bar{a} b c + \bar{a} \bar{b} d,$$

Do jej realizacji potrzeba jednej bramki trójwejściowej sumy na pierwszym poziomie oraz dwóch trójwejściowych bramek iloczynu na poziomie drugim (rys. 2.28a). Przekształcając funkcję poprzez wyciągnięcie literału $\bar{a}$ przed nawias, otrzymujemy formułę dającą się zrealizować wyłącznie na bramkach dwuwejściowych

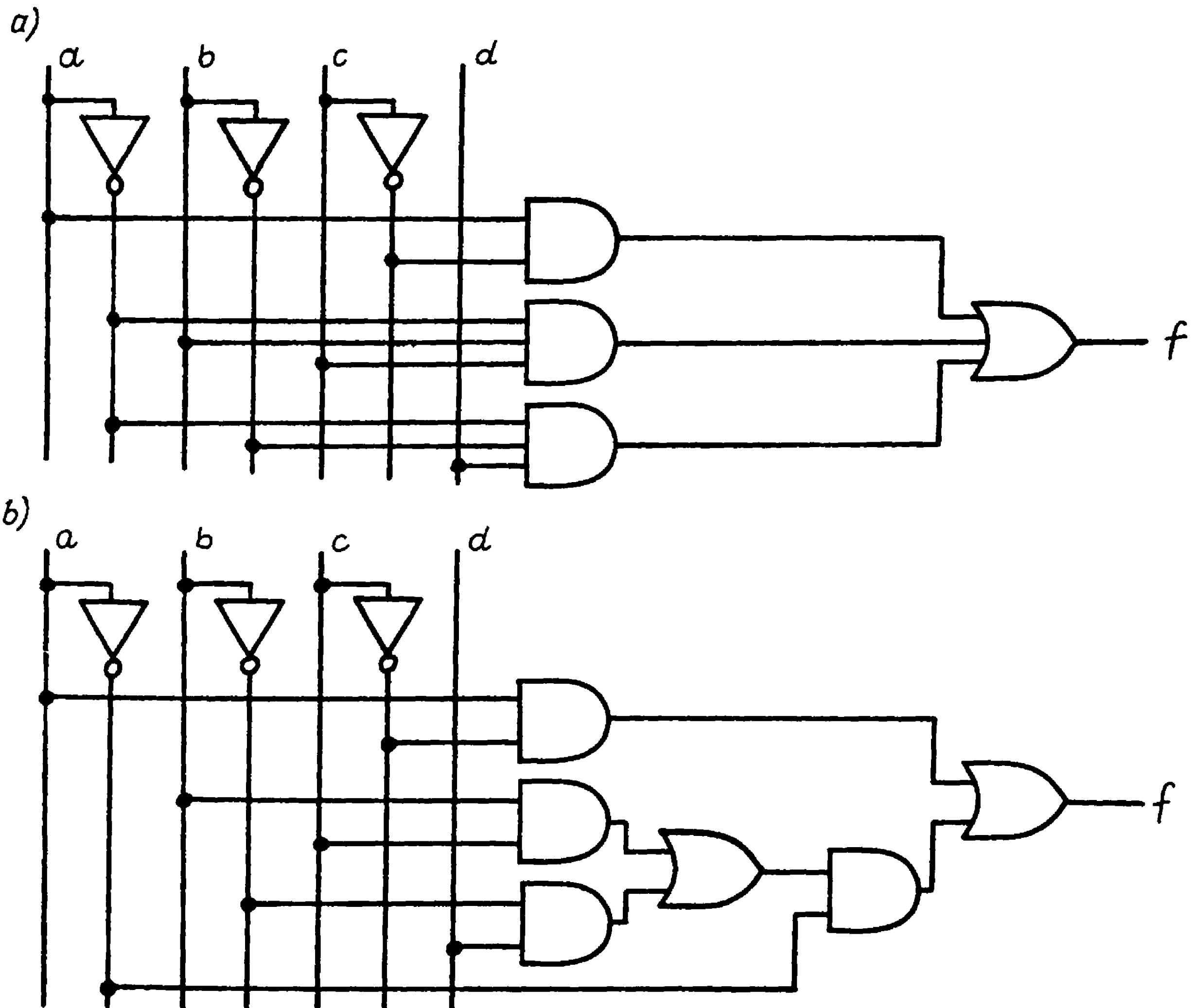
$$f = a \bar{c} + \bar{a} (b c + \bar{b} d)$$

(rys. 2.28b), a ponadto zawierającą o jeden literał mniej.

Szczególne znaczenie w procesie faktoryzacji ma przekształcenie de Morgana, dokonane w wyniku podwójnego zaprzeczenia. Dla funkcji podanej na rysunku 2.28, można wówczas napisać

$$f = \overline{\overline{f}} = \overline{(a \overline{c} + \overline{a} b c + \overline{a} \overline{b} d)} = \overline{(a \overline{c})(\overline{a} b c)(\overline{a} \overline{b} d)}.$$

Otrzymana formuła może być bezpośrednio zrealizowana z bramek NAND (rys. 2.29a).



Rys. 2.28. Schemat realizacji funkcji: a) w postaci normalnej, b) w postaci faktoryzowanej dla bramek dwuwejściowych

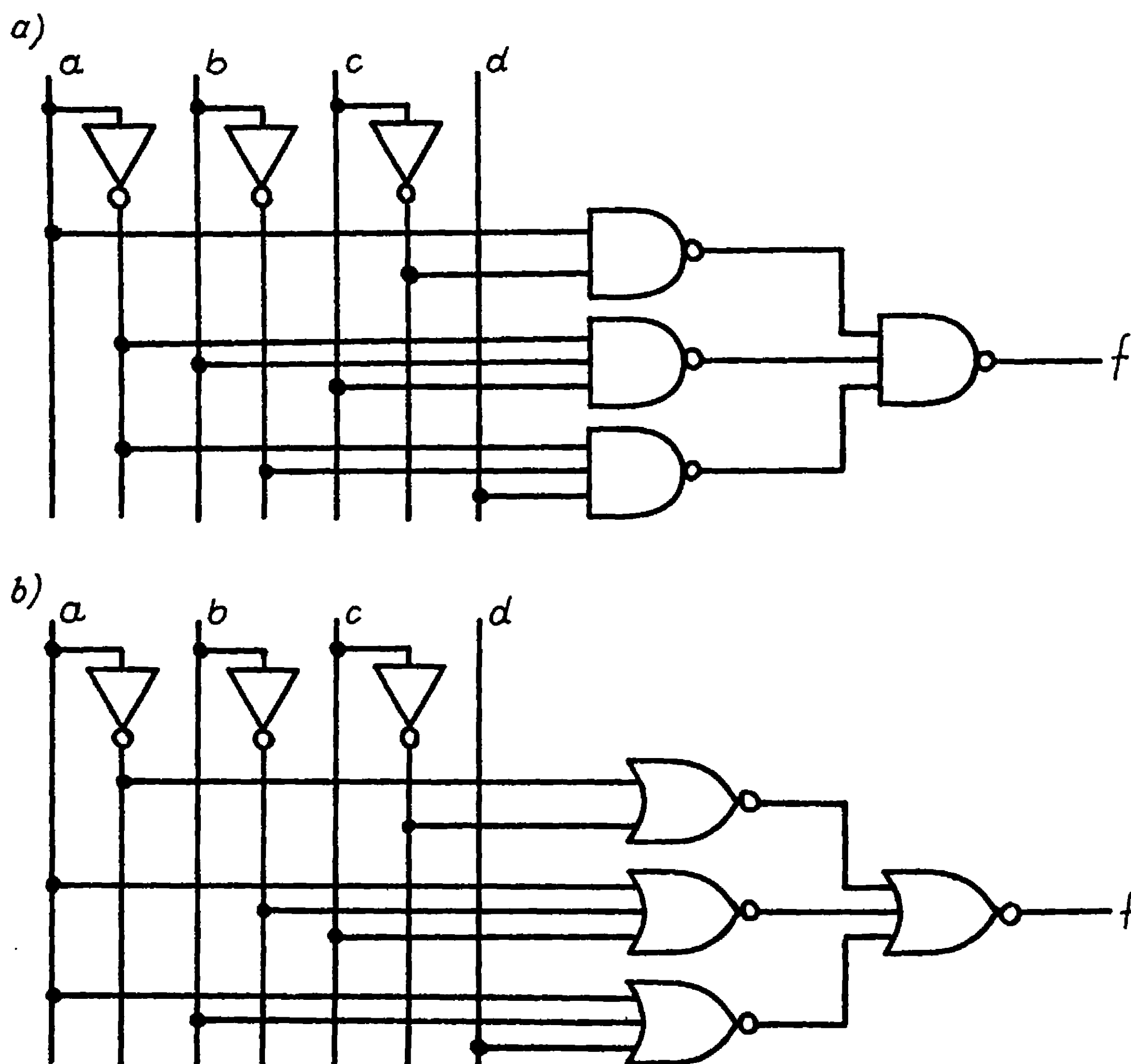
Koniunkcyjna postać normalna rozpatrywanej funkcji

$$f = (\overline{a} + \overline{c})(a + \overline{b} + c)(a + b + d),$$

daje się przekształcić po podwójnym zaprzeczeniu do formuły dostosowanej do realizacji z bramek NOR (rys. 2.29b)

$$\begin{aligned} f = \overline{\overline{f}} &= \overline{(\overline{a} + \overline{c})(a + \overline{b} + c)(a + b + d)} = \\ &= \overline{(\overline{a} + \overline{c}) + (a + \overline{b} + c) + (a + b + d)}. \end{aligned}$$

Algorytm przekształceń jest taki sam dla każdej postaci normalnej, a w wyniku otrzymujemy formułę nadającą się do bezpośredniej realizacji albo z bramek typu NAND, albo z bramek typu NOR. Bramki typu NAND oraz NOR są realizowane technologicznie w związku z czym omawiane przekształcenie jest bardzo użyteczne w zastosowaniach i to tym bardziej, że liczba poziomów bramek, jakie musi przejść sygnał w wyniku propagacji od wejścia do wyjścia, nie ulega zmianie. Nie ulega zmianie również wskaźnik kosztów (1) układu.



Rys. 2.29. Realizacje układu pokazanego na rys. 2.28 za pomocą bramek NAND (a) oraz NOR (b)

Dla każdego z wymienionych w punkcie 2.10 systemu funkcjonalnie pełnego można określić algorytm poszukiwania formuł minimalnych w ramach wyłącznie tego systemu. Algorytmy te mają jednak znaczenie głównie teoretyczne i nie znajdują zastosowania w aplikacjach. Pewnym wy-

jątkiem jest omawiany już system negacji iloczynu oraz negacji sumy (p. 2.10 systemy k), l)), a więc system NAND lub NOR. Jeśli zmienne wejściowe dostępne są zarówno w postaci afirmacyjnej, jak i zanegowanej, to minimalna formuła boolowska przekształcona wg praw de Morgana przechodzi również w minimalną formułę NAND lub NOR. Wskaźnik kosztu (1) nie obejmuje wówczas negatorów i nie ulega zmianie. Dla funkcji pokazanej na rysunku 2.28a wyniesie on $\langle 11, 4 \rangle$ i taki sam będzie dla postaci NAND pokazanej na rysunku 2.29a. Analogiczny rezultat otrzymamy dla postaci koniunkcyjnej (rys. 2.29b). Jeśli jednak zmienne dostępne są tylko w jednej z postaci (dla wygody przyjmijmy że wszystkie w afirmacyjnej), to minimalna formuła NAND lub NOR będzie, ogólnie rzecz biorąc, inna. Wiąże się to z możliwością dokonywania następujących przekształceń:

$$\begin{aligned} \text{a)} \quad & A \bar{B} + A \bar{C} = A (\bar{B} + \bar{C}) = A \overline{B C}, \\ \text{b)} \quad & A B \bar{C} = A B \overline{B C} = A B \overline{A B C} \end{aligned} \quad (3)$$

lub dualnie:

$$\begin{aligned} \text{a)} \quad & (A + \bar{B})(A + \bar{C}) = A + (\overline{B C}), \\ \text{b)} \quad & A + B + \bar{C} = A + B + (\overline{B C}) = A + B + (\overline{A + B + C}). \end{aligned} \quad (4)$$

Przekształcenia te faktoryzują postacie normalne, umożliwiając dokonanie minimalnej realizacji bramkowej funkcji logicznych dla systemu NAND lub NOR. Na przykład minimalna postać funkcji

$$f = w \times \bar{y} + w \times \bar{z} + y \times \bar{x},$$

poddana przekształceniu a)

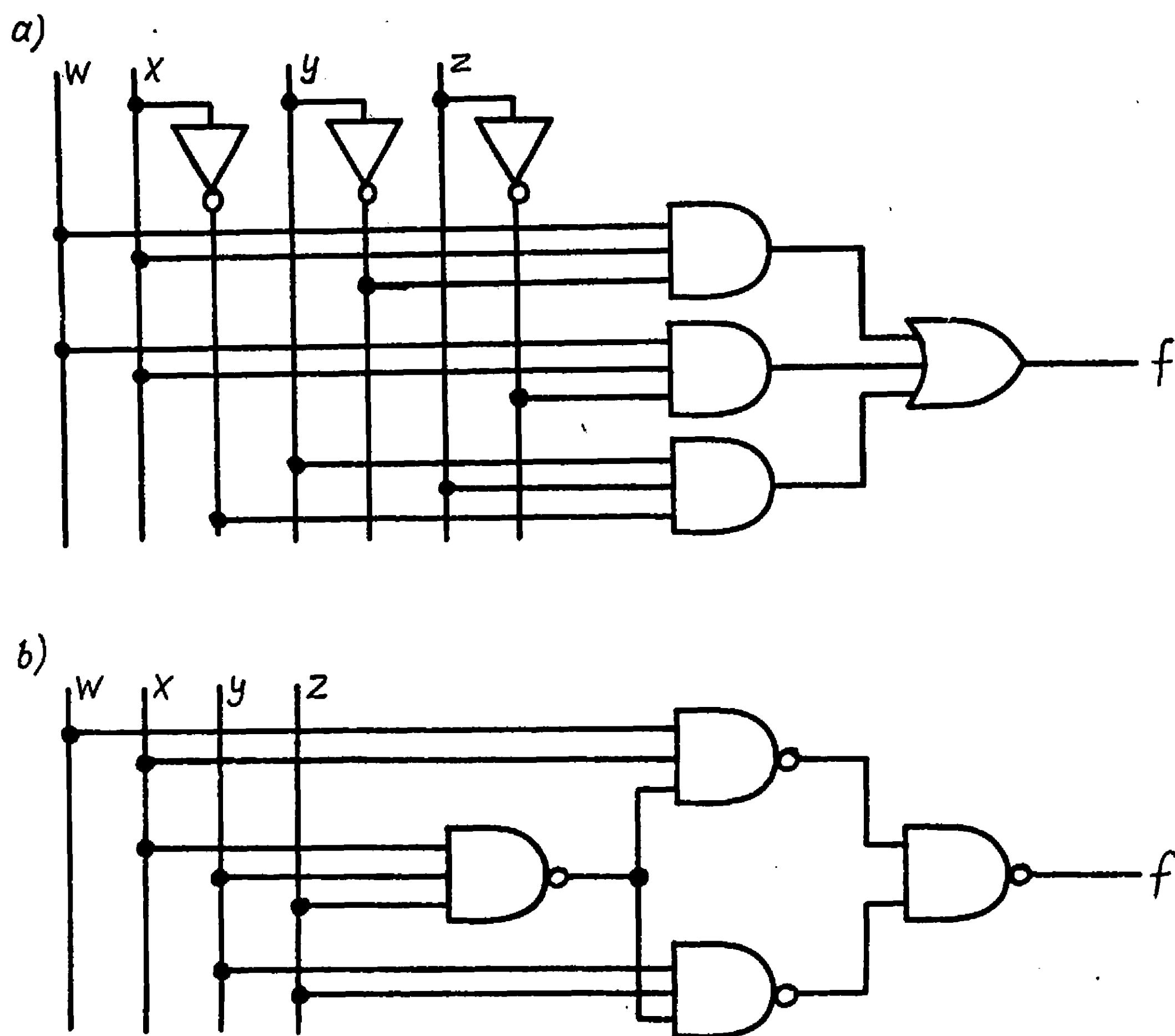
$$f = w \times \overline{y z} + y \times \bar{x},$$

zmniejsza liczbę literałów z 9 do 7. Dalszą redukcję schematu uzyskamy stosując przekształcenie b):

$$f = w \times \overline{y z} + y \times \bar{x} = w \times \overline{x y z} + y \times \overline{x y z},$$

$$\overline{\overline{f}} = \overline{\overline{w \times \overline{x y z} + y \times \overline{x y z}}}$$

Na rysunku 2.30 przedstawiono realizację omawianej funkcji bezpośrednio w postaci normalnej (rys. 2.30a) oraz zminimalizowanej NAND (rys. 2.30b). Wskaźnik kosztów (z negatorami) dla schematu a) wynosi $\langle 15, 7 \rangle$, natomiast dla schematu b): $\langle 11, 4 \rangle$. Jednocześnie nie uległa zmianie maksymalna liczba poziomów propagacji sygnału z wejścia na



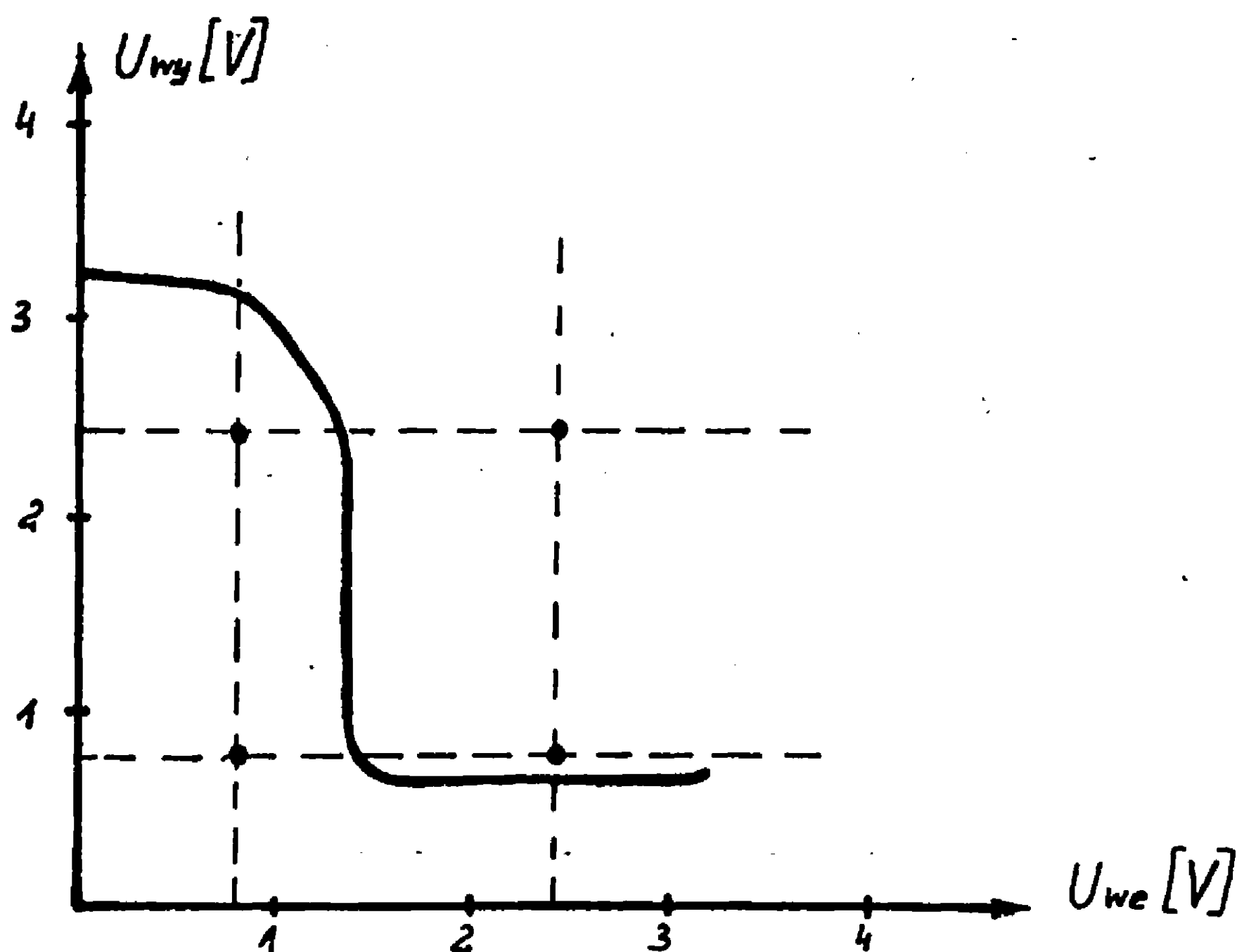
Rys. 2.30. Realizacje funkcji w postaci minimalnej normalnej (a) i w postaci minimalnej NAND (b)

wyjście (3 poziomy). Powyższy rezultat daje się również osiągnąć w wyniku postępowania algebraicznego. Algorytm ten może opracować w sposób autonomiczny dla systemu AND-NOT lub OR-NOT bądź też z wykorzystaniem dotychczasowych metod charakterystycznych dla definicji algebry podanej w punkcie 2.4. W ostatnim przypadku wykorzystuje się zależności (3) lub (4). Algorytm autonomiczny wymagałby zmiany całego dotychczasowego podejścia do formuł algebry Boole'a, ale algorytm wykorzystujący omówione już metody analityczne i zależności (3) lub (4) ma również postać bardzo złożoną. Jednocześnie, ranga użytkowa tego typu minimalizacji nie jest duża z uwagi na wzrost zastosowań układów scalonych średniej i dużej skali integracji oraz fakt powszechnej dostępności w systemach cyfrowych zmiennych, zarówno w postaci afirmacyjnej jak i zanegowanej. W związku z powyższym zagadnienie poszukiwań minimalnych formuł NAND lub NOR przy dostępnych jedynie afirmacjach zmiennych zostało tu jedynie zasygnalizowane i nie będzie dalej rozpatrywane.

3. UKŁADY KOMBINACYJNE

3.1. Uwagi ogólne

Jak wspomniano we wstępie układy kombinacyjne należą do tej klasy układów logicznych, dla której stan wejść jednoznacznie określa stan wyjść. Poszczególne wyjścia są funkcjami boolowskimi zmiennych wejściowych. Praca szerokiej klasy urządzeń technicznych z różnych dziedzin może być rozpatrywana w kategoriach kombinacyjnego układu logicznego. Ogólny opis analityczny jest niezależny od konkretnych realizacji, jednak charakter realizowanych zadań wiąże się ze specyfiką rozpatrywanej dziedziny, w związku z czym poszczególne układy często noszą nazwy charakterystyczne dla ich zastosowań.



Rys. 3.1. Charakterystyka przejściowa bramki TTL

W niniejszej pracy ograniczono się zasadniczo do układów logicznych mających zastosowanie w systemach cyfrowych. Są to głównie układy półprzewodnikowe małej i średniej skali integracji. Sygnały zera i je-

dynki logicznej reprezentują tu określone poziomy lub biegunowości napięć i prądów, ewentualnie przednie lub tylne zbocza impulsów prądowych lub napięciowych itp. Na rysunku 3.1 przedstawiono charakterystykę przejściową: napięcie wyjściowe w funkcji napięcia wejściowego standardowej bramki półprzewodnikowej TTL. Dla tych układów poziom jedynki logicznej odpowiada napięciom powyżej 2,4 V, natomiast poziom zera - poniżej 0,8 V. Między tymi poziomami bramka znajduje się w strefie tzw. dynamicznego przełączenia, które ze względów niezawodnościowych musi trwać bardzo krótko, zwykle kilka lub kilkanaście nanosekund. Zero i jedynkę logiczną można również przypisać podanym poziomom odwrotnie, tzn. "0" $\rightarrow$ 2,4 V oraz "1" $\rightarrow$ 0,8 V. Mówimy wówczas o tzw. logice negatywowej w odróżnieniu od przypadku poprzedniego, w którym poziomowi wysokiemu przypisano 1, natomiast poziomowi niskiemu 0, i który jest określany jako tzw. logika pozytywna.

Zagadnienia techniczne są formułowane w kategoriach określonych potrzeb użytkowych, które dopiero w procesie projektowania są tłumaczone na język teorii układów logicznych. Etap ten jest zwykle określany jako tzw. synteza właściwa, albo formalizacja logiczna zagadnienia aplikacyjnego. Należy podkreślić, że zasadniczo nie ma reguł algorytmicznych pozwalających dokonywać tego typu formalizacji - decydującą rolę odgrywa więc tutaj intuicja projektanta. Wprawdzie istnieją tzw. języki wyrażeń regularnych, z których bezpośrednio można dokonać syntezy właściwej, ale języki te są zbyt sformalizowane i ukierunkowane pod kątem badań poznawczych, co z punktu widzenia aplikacyjnego czyni je mało użytecznymi. Rezultatem formalizacji jest najczęściej tablica wartości logicznych, ewentualnie niezminimalizowana postać analityczna. Kolejnym etapem jest poszukiwanie minimalnej formuły boolowskiej i (lub) ewentualna faktoryzacja otrzymanego wyrażenia dostosowana do wymagań konkretnej realizacji. W następnych punktach zostaną omówione typowe układy mające zastosowanie w systemach cyfrowych.

3.2. Sumator jednobitowy

Postęp w dziedzinie konstrukcji maszyn liczących datuje się od momentu, gdy zauważono, że wyniki obliczeń są niezależne od podstawy systemu liczbowego, co umożliwiło wykorzystanie szczególnie prostych fizycznych analogów powszechnie stosowanego systemu binarnego. Takimi analogami są m.in. bramki logiczne wytwarzane techniką półprzewodniko-

wą. Sumator jednobitowy jest komórką kaskady układów tworzących sumator realizujący dodawanie wielobitowych liczb binarnych. Musi on posiadać trzy wejścia: dla bitu dodajnej, dodajnika i przeniesienia z niższej pozycji. Z kolei wynik sumowania jest realizowany na dwóch wyjściach: bitu sumy przypisanej pozycji sumatora jednobitowego w kaskadzie oraz bitu przeniesienia na pozycję wyższą. W oparciu o powyższe warunki oraz znajomość procedury sumowania liczb binarnych, można określić tablicę wartości logicznych sumatora (rys. 3.2). Bity dodajnej, dodajnika oraz przeniesienia z pozycji o niższej wadze, oznaczono literami a, b, c . Wyjście przeniesienia przez C (ang. carry) sumy przez S . Na rysunku 3.2 przedstawiono tablice Karnaugh dla obu wyjść. Na ich podstawie można określić minimalne formuły boolowskie:

$$C = a c + b c + a b,$$

$$S = \bar{a} \bar{b} c + \bar{a} b \bar{c} + a \bar{b} \bar{c} + a b c = \sum (1, 2, 4, 7).$$

a)

a	b	c	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

b)

$a \backslash bc$	00	01	11	10
0	0	0	1	0
1	0	1	1	1

c)

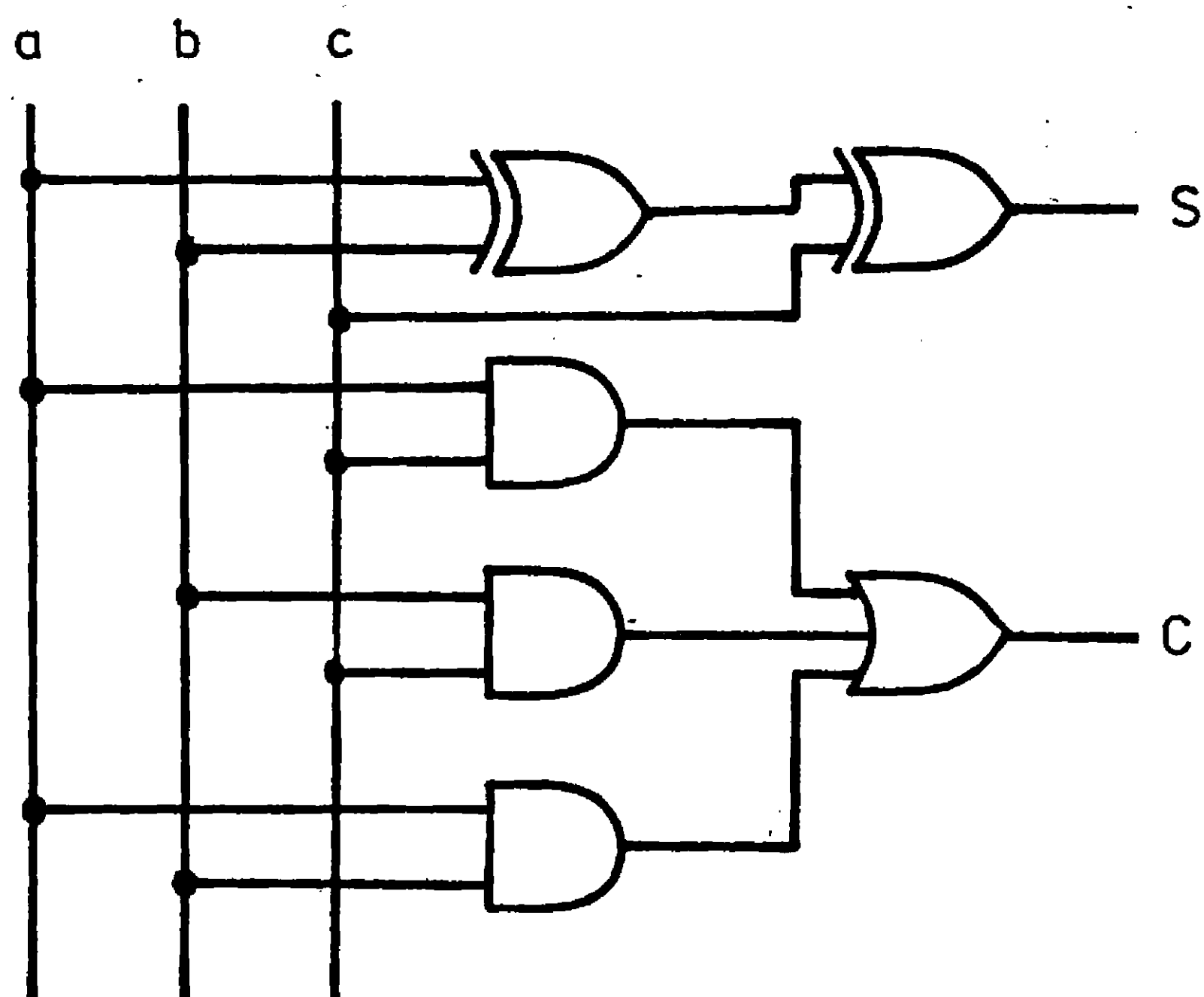
$a \backslash bc$	00	01	11	10
0	0	1	0	1
1	1	0	1	0

Rys. 3.2. Tablica wartości logicznych (a) i tablice Karnaugh sumatora jednobitowego (b,c)

Funkcja S nie daje się minimalizować, jej postać kanoniczna jest jednocześnie postacią minimalną. Jeśli jednak poddać tę funkcję przekształceniom (faktoryzacji), można uzyskać następujący rezultat:

$$\begin{aligned}
 S &= \bar{a}(\bar{b} c + b \bar{c}) + a(\bar{b} \bar{c} + b c) = \bar{a}(b \oplus c) + a(b \odot c) = \\
 &= \bar{a}(b \oplus c) + a \overline{(b \oplus c)} = a \oplus b \oplus c.
 \end{aligned}$$

Jest to operacja EXCLUSIVE-OR dla trzech argumentów. Na rysunku 3.3 przedstawiono realizację bramkową sumatora jednobitowego. Z uwagi na to, że zwykle nie buduje się bramek XOR trójwejściowych do realizacji sumy zastosowano dwie bramki dwuwejściowe.



Rys. 3.3. Realizacja bramkowa sumatora jednobitowego

Na rysunku 3.4 przedstawiono tabelę wartości logicznych, dla przypadku sumowania liczb z różnymi znakami (odejmowania). Zmienne a, b, c oznaczają kolejne: odjemną, odjemnik i pożyczkę z dolnej pozycji. Wyjście D (ang. difference) jest wyjściem różnicy, natomiast B (ang. borrow) pożyczki z wyższej pozycji. Porównując tabele podane na rysunkach 3.2 i 3.4 widzimy, że funkcja $D = S$, natomiast minimalna postać B jest następująca (rys. 3.5):

a	b	c	B	D
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

$B = \bar{a}c + \bar{a}b + bc.$

Rys. 3.4. Tabele wartości logicznych jednobitowego układu odejmującego

Funkcja B różni się od funkcji przeniesienia C tylko występowaniem odjemnej w postaci zanegowanej. Podanie więc na wejścia bramek tworzących przeniesienie w sumatorze jednobitowym negacji dodajnej (odjemnej), powoduje, że sumator realizuje różnicę jednobitową.

a \ bc				
	00	01	11	10
0	0	1	1	1
1	0	0	1	0

Rys. 3.5. Tablica Karnaugh dla wyjścia pożyczki układu odejmującego

3.3. Komparator

W systemach liczących bardzo często decyzje są podejmowane zależnie od wyniku porównania między liczbami binarnymi reprezentowanymi przez ciągi zero-jedynkowe. Kombinacyjne układy logiczne dokonujące

a_1	a_0	b_1	b_0	$(A < B)$	$(A = B)$	$(A > B)$
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

Rys. 3.6. Tablica wartości logicznych komparatora liczb dwubitowych

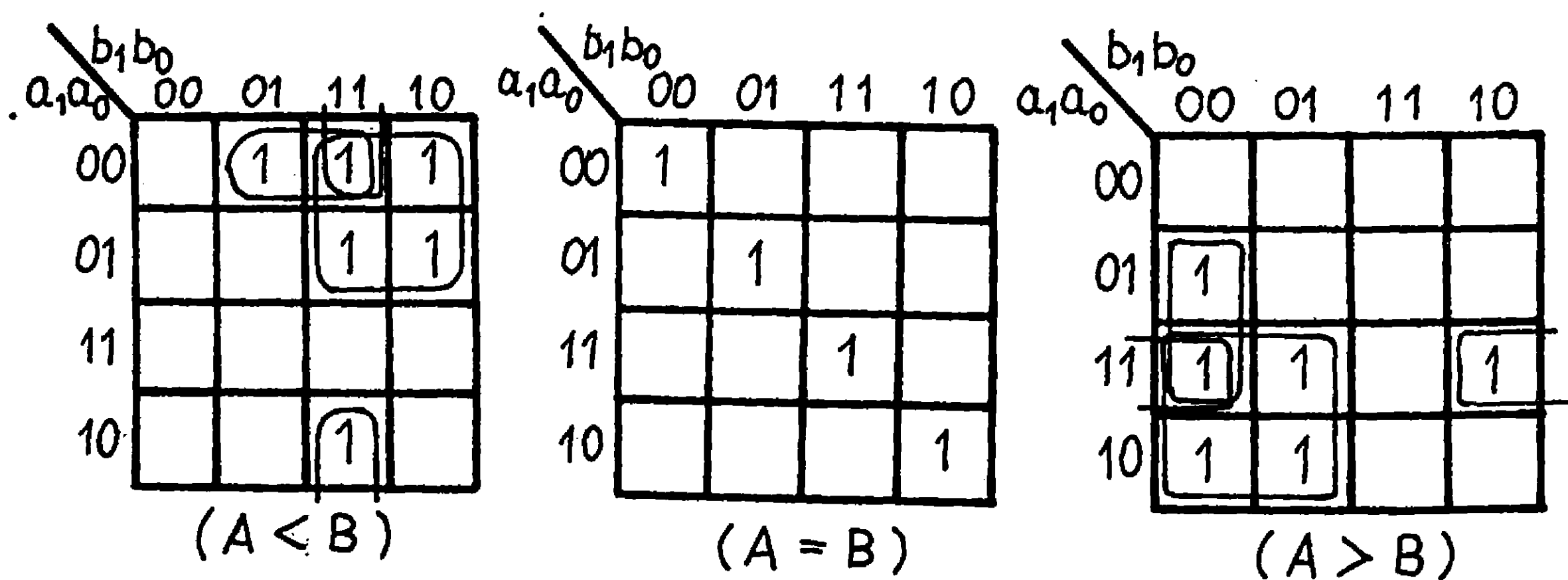
takiego porównania noszą nazwę komparatorów cyfrowych. Najprostszym przykładem komparatora cyfrowego jest układ porównujący dwie liczby binarne dwubitowe $A(a_1, a_0)$ i $B(b_1, b_0)$. Zmienne a_1, a_0, b_1, b_0 reprezentują wagi w zapisie liczby i są jednocześnie zmiennymi wejściowymi. Komparator posiada trzy wyjścia (rys. 3.6), na których pojawia się logiczna jedynka zależnie od tego, czy $A > B$, $A = B$ lub $A < B$. Na rysunku 3.6 przedstawiono tablicę wartości logicznych omawianego komparatora. Optymalną realizację poszczególnych wyjść uzyskamy korzystając z tablic Karnaugh przedstawionych na rysunku 3.7, w które dla uproszczenia wpisano tylko jedynki. Puste kratki reprezentują tym samym zera funkcji. Na podstawie tablic można napisać dla poszczególnych wyjść następujące formuły minimalne:

$$(A < B) = \bar{a}_1 b_1 + \bar{a}_1 \bar{a}_0 b_0 + \bar{a}_0 b_1 b_0.$$

$$(A = B) = \sum (0, 5, 10, 15),$$

$$(A > B) = a_1 \bar{b}_1 + a_0 \bar{b}_1 \bar{b}_0 + a_1 a_0 \bar{b}_0.$$

Wyjście równościowe nie daje się zminimalizować, w związku z czym formułę zapisano w postaci kanonicznej. Zwraca uwagę symetria tablic względem A i B oraz fakt, że trójwymiarowy wektor wyjściowy, ze zbioru ośmiu wartości przyjmuje tylko trzy: 001, 010, 100. Wektor wejściowy układu jest określony dla wszystkich szesnastu czterobitowych wartości.



Rys. 3.7. Tablice Karnaugh dla wyjść komparatora liczb dwubitowych

Dla liczb dwubitowych do wejścia komparatora należy doprowadzić cztery sygnały, ale dla trójbitych już sześć. Rozmiary tablicy wartości logicznych rosną i wyłania się problem ograniczenia procedury dalszej syntezy ze względu na formę tego typu opisu. Jeśli zrezygnować z minimalizacji formuł, funkcje wyjściowe można ułożyć bezpośrednio śledząc proces komparacji. Dla liczb trójbitych $A(a_2, a_1, a_0)$ oraz $B(b_2, b_1, b_0)$ dla wyjścia równościowego otrzymamy

$$(A = B) = (a_2 \odot b_2)(a_1 \odot b_1)(a_0 \odot b_0).$$

Dla wyjść $(A < B)$ oraz $(A > B)$, formułę należy ułożyć dokonując porównania kolejno, poczynawszy od pozycji o najwyższej wadze:

$$(A < B) = \bar{a}_2 b_2 + \bar{a}_1 b_1 (a_2 \odot b_2) + \bar{a}_0 b_0 (a_2 \odot b_2)(a_1 \odot b_1).$$

$$(A > B) = a_2 \bar{b}_2 + a_1 \bar{b}_1 (a_2 \odot b_2) + a_0 \bar{b}_0 (a_2 \odot b_2)(a_1 \odot b_1).$$

Według tego algorytmu można tworzyć formuły dla dowolnej liczby zmiennych, jednak nie są one minimalne, a ponadto liczba poziomów propagacji sygnału nie ulega zmianie tylko wówczas, gdy są do dyspozycji dodatkowe bramki równoważności (XNOR, rys. 2.10).

3.4. Układy iteracyjne

Teoretycznie każdy układ kombinacyjny daje się zrealizować w sposób optymalny w oparciu o tablicę wartości logicznych. Jednak dla większej liczby zmiennych rozmiary tablicy są tak duże, że nie daje się z niej bezpośrednio korzystać. Konieczne jest więc zastosowanie maszyn liczących, ale dla dużej liczby zmiennych struktura układu staje się, mimo minimalizacji, bardzo złożona i niejednorodna, co utrudnia analizę i diagnostykę. Jest to bardzo poważna wada konstrukcyjna, w związku z czym w procesie syntezy właściwej dokonuje się podziału całości lub części projektowanego systemu kombinacyjnego na bloki złożone z kaskady wzajemnie połączonych jednakowych układów o niezbyt złożonej strukturze. Wprawdzie podział taki zmniejsza szybkość pracy bloku z uwagi na konieczność wielopoziomowej propagacji sygnału poprzez poszczególne stopnie, jednak w wielu zastosowaniach ograniczenie to nie jest decydujące. Tak zdekomponowany blok kombinacyjny nosi nazwę bloku iteracyjnego,

a elementarna struktura - układu iteracyjnego lub kaskadowego. Zależnie od sposobu łączenia, układy iteracyjne można dzielić na jedno i wielowymiarowe, przy czym pojęcie wymiarowości wiąże się ze wzrostem liczby układów proporcjonalnie do odpowiedniej potęgi liczby wektorów zmiennych wejściowych.

3.4.1. Jednowymiarowe układy iteracyjne

Na rysunku 3.8 przedstawiono schemat jednowymiarowego bloku iteracyjnego złożonego z n elementarnych identycznych układów kombinacyjnych, zwanych także komórkami iteracyjnymi. Wejścia X_i oraz F_i oraz wyjścia Y_i , F_{i+1} ($i = 0, 1, \dots, n-1$) mogą w ogólnym przypadku być wektorami o dowolnej wymiarowości z tym, że wymiarowość F_i oraz F_{i+1} musi być taka sama. Wektor F_i (F_{i+1}) nosi nazwę wektora stanu, gdyż zawiera on informację o stanie wszystkich $i-1$ bloków, bowiem:

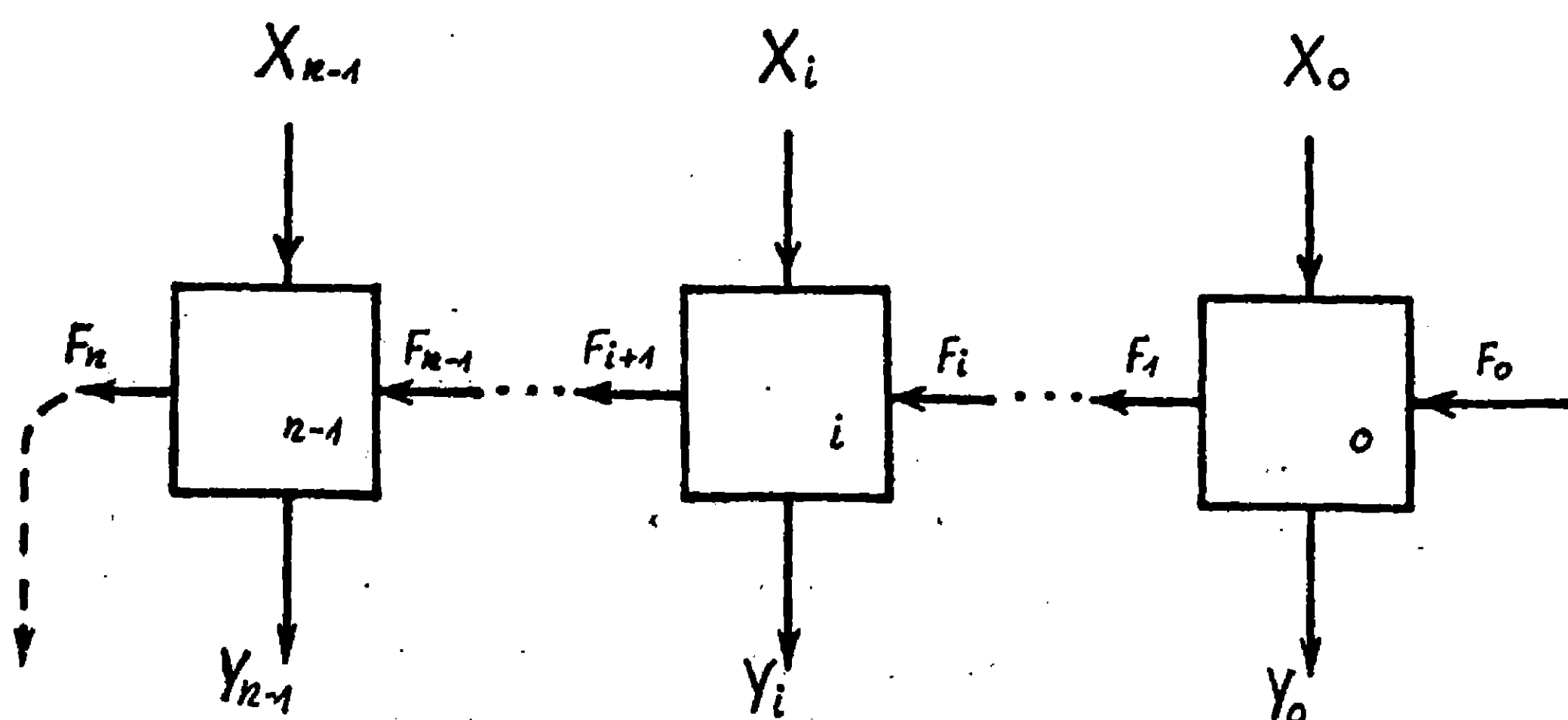
$$F_i = F_i(x_{i-1}, F_{i-1}),$$

$$F_{i-1} = F_{i-1}(x_{i-2}, F_{i-2}),$$

$$\dots\dots\dots$$

$$F_1 = F_1(x_0, F_0),$$

przy czym $\max(i) = n$.



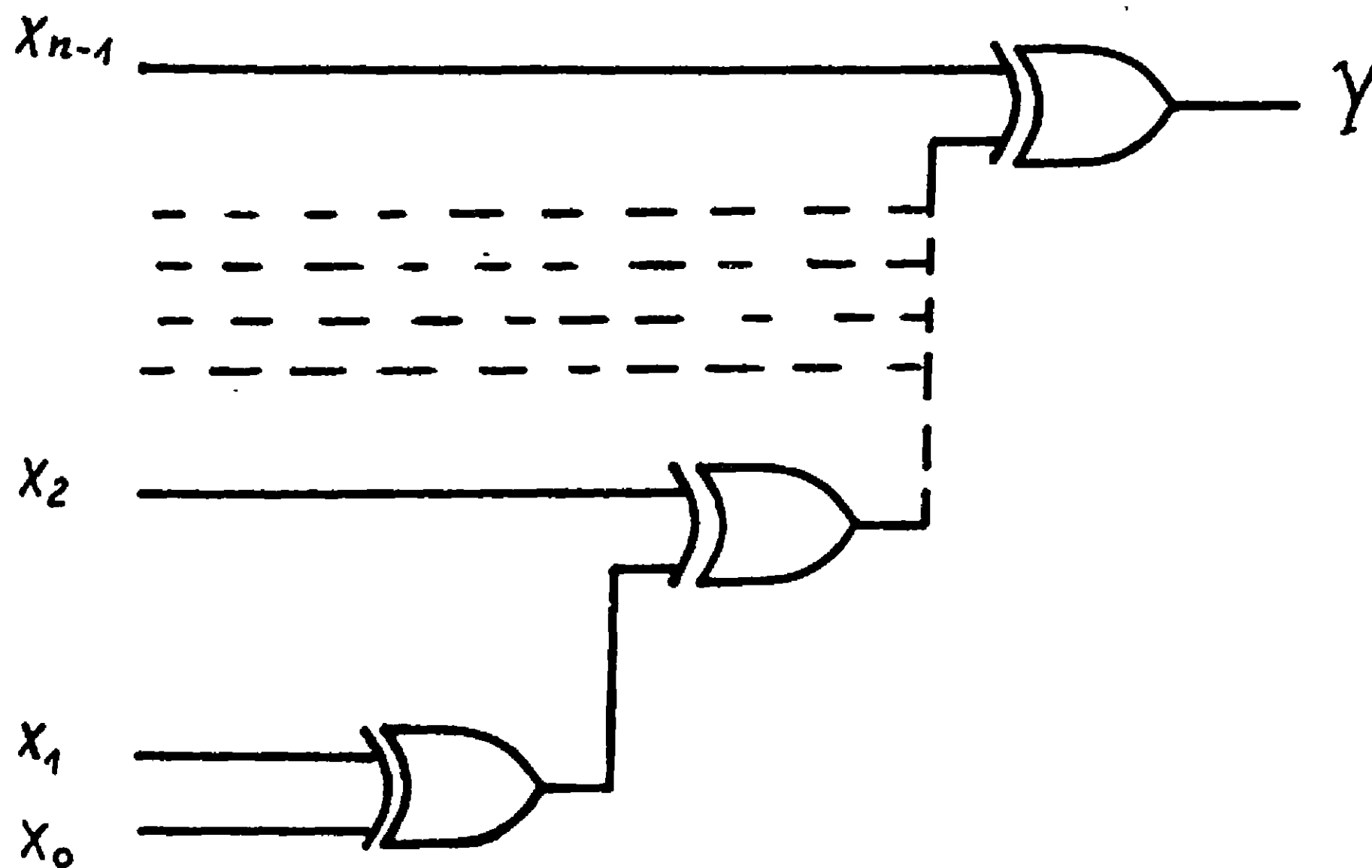
Rys. 3.8. Schemat jednowymiarowego bloku iteracyjnego

Wektory X_i , Y_i są klasycznymi wektorami wejściowymi i wyjściowymi układu kombinacyjnego. Warunki pracy kaskady traktowanej jako całość, mogą narzucić inną strukturę, zarówno sterowania jak i budowy dla bloków skrajnych. Precyzują to warunki brzegowe, które należy uwzględnić w procesie syntezy właściwej podczas dekompozycji iteracyjnej problemu projektowego.

Przykład 3.1. Zbudować układ sumatora n -bitowego.

Struktura blokowa sumatora odpowiada dokładnie strukturze ogólnej przedstawionej na rysunku 3.8, natomiast elementarna komórka została już opisana w punkcie 3.2. Wektor wejściowy ma trzy składowe: a_i , b_i , c_i (rys. 3.2); wektor wyjściowy jedną: S_i ; wektor stanu również jedną: C_{i+1} . Procedura sumowania narzuca następujące warunki na brzegach: $C_0 = 0$; $C_n = S_n$ (wynik sumowania może być $n+1$ -bitowy).

Przykład 3.2. Zbudować układ kontroli parzystości liczby jedynek w ciągu n zmiennych binarnych.



Rys. 3.9. n -Bitowy układ kontroli parzystości

Aby rozwiązać ten problem wystarczy zauważyć, że operacja EXCLUSIVE-OR dla n zmiennych (p. 2.11) daje w wyniku 1 dla nieparzystej liczby jedynek. Brak parzystości będzie więc sygnalizowany pojawieniem się "1" na wyjściu ostatniej bramki kaskady dwuwejściowych bramek XOR

(rys. 3.9), przy czym elementarnym układem iteracyjnym będzie właśnie ta bramka. Zmienna stanu jest propagowana przez wyjścia i wejścia bramek, aż do bramki końcowej, której wyjście staje się wyjściem wynikowym. Dla elementarnej struktury (bramki XOR), wektory wejściowy i stanu są jednowymiarowe, natomiast brak jest wyjścia wynikowego, które pojawia się dopiero dla bramki brzegowej o najwyższej wadze. Brak wyjścia wynikowego można utożsamić z zerową wymiarowością wektora wyjściowego i wtedy układ pokazany na rysunku 3.9 staje się zgodny z układem ogólnym (rys. 3.8).

Przykład 3.3. Zaprojektować n wejściowy układ iteracyjny, na którego wyjściach pojawi się sygnał: 00 - gdy nie ma zer na wejściach, 01 - gdy na wejściach jest jedno zero, 10 - gdy na wejściach są dwa zera, 11 - gdy na wejściach jest większa liczba zer.

Z treści zadania wynika, że układ posiada dwa wyjścia, które oznaczmy literami y, z. Elementarna komórka iteracyjna musi posiadać informację o stanie jej wejścia i łącznej liczbie zer podanych na wejście komórek o mniejszych indeksach. Do przeniesienia tej informacji wystarczą dwa wejścia dla zmiennych stanu, przy czym na końcu wyjścia te utożsamiają się z wyjściami wynikowymi. W tabeli pokazanej na rysunku 3.10 zmienne y_i , z_i kodują zgodnie z warunkami zadania informacje o liczbie zer podanych na wejścia od indeksu $i-1$ do 0. Jeśli do i -tej pozycji dochodzi zero, to na wyjściach y_{i+1} , z_{i+1} musi być to uwzględnione. Jeśli $x_i = 1$, to wyjścia stanu pozostają bez zmian, tzn. $y_i = y_{i+1}$ oraz $z_i = z_{i+1}$. Z tablic Karnaugh podanych na rysunku 3.11 wynikają następujące formuły minimalne dla funkcji wyjściowych:

$$y_{i+1} = y_i + \bar{x}_i z_i, \quad (i = 0, 1, \dots, n-1),$$

lub

$$y_{i+1} = (y_i + z_i)(\bar{x}_i + y_i),$$

oraz

$$z_{i+1} = \bar{x}_i \bar{z}_i + \bar{x}_i y_i + x_i z_i,$$

lub

$$z_{i+1} = (\bar{x}_i + z_i)(x_i + y_i + \bar{z}_i).$$

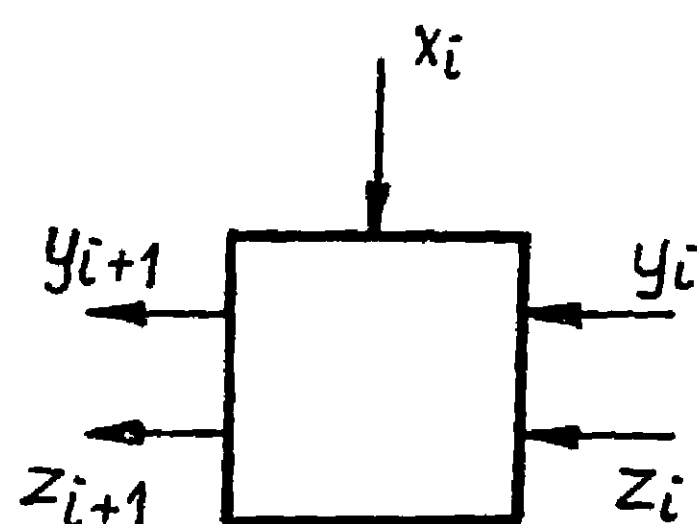


Formuły te można realizować bezpośrednio lub też przekształcić je do postaci nadającej się do realizacji z bramek NOR lub NAND:

$$y_{i+1} = \overline{\overline{y_i} \overline{x_i z_i}}; \quad y_{i+1} = \overline{(y_i + z_i) + (\overline{x_i} + y_i)},$$

$$z_{i+1} = \overline{\overline{x_i z_i} \overline{x_i y_i} \overline{x_i z_i}}; \quad z_{i+1} = \overline{(\overline{x_i} + z_i) + (x_i + y_i + \overline{z_i})}.$$

x_i	y_i	z_i	y_{i+1}	z_{i+1}
0	0	0	0	1
0	0	1	1	0
0	1	0	1	1
0	1	1	1	1
1	0	0	0	0
1	0	1	0	1
1	1	0	1	0
1	1	1	1	1



Rys. 3.10. Tablica wartości logicznych układu z przykładu 3.3

$x_i \backslash y_i z_i$	00	01	11	10
0	0	1	1	1
1	0	0	1	1

y_{i+1}

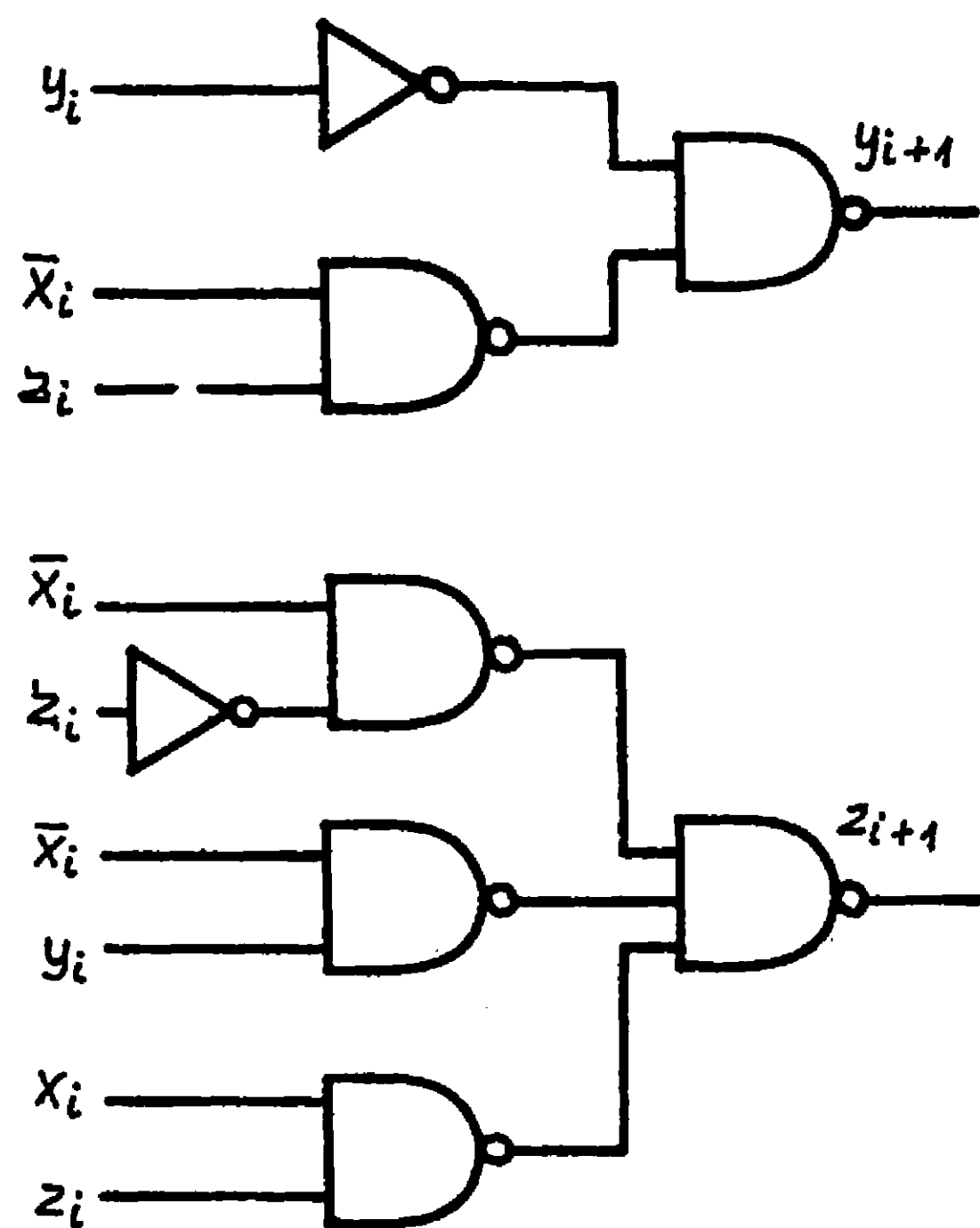
$x_i \backslash y_i z_i$	00	01	11	10
0	1	0	1	1
1	0	1	1	0

z_{i+1}

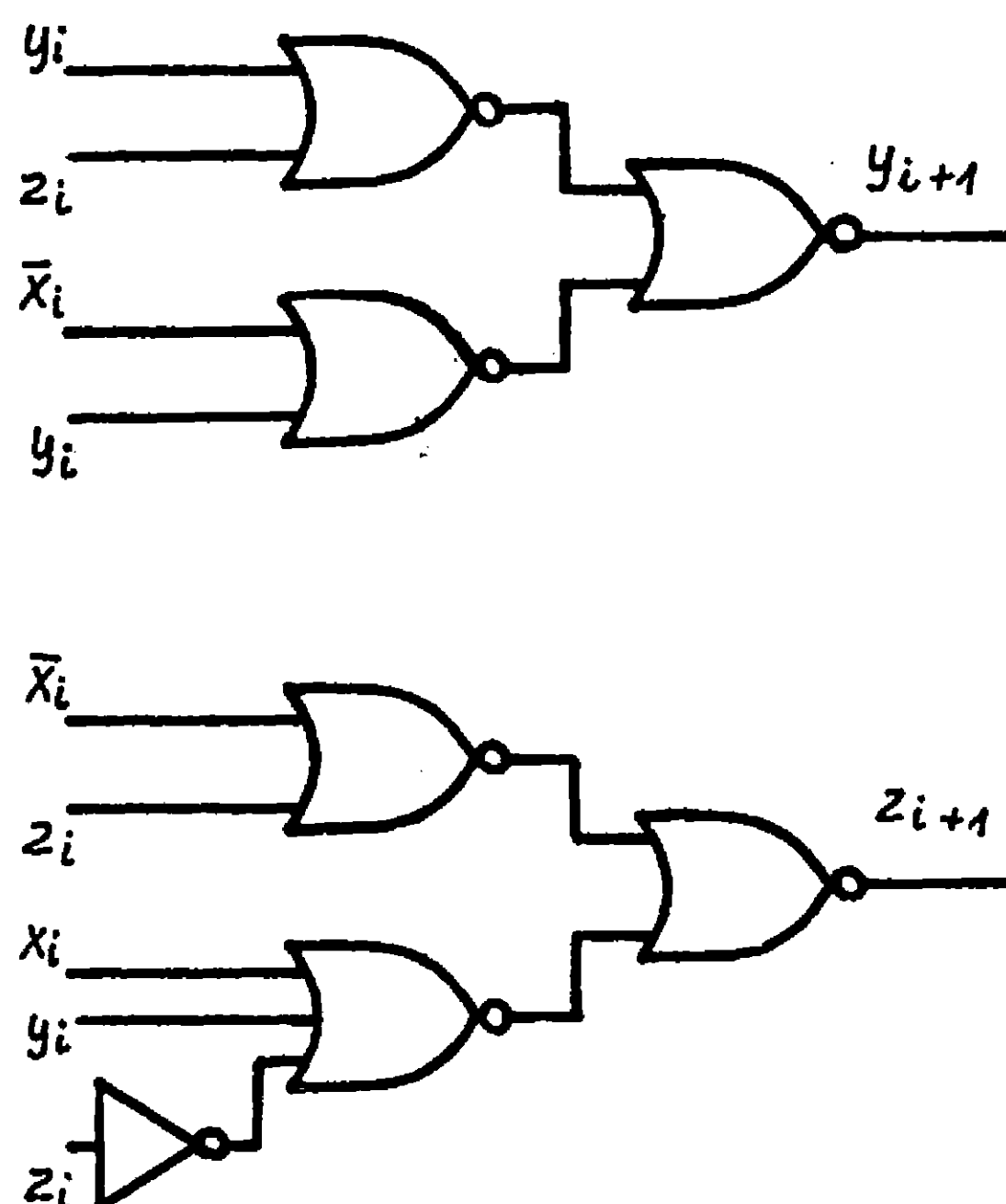
Rys. 3.11. Tablice Karnaugh dla wyjść układu z przykładu 3.3

Na rysunku 3.12 podano obie realizacje. Dla uproszczenia opuszczono połączenia między wejściami, zastępując je literałami oraz założono, że zmienna wejściowa są dostępne zarówno w postaci afirmacyjnej jak i negowanej. Warunki brzegowe: $y_0 = 0$, $z_0 = 0$.

a)



b)



Rys. 3.12. Realizacja z bramek NAND oraz NOR komórki iteracyjnej z przykładu 3.3 (a,b)

Przykład 3.4. Zaprojektować komparator dwu liczb n -bitowych.

Omówiony w punkcie 3.3 układ komparatora wygodnie jest zdekomponować do postaci iteracyjnej. Zamiast trzech wyjść, a mianowicie dwóch nierównościowych i jednego równościowego, wystarczy ograniczyć się do dwu, gdyż dwuwymiarowy wektor o składowych binarnych ma aż cztery różne wartości, którym można przypisać odpowiedni wynik częściowej komparacji. Tego typu proces nosi nazwę kodowania informacji. Jest to tu uzasadnione, gdyż dla długiej kaskady połączeń zmniejszenie liczby wyjść zmniejsza koszt układu. Kodując:

$(A = B) \rightarrow 11$

$(A < B) \rightarrow 01$

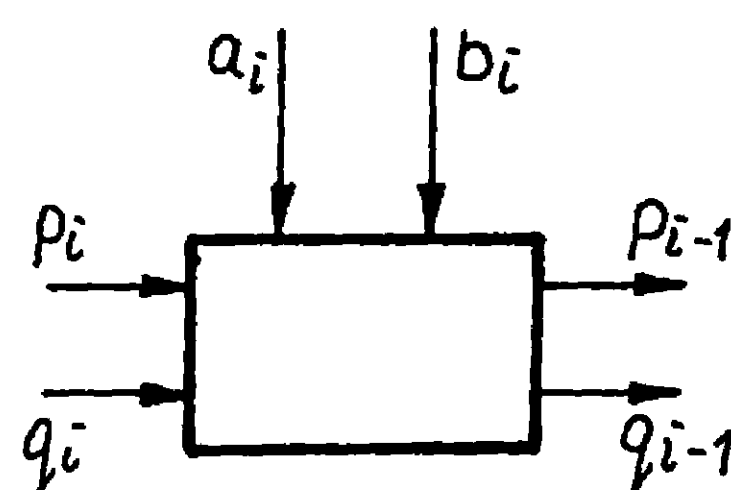
$(A > B) \rightarrow 10$

oraz wprowadzając symbole p, q dla wyjść stanu dla elementarnego układu otrzymamy tabelę przedstawioną na rysunku 3.13. Kierunek komparacji przebiega od cyfr o najwyższej wadze, dlatego zmieniono kolejność in-

dekowania wyjść zmiennych stanu. Zgodnie z założeniem stan wejść $p = 0$ i $q = 0$ nie jest przypisany procesowi komparacji. Można to wykorzystać pozostawiając swobodę projektantowi w procesie minimalizacji funkcji. Zależnie od potrzeb wartości nieokreślone poszczególnych zmiennych może potraktować bądź jako jedynki bądź jako zera, osiągając tym samym jeszcze prostszą formułę. W tablicy pokazanej na rysunku 3.13, wyjścia stanu pozostają dla $p_i = 0$ i $q_i = 0$ - nieokreślone, co znalazło swój wyraz w tablicach Karnaugh, pokazanych na rysunku 3.14. Kolumny $p_i, q_i \rightarrow 00$ mogą być wykorzystane bądź jako zera bądź jako jedynki. Jeśli dla wyjścia p_{i-1} stanom nieokreślonym nadać wartość 1, to funkcja będzie mieć postać:

$$p_{i-1} = \bar{q}_i + a_i p_i + \bar{b}_i p_i,$$

a_i	b_i	p_i	q_i	p_{i-1}	q_{i-1}
0	0	0	0	X	X
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	1
0	1	0	0	X	X
0	1	0	1	0	1
0	1	1	0	1	0
0	1	1	1	0	1
1	0	0	0	X	X
1	0	0	1	0	1
1	0	1	0	1	0
1	0	1	1	1	0
1	1	0	0	X	X
1	1	0	1	0	1
1	1	1	0	1	0
1	1	1	1	1	1



Rys. 3.13. Tablica wartości logicznych układu komparatora w wersji iteracyjnej (przykład 3.4)

$a_i b_i$	$p_i q_i$			
	00	01	11	10
00	X	0	1	1
01	X	0	0	1
11	X	0	1	1
10	X	0	1	1

p_{i-1}

$a_i b_i$	$p_i q_i$			
	00	01	11	10
00	X	1	1	0
01	X	1	1	0
11	X	1	1	0
10	X	1	0	0

q_{i-1}

Rys. 3.14. Tablice Karnaugh układu komparatora (przykład 3.4)

Dla samych zer wstawionych w kolumnę 00 otrzymamy natomiast

$$p_{i-1} = p_i (a_i + \bar{b}_i + \bar{q}_i).$$

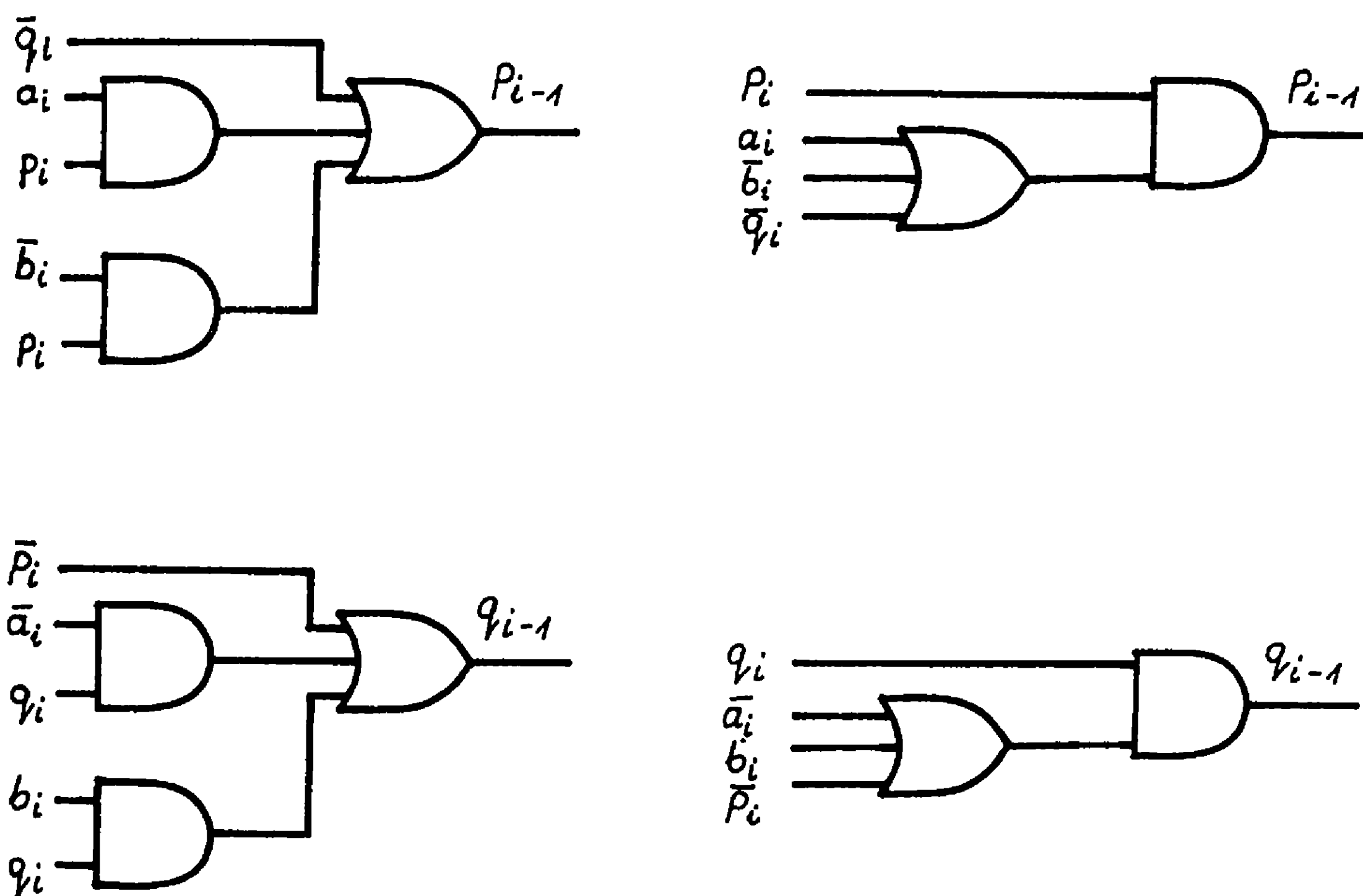
Funkcje te nie są algebraicznie równoważne, jednak z uwagi na fakt, że $p_i + q_i \neq 0$, spełniają w układzie tę samą rolę. Podobnie można napisać dla wyjścia q_{i-1} :

$$q_{i-1} = \bar{p}_i + \bar{a}_i q_i + b_i q_i \quad (i = 1, 2, \dots, n),$$

oraz

$$q_{i-1} = q_i (\bar{a}_i + b_i + \bar{p}_i).$$

Otrzymane formuły są prostsze od tych, które można by było osiągnąć bez uwzględniania kolumny nieokreślonej. Jednak w tego typu procedurze trzeba mieć pewność, że stan nieuwzględniany nie wystąpi. W rozpatrywanym przypadku pewność tę osiągamy podając na wejścia p_n oraz q_n komórki o najwyższej wadze, sygnały brzegowe $p_n = 1$ oraz $q_n = 1$. Sygnałami wynikowymi są sygnały otrzymane na wyjściach q_0 i p_0 komórki pierwszej, dokonującej porównania cyfr o najniższej wadze. Na rysunku 3.15 przedstawiono uproszczony schemat logiczny komórki komparatora zrealizowanej dla obu wariantów wykorzystania stanów nieokreślonych. Sumaryczny wskaźnik kosztów po wliczeniu negatorów dla układu a) wynosi $\langle 18, 10 \rangle$, natomiast dla układu b) $\langle 14, 8 \rangle$, a więc układ ten jest konstrukcyjnie tańszy.



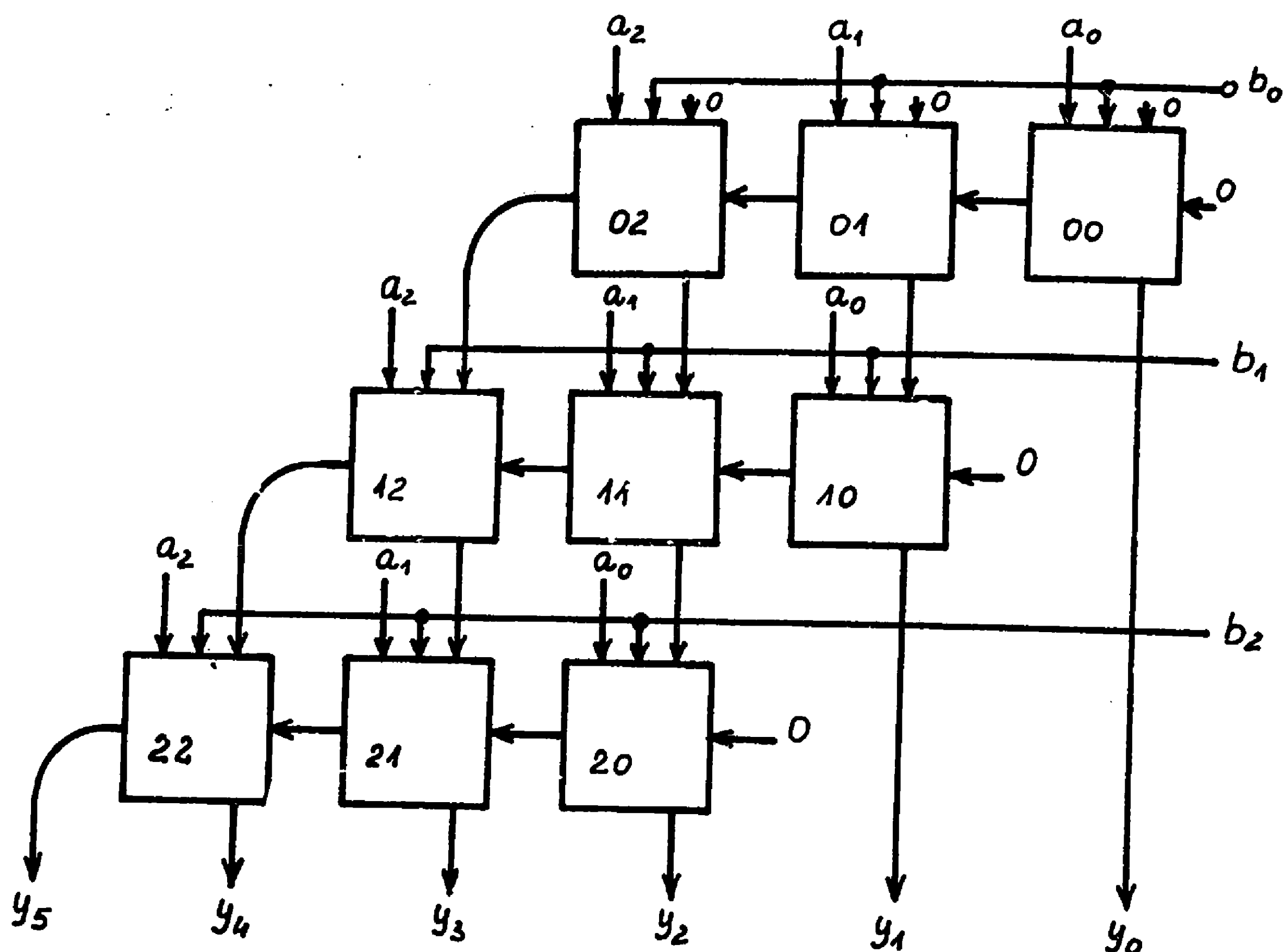
Rys. 3.15. Realizacja komórki iteracyjnej komparatora (przykład 3.4)

3.4.2. Wielowymiarowe układy iteracyjne

Wielowymiarowy blok iteracyjny jest prostym rozwinięciem układu jednowymiarowego. Potrzeba wprowadzenia tego typu bloków wynika z faktu, że wiele zagadnień technicznych nie daje się zdekomponować do postaci blokowej przedstawionej na rysunku 3.8. Przykładem może tu być n-bitowy układ mnożący, dający się zrealizować tylko w oparciu o dwuwymiarową strukturę łączy poszczególnych komórek iteracyjnych, bądź też układ porządkujący m nieujemnych liczb n-bitowych wg kolejności wzrastania, możliwy do zrealizowania również przez strukturę dwuwymiarową.

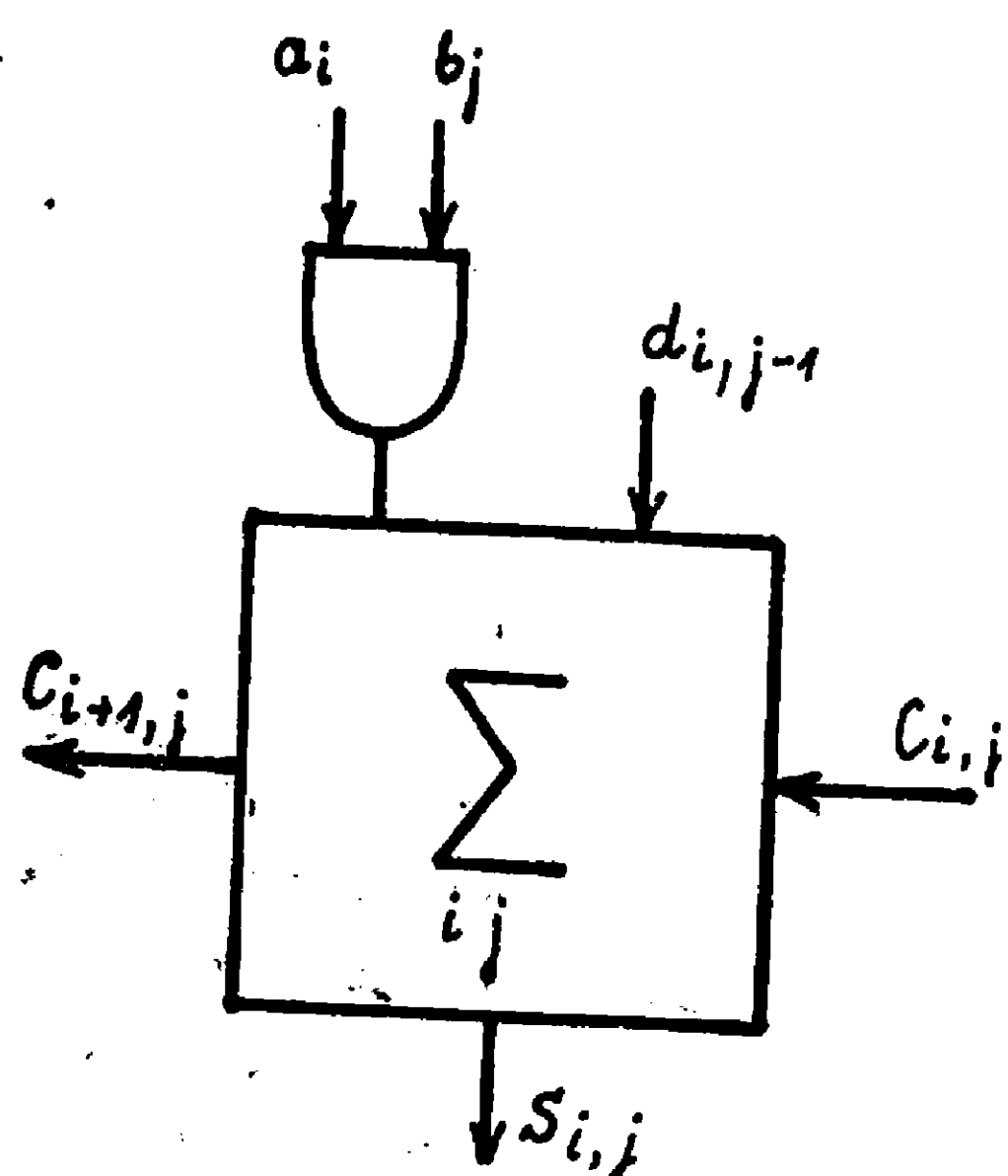
Przykład 3.5. Zrealizować blok iteracyjny realizujący mnożenie liczb n-bitowych.

Aby dokonać dekompozycji problemu zauważmy, że mnożenie ogólnie polega na sumowaniu wyników częściowych przesuniętych względem siebie o jedną pozycję. Dla liczb binarnych wynikiem częściowym jest mnożna - w wypadku gdy cyfra liczby mnożnika odpowiadającej danej pozycji wynosi 1, bądź zero gdy cyfra ta ma wartość 0. Schemat blokowy układu



Rys. 3.16. Schemat iteracyjny bloku mnożącego dwie liczby trójbitowe; zaznaczono indeksy porządkowe (przykład 3.5)

mnożącego przedstawiono dla przypadku liczb trójbitowych $A(a_2 a_1 a_0)$, $B(b_2 b_1 b_0)$ na rysunku 3.16, natomiast na rysunku 3.17 przedstawiono



Rys. 3.17. Schemat komórki iteracyjnej układu mnożącego (przykład 3.5)

elementarną komórkę iteracyjną. Jest nią omówiony już sumator jednobitowy z tym, że dodajna jest realizowana jako koniunkcja i -tej cyfry liczby A z j -tą cyfrą liczby B . Warunki brzegowe i technikę łączeń podano na rysunku 3.16. Z rysunku tego również daje się określić maksymalną liczbę komórek iteracyjnych przez jaką musi przejść sygnał w stanie przejściowym ustalania wyniku; wynosi ona 7. Liczba ta charakteryzuje maksymalne opóźnienie wnoszone przez blok.

3.5. Konwertery kodu

W wielu zastosowaniach za pomocą ciągów zero-jedynkowych koduje się informację przesyłaną do poszczególnych bloków systemu cyfrowego. W kategoriach układu logicznego kodowanie polega na przypisaniu poszczególnym wartościom wektora o n składowych binarnych pewnych informacji, przy czym maksymalna pojemność informacyjna dla n -bitów wynosi 2^n . W procesie przetwarzania informacji zachodzi często potrzeba zmiany kodu i rolę tę spełniają kombinacyjne układy logiczne zwane konwerterami kodu.

Cyfra dziesiętna	B C D 8 4 2 1	plus - 3	2 4 2 1	Dwupiętkowy 5 0 4 3 2 1 0	8 4 -2 -1
0	0 0 0 0	0 0 1 1	0 0 0 0	0 1 0 0 0 0 1	0 0 0 0
1	0 0 0 1	0 1 0 0	0 0 0 1	0 1 0 0 0 1 0	0 1 1 1
2	0 0 1 0	0 1 0 1	0 0 1 0	0 1 0 0 1 0 0	0 1 1 0
3	0 0 1 1	0 1 1 0	0 0 1 1	0 1 0 1 0 0 0	0 1 0 1
4	0 1 0 0	0 1 1 1	0 1 0 0	0 1 1 0 0 0 0	0 1 0 0
5	0 1 0 1	1 0 0 0	1 0 1 1	1 0 0 0 0 0 1	1 0 1 1
6	0 1 1 0	1 0 0 1	1 1 0 0	1 0 0 0 0 1 0	1 0 1 0
7	0 1 1 1	1 0 1 0	1 1 0 1	1 0 0 0 1 0 0	1 0 0 1
8	1 0 0 0	1 0 1 1	1 1 1 0	1 0 0 1 0 0 0	1 0 0 0
9	1 0 0 1	1 1 0 0	1 1 1 1	1 0 1 0 0 0 0	1 1 1 1

Rys. 3.18. Przykłady kodów binarnych cyfr dziesiętnych

Typowym przykładem kodów spotykanych w systemach cyfrowych są kody binarne cyfr dziesiętnych. Są to kody mające co najmniej cztery bity, gdyż trzy bity dają tylko osiem wartości wektora kodującego, a cztery - szesnaście. Dziesięć z nich może być przyporządkowanych odpowiednim cyfrom w dowolny sposób. Do typowych należą kody przedstawione w tabeli pokazanej na rysunku 3.18. Kod BCD (ang. Binary Coded Decimal) jest bezpośrednim zapisem binarnym cyfr dziesiętnych traktowanych jako liczby. Należy jednak odróżniać zapis binarny liczby od jej zapisu w kodzie

binarnym. W ostatnim przypadku kodowanie obejmuje niezależnie każdą cyfrę, a więc przykładowo liczba 12 będzie zapisana w kodzie BCD następująco:

0001 0010,

natomiast jej postać binarna jest inna: 1100. Kod bezwagowy plus 3 był używany w maszynach cyfrowych starego typu. Posiada on tę właściwość, że uzupełnienie do dziewięciu odpowiedniej cyfry uzyskujemy poprzez zamianę $0 \rightarrow 1$ i $1 \rightarrow 0$. Na przykład $3 \rightarrow 0110$; $6 \rightarrow 1001$, a 6 jest uzupełnieniem dziesiętkowym 3. Pozwala to na prostą realizację odejmowania poprzez sumowanie uzupełnień. Taką samą właściwość ma kod wagowy (2 4 2 1) oraz kod z wagami ujemnymi (8 4 - 2 - 1). Kod dwupiętkowy posiada większą liczbę bitów niż cztery, ale dzięki temu każda cyfra jest zapisana za pomocą dwóch jedynek i pięciu zer, co ułatwia detekcję zakłóceń.

Przykład 3.6. Zaprojektować konwerter kodu BCD na kod (2 4 2 1).

Wartości podane w tablicy pokazanej na rysunku 3.18 można bezpośrednio wpisać do tablic Karnaugh (rys. 3.19). Poszczególne wejścia wagowe oznaczono odpowiednio: $8 \rightarrow a$, $4 \rightarrow b$, $2 \rightarrow c$, $1 \rightarrow d$. Nazwy wyjść kodu (2 4 2 1) określono dużymi literami odpowiednio (A B C D). Sześć mintermów nieokreślonych wykorzystano w procesie minimalizacji. Dla poszczególnych wyjść, łącząc w grupy jedynki otrzymamy:

$$A = a + b d + b c,$$

$$B = a + b \bar{d} + b c,$$

$$C = a + \bar{b} c + b \bar{c} d,$$

$$D = d.$$

Łącząc w grupy zera uzyskujemy następujące formuły minimalne dla poszczególnych wyjść:

$$A = (a + b)(a + c + d),$$

$$B = (a + b)(a + c + \bar{d}),$$

$$C = (\bar{b} + \bar{c})(a + b + c)(a + c + d),$$

$$D = d.$$

ab \ cd		00	01	11	10
		00	01	11	10
00		0	0	0	0
01		0	1	1	1
11		X	X	X	X
10		1	1	X	X

A

ab \ cd		00	01	11	10
		00	01	11	10
00		0	0	0	0
01		1	0	1	1
11		X	X	X	X
10		1	1	X	X

B

ab \ cd		00	01	11	10
		00	01	11	10
00		0	0	1	1
01		0	1	0	0
11		X	X	X	X
10		1	1	X	X

C

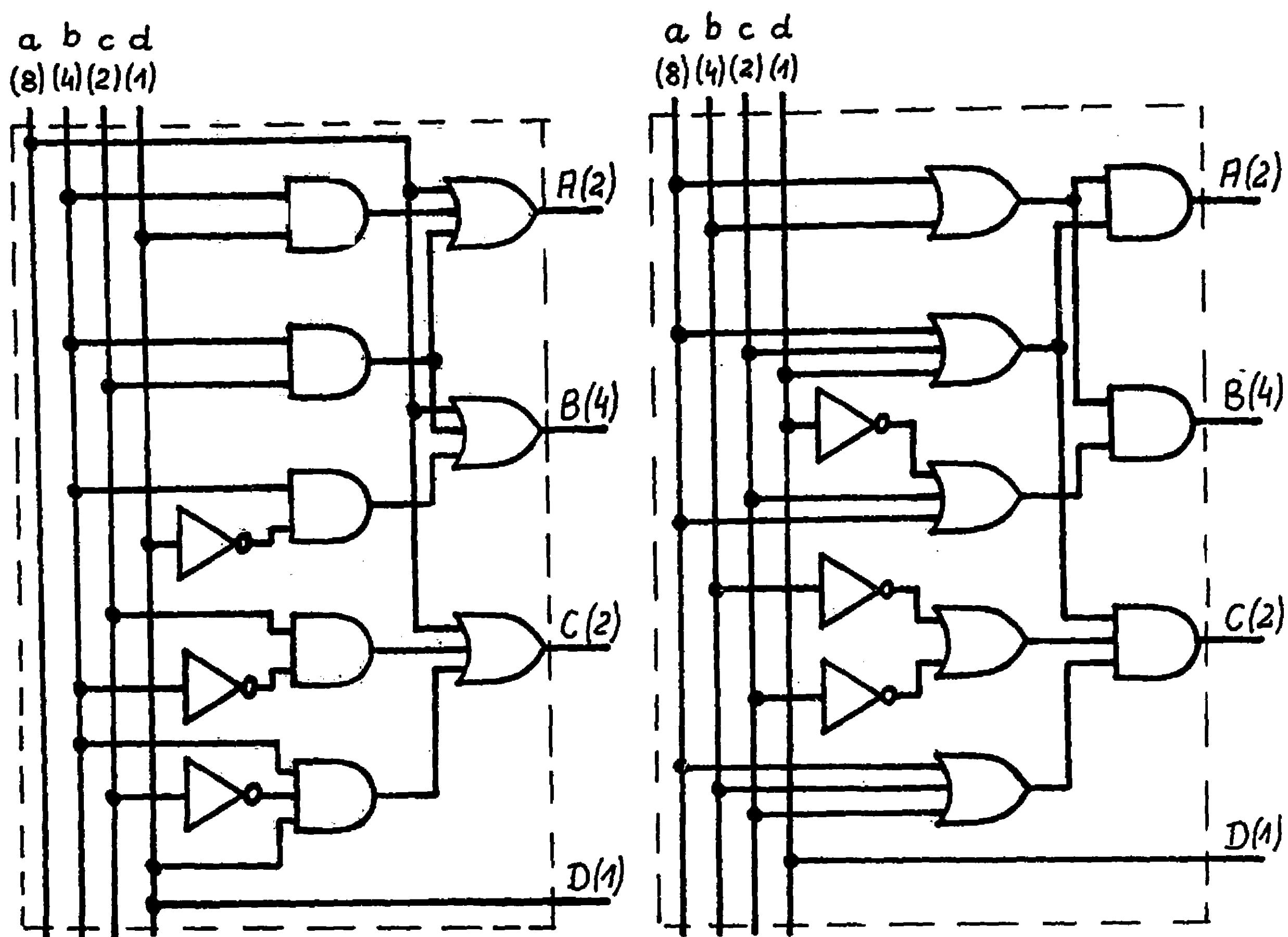
ab \ cd		00	01	11	10
		00	01	11	10
00		0	1	1	0
01		0	1	1	0
11		X	X	X	X
10		0	1	X	X

D

Rys. 3.19. Tablice Karnaugh konwertera kodu BCD na kod (2 4 2 1) (przykład 3.6)

Schematy logiczne obu wariantów konwertera przedstawiono na rysunku 3.20. Wskaźnik kosztu dla obu realizacji jest taki sam i wynosi $\langle 23, 11 \rangle$. Na uwagę zasługuje fakt, że niektóre bramki drugiego poziomu sterują więcej niż jedno wejście bramek pierwszego poziomu, co dodatkowo zmniejsza koszt układu. Fakt ten wynika z ogólnych metod minimalizacji zespołu funkcji boolowskich omówionych dalej w punkcie 3.7.

Specjalną rolę wśród konwerterów kodu odgrywają układy przetwarzające dowolny kod wielobitowy na kod "1 z n", charakteryzujący się występowaniem jedynki tylko na jednym wyjściu przypisanym ściśle określonej wartości wektora wejściowego. Konwertery te noszą nazwę dekodatorów, gdyż ich funkcjonowanie może być utożsamiane z dekodowaniem mintermów kodu wielobitowego.



Rys. 3.20. Dwa warianty realizacji konwertera kodu BCD na kod (2 4 2 1) (przykład 3.6)

Przykład 3.7. Zaprojektować układ dekodera kodu B C D na kod "1 z 10".

Tablicę wartości logicznych układu dekodera przedstawiono na rysunku 3.21. Jest to tablica diagonalna, w której jedynki występują tylko na przekątnej, a w pozostałych kratkach są zera. W tej sytuacji w tablicę Karnaugh zamiast jedynek można wpisać nazwy wyjść i minimalizację przeprowadzać wykorzystując mintermy nieokreślone. Otrzymamy następujący zbiór funkcji wyjściowych:

$$\begin{aligned}
 D_0 &= \bar{a} \bar{b} \bar{c} \bar{d}, & D_5 &= b \bar{c} d, \\
 D_1 &= \bar{a} \bar{b} \bar{c} d, & D_6 &= b c \bar{d}, \\
 D_2 &= \bar{b} c \bar{d}, & D_7 &= b c d, \\
 D_3 &= \bar{b} c d, & D_8 &= a \bar{c} \bar{d}, \\
 D_4 &= b \bar{c} \bar{d}, & D_9 &= a d.
 \end{aligned}$$

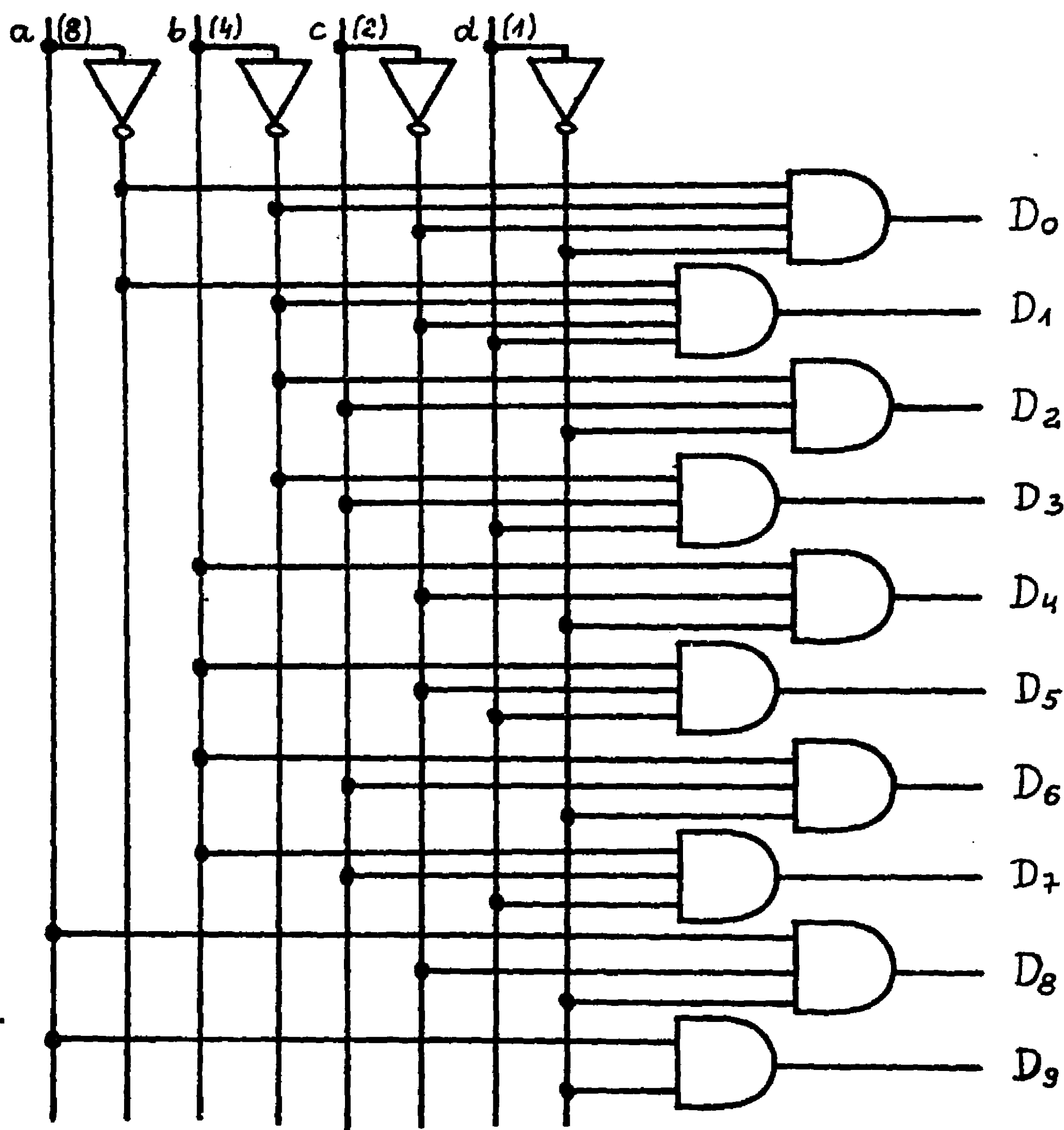
a	b	c	d	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9
8	4	2	1										
0	0	0	0	1									
0	0	0	1		1								
0	0	1	0			1							
0	0	1	1				1						
0	1	0	0					1					
0	1	0	1						1				
0	1	1	0							1			
0	1	1	1								1		
1	0	0	0									1	
1	0	0	1										1

cd \ ab		cd			
		00	01	11	10
00	00	D_0	D_1	D_3	D_2
01	01	D_4	D_5	D_7	D_6
11	11	X	X	X	X
10	10	D_8	D_9	X	X

Rys. 3.21. Tablica wartości logicznych dekodera kodu BCD, wraz z sumaryczną tablicą Karnaugh (przykład 3.7)

Schemat logiczny układu przedstawiono na rysunku 3.22. Jeśli wyjścia dekodera będą sterować odpowiednie wyświetlacze cyfr dziesiętnych, to uzyskujemy przetworzenie wyniku binarnego na obrazową postać liczby dziesiętnej.

Proces odwrotny do dekodowania nazywamy kodowaniem. Układ realizujący powyższe zadanie jest konwerterem kodu "1 z n" na odpowiedni kod binarny. Nosi on nazwę kodera.



Rys. 3.22. Schemat logiczny układu dekodera (przykład 3.7)

Przykład 3.8. Zaprojektować koder "1 z 10" na kod B C D.

Tabela wartości logicznych układu koder jest taka sama jak i dla dekodera (rys. 3.21) tylko, że zmiennymi wejściowymi są tym razem zmienne $D_0, \dots, D_9$, natomiast wyjściowymi: a, b, c, d . Mamy więc do czynienia z układem, który ma 10 wejść i cztery wyjścia. 10 zmiennych daje 1024 różnych wartości wektora wejściowego, z których jednak 1014 jest niewykorzystywanych. Przeprowadzenie minimalizacji metodami tabulacyjnymi (p. 2.12) wymaga przetwarzania długich kolumn generujących implikanty (implicenty), przy czym dopiero na końcu tablica implikantów implimentów prostych pozwoli zdyskontować fakt nieokreśloności

wielkiej liczby mintermów (makstermów). Problem ten będzie omówiony szerzej w punkcie 3.8, tutaj wynik końcowy uzyskamy bezpośrednio z tabeli pokazanej na rysunku 3.21, poprzez sumowanie tych zmiennych, dla których określone wyjście przyjmuje wartość 1. W rezultacie otrzymamy następujący zbiór funkcji:

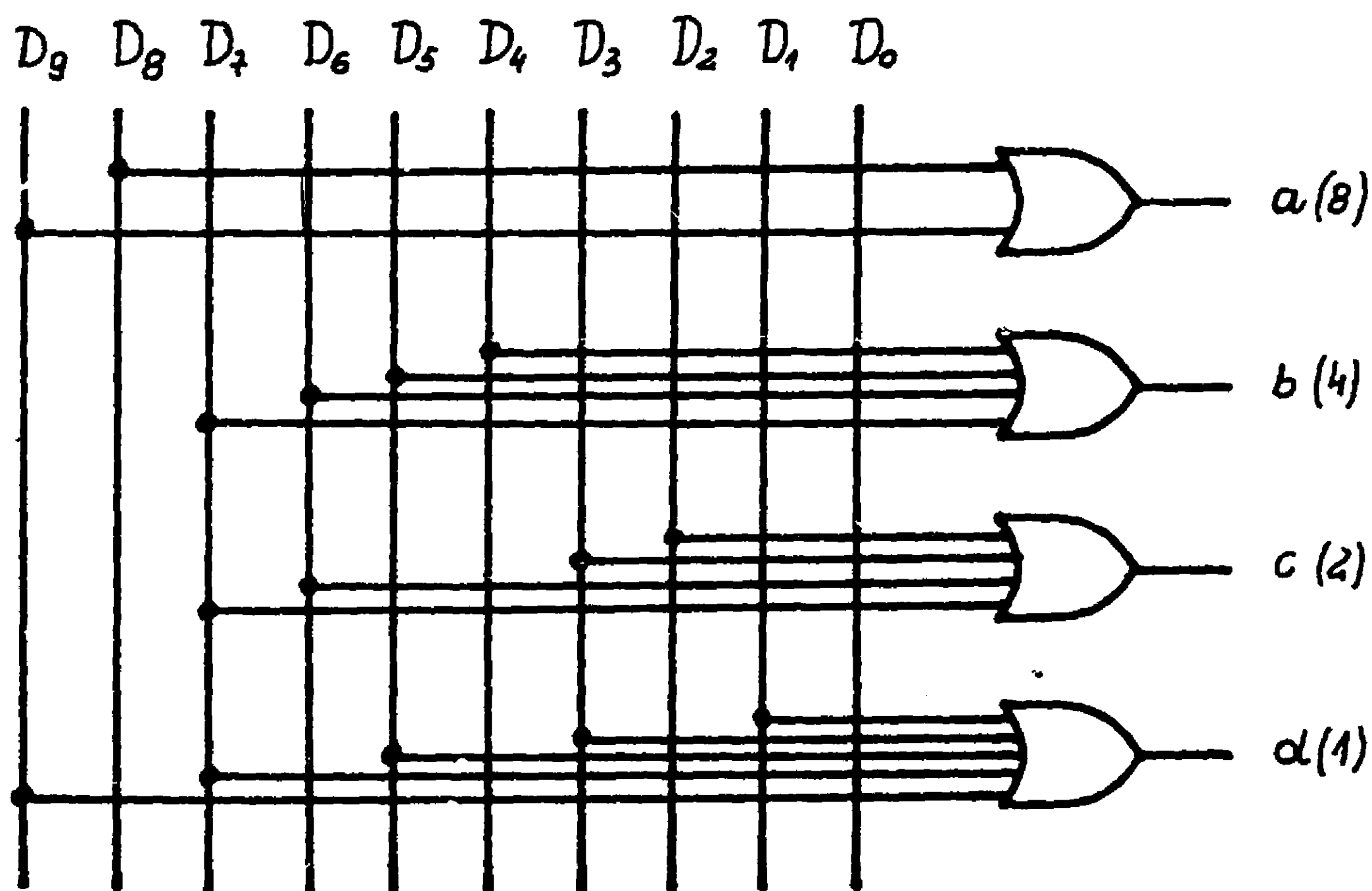
$$a = D_8 + D_9,$$

$$b = D_4 + D_5 + D_6 + D_7,$$

$$c = D_2 + D_3 + D_6 + D_7,$$

$$d = D_1 + D_3 + D_5 + D_7 + D_9.$$

Zwraca uwagę fakt niewykorzystania w procesie kodowania wejścia D_0 . Na rysunku 3.23 przedstawiono schemat logiczny omawianego kodera.



Rys. 3.23. Schemat logiczny kodera kodu "1 z 10" na kod BCD (przykład 3.8)

Ważną rolę w procesie sterowania automatycznego odgrywają tzw. kody refleksyjne. Są to kody bezwagowe, a ich cechą charakterystyczną jest to, że kolejne sekwencje wektorów wejściowych różnią się tylko na jednej pozycji. Unika się w ten sposób nieokreśloności w stanach przejś-

ciowych przełączania poszczególnych sekwencji, gdyż z wyjątkiem jednej pozycji, wszystkie pozostałe bity są niezmienione. Przykładem kodu refleksyjnego jest wprowadzony w punkcie 2.12 system opisu tablic Karnaugh, gwarantujący to, że mintermy w sąsiadujących ze sobą kratkach różnią się tylko na jednej pozycji. Można tworzyć dowolne kody refleksyjne, szczególnie interesujące są jednak te, dla których daje się określić algorytm syntezy kodu bez ograniczania liczby bitów. Do takich kodów należy omówiony już kod tablic Karnaugh, nazwany kodem Gray'a. Na rysunku 3.24 przedstawiono tablicę zawierającą czterobitowy kod Gray'a. W oparciu o nią można określić konwerter kodu Gray'a na kod binarny (liczbę w zapisie binarnym). Bezpośrednio z tabeli wynika, że:

Kod Gray'a							
G_3	G_2	G_1	G_0	y_3	y_2	y_1	y_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	0
0	0	1	0	0	0	1	1
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
0	1	0	1	0	1	1	0
0	1	0	0	0	1	1	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	1	1	0	1	0
1	1	1	0	1	0	1	1
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	0	0	1	1	1	1	0
1	0	0	0	1	1	1	1

Rys. 3.24. Tablica wartości logicznych czterobitowego kodu Gray'a i czterobitowych liczb binarnych

$$y_3 = G_3,$$

oraz

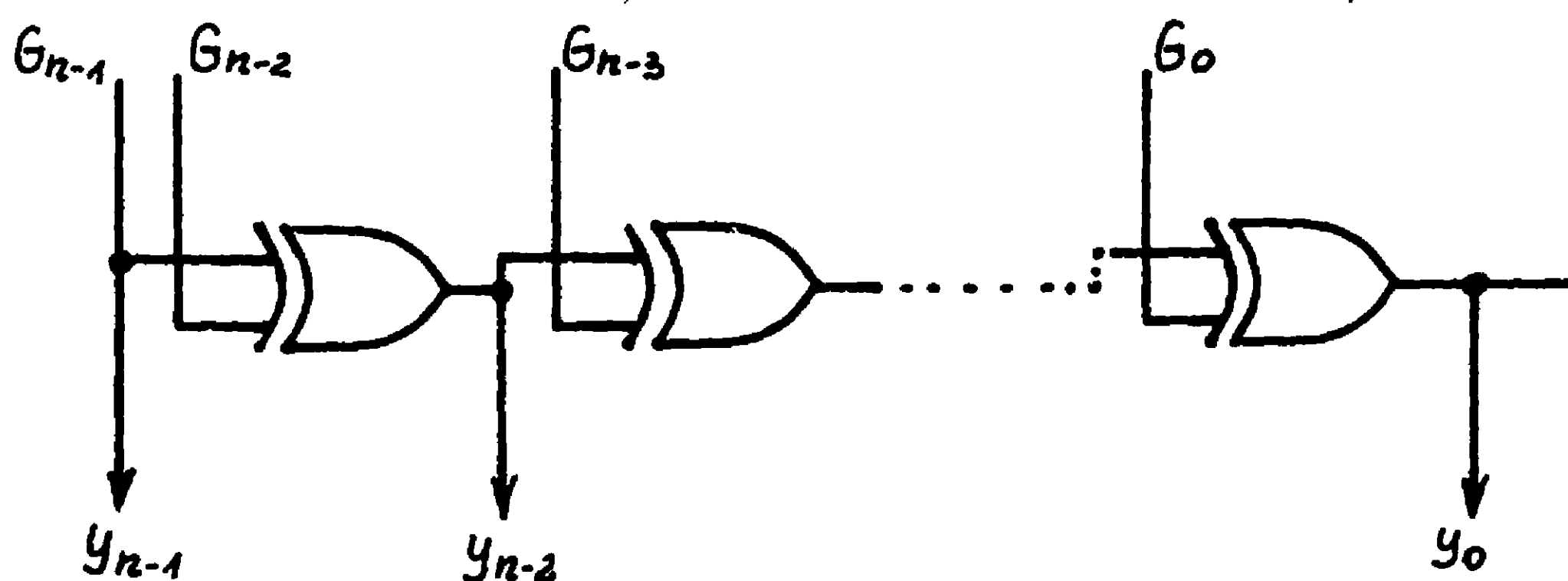
$$y_2 = G_2 \bar{G}_3 + \bar{G}_2 G_3 = G_2 \oplus G_3,$$

i dalej

$$y_1 = G_1 \oplus G_2 \oplus G_3,$$

$$y_0 = G_0 \oplus G_1 \oplus G_2 \oplus G_3.$$

Otrzymaliśmy strukturę iteracyjną, która może być dalej rozwijana w sposób przedstawiony na rysunku 3.25.

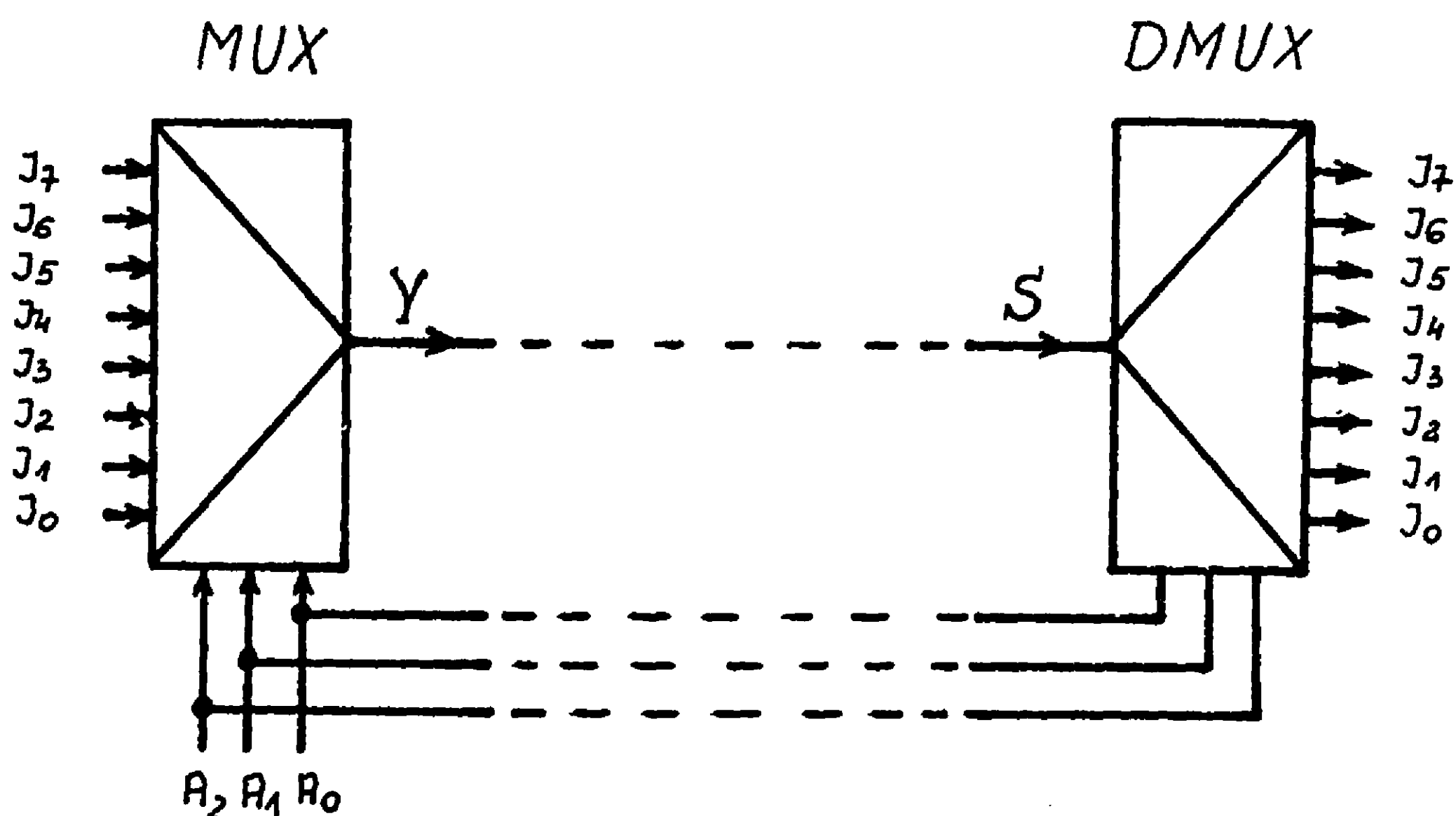


Rys. 3.25. Schemat iteracyjny konwertera n-bitowego kodu Gray'a na kod binarny

3.6. Multiplexery i demultiplexery

Nazwy multiplexery i demultiplexery pochodzą od nazw znanego w telekomunikacji procesu multipleksowania i demultipleksowania. Polega on po stronie nadawczej na przesyłaniu informacji z większej liczby kanałów mniejszą liczbą linii przesyłowych (multipleksowanie), natomiast po stronie odbiorczej - na odtworzeniu poprzedniej liczby kanałów informacyjnych (demultipleksowanie). Na rysunku 3.26 przedstawiono schemat procesu multipleksowania i demultipleksowania dla ośmiu kanałów informacyjnych. Multiplexer posiada osiem wejść informacyjnych oznaczonych $I_0 \div I_7$, oraz trzy wejścia adresowe A_2, A_1, A_0 . Zależnie od stanu wejść adresowych, na wyjście Y jest podawany sygnał z określonego wejścia informacyjnego. Ogólnie można napisać

$$Y = I_0 \bar{A}_2 \bar{A}_1 \bar{A}_0 + I_1 \bar{A}_2 \bar{A}_1 A_0 + I_2 \bar{A}_2 A_1 \bar{A}_0 + \dots + I_7 A_2 A_1 A_0. \quad (1)$$



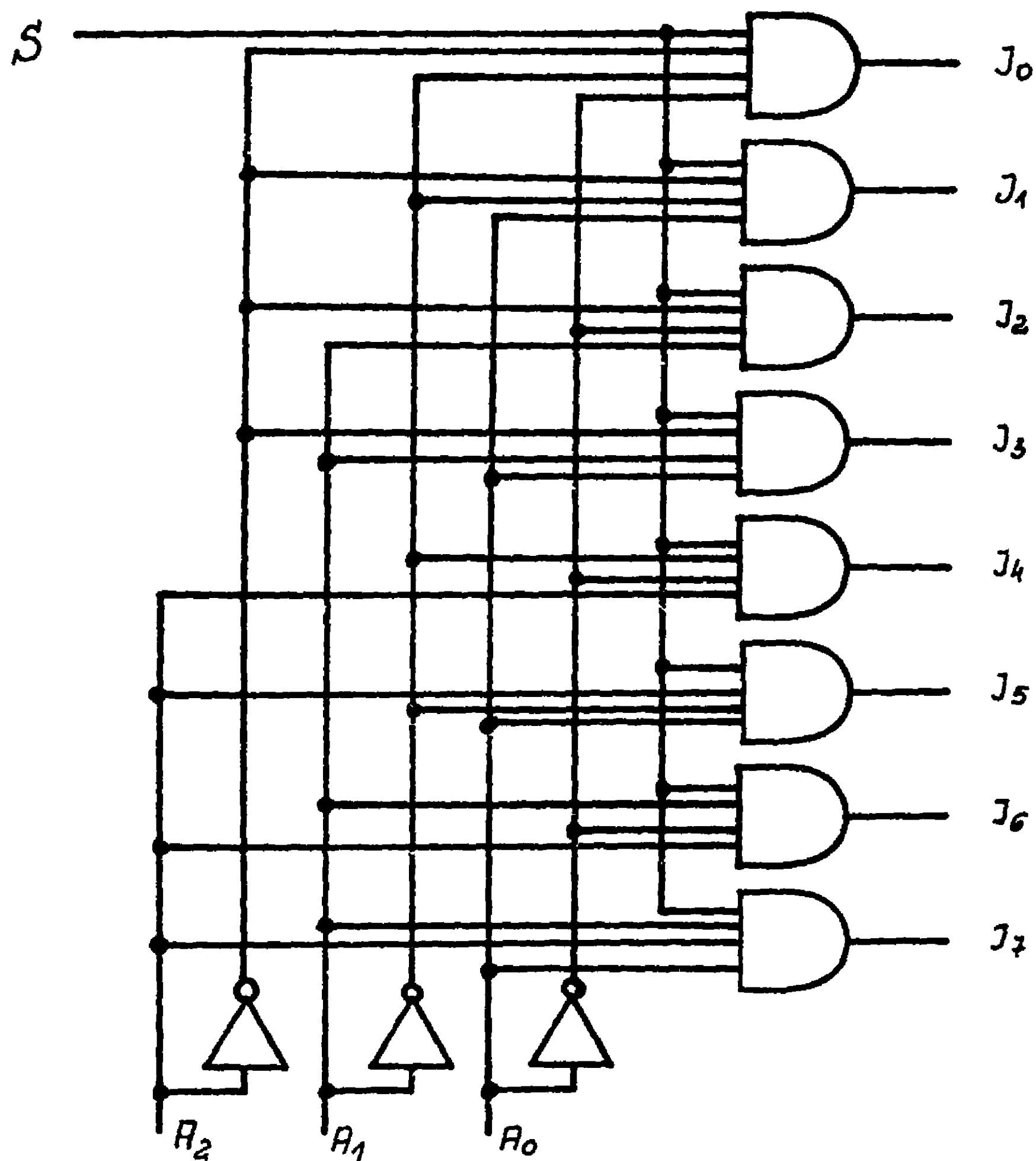
Rys. 3.26. Ilustracja procesu multipleksowania i demultipleksowania, na przykładzie 8 wejść informacyjnych

Zysk z zastosowania multipleksera wynika z faktu, że n wejść adresowych pozwala obsłużyć 2^n wejść informacyjnych. Liczba linii przesyłowych zmniejsza się z 2^n do wartości

$$N = n + 1.$$

Dla $n = 3$ (rys. 3.26) daje to redukcję z 8 do 4, ale już np. dla $n = 5$ z 32 do $5 + 1 = 6$. Odbywa się to kosztem podziału w czasie, ale w większości zastosowań, z uwagi na dużą szybkość pracy multipleksera, nie stanowi to istotnego ograniczenia.

Do demultipleksera są przesyłane te same sygnały adresowe (rys. 3.27), w związku z czym na odpowiednim wyjściu informacyjnym pojawiają się sygnały z odpowiadającego mu wejścia informacyjnego multipleksera. Praca demultipleksera jest podobna do omówionej już pracy dekodera (p. 3.5). Dla przypadku przedstawionego na rysunku 3.26 demultipleksers można potraktować jako dekodery ze strobowaniem, tzn. z wejściem wymuszającym stan 0 na wszystkich wyjściach, jeśli $S = 0$. Dla danego adresu odpowiednie wyjście, zgodnie z ideą pracy dekodera, ma stan 1 i trwa to tak długo, dopóki stan 1 występuje na wejściu S . Jeśli na tym wejściu pojawi się 0, to i na wyjściu informacyjnym pojawi się 0. W ten sposób sygnał na danym wyjściu demultipleksera powtarza sygnał na odpowiadającym mu wejściu multipleksera.



Rys. 3.27. Schemat demultipleksera 8-bitowego (dekodera ze strobowaniem S)

Zarówno multiplexery jak i demultiplexery są wytwarzane jako półprzewodnikowe układy scalone średniej skali integracji, z tym że demultiplexery występują jako dekodery ze strobowaniem (rys. 3.22 i rys. 3.27).

Fakt wytwarzania multiplexerów jako układów średniej skali integracji umożliwia wykorzystanie ich jako uniwersalnych generatorów funkcji boolowskich $n + 1$ zmiennych, gdzie n - liczba wejść adresowych. Możliwość ta wynika bezpośrednio z zależności (1) podanej dla multiplexera pokazanego na rysunku 3.26. W ogólnym przypadku funkcja wyjściowa (1) przyjmie postać

$$y = \sum_{i=0}^{2^n-1} I_i m_{Ai}, \quad (2)$$

gdzie m_{Ai} - i -ty minterm określony tylko dla wejść adresowych.

Na wejścia adresowe jest podawane w sposób uporządkowany n -zmiennych wejściowych, natomiast $n + 1$ -sza zmienna jest realizowana na wejściach informacyjnych. Oznaczając tę zmienną przez x , do wejść informacyjnych należy dołączyć elementy z następującego zbioru sygnałów

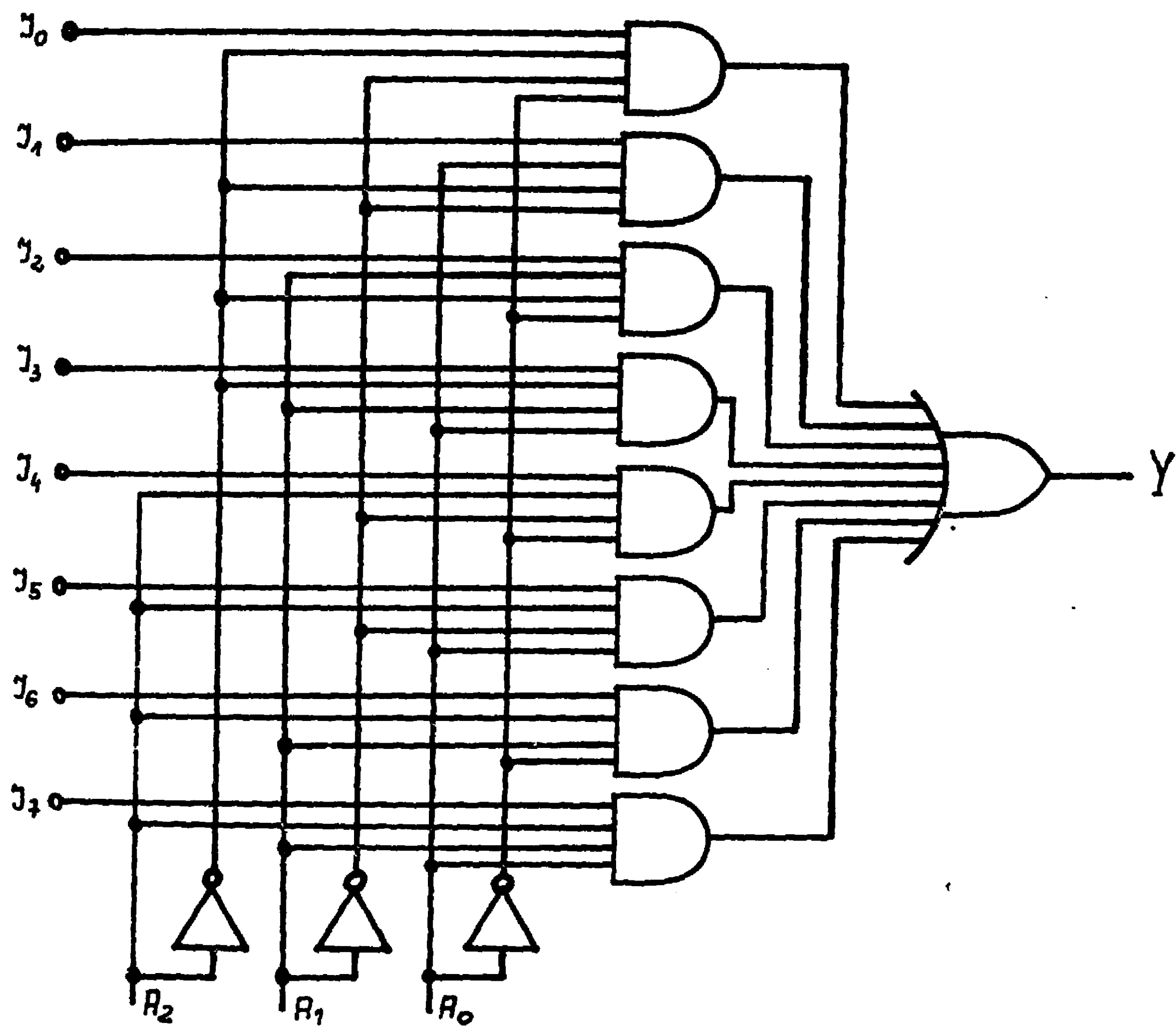
$$\{x, \bar{x}, 0, 1\}. \quad (3)$$

Jest to związane z faktem, że zbiór wszystkich mintermów funkcji k -zmiennych można zawsze podzielić względem dowolnej zmiennej x_i na dwa podzbiory o liczności 2^{k-1} , przy czym w jednym z nich zmienna x_i wystąpi w postaci afirmacyjnej, w drugim – w postaci zanegowanej. Dwa mintermy różniące się tylko postacią zmiennej x_i są mintermami sąsiednimi i są reprezentowane przez jeden implikant, będący iloczynem skróconym o tę zmienną. Iloczyn ten realizują zmienne adresowe, a na odpowiednim wejściu informacyjnym zgodnie z (2) należy przyłączyć sygnał 1. W ten sposób skrócą się wszystkie mintermy sąsiednie względem zmiennej x_i występujące w postaci kanonicznej danej funkcji. Pozostaną mintermy niesąsiednie, do realizacji których na odpowiednie wejście informacyjne należy włączyć sygnał x_i lub $\bar{x}_i$. Mintermy nie występujące w funkcji są eliminowane przyłączeniem 0 do właściwych wejść multiplexera.

Przykład 3.9. Zrealizować za pomocą multiplexera funkcję

$$f(w,x,y,z) = \sum (2,3,6,9,11,13,14).$$

Do realizacji funkcji zastosujemy układ multiplexera o $n = 3$ wejściach adresowych i $2^3 = 8$ wejściach informacyjnych (rys. 3.28). Do wejść adresowych dołączamy w sposób uporządkowany zmienne: x y z . Aby ustalić sposób rozdziału sygnałów (3) na poszczególne wejścia informacyjne wpiszemy funkcję do tabeli pokazanej na rysunku 3.29. Tabelę tę można nazwać tabelą wejść multiplexera. Wpisywanie funkcji odbywa się w sposób analogiczny jak do tablicy Karnaugh. Dolny wiersz krótkiej tabeli zawiera informację o sygnałach, jakie muszą być dołączone do wejść informacyjnych multiplexera. Dwie jedynki w pionie danego adresu są odpowiednikami mintermów sąsiednich, a więc należy na wejście informacyjne o odpowiednim adresie podać sygnał 1. Brak mintermów w funkcji odnotowują dwa pionowo usytuowane zera – eliminujemy je przez podane na właściwe wejście sygnału 0. Jeśli dla tego samego adresu występuje 0 i 1, w miejsce wyjścia informacyjnego należy włączyć dla jedynki funkcję zmien-

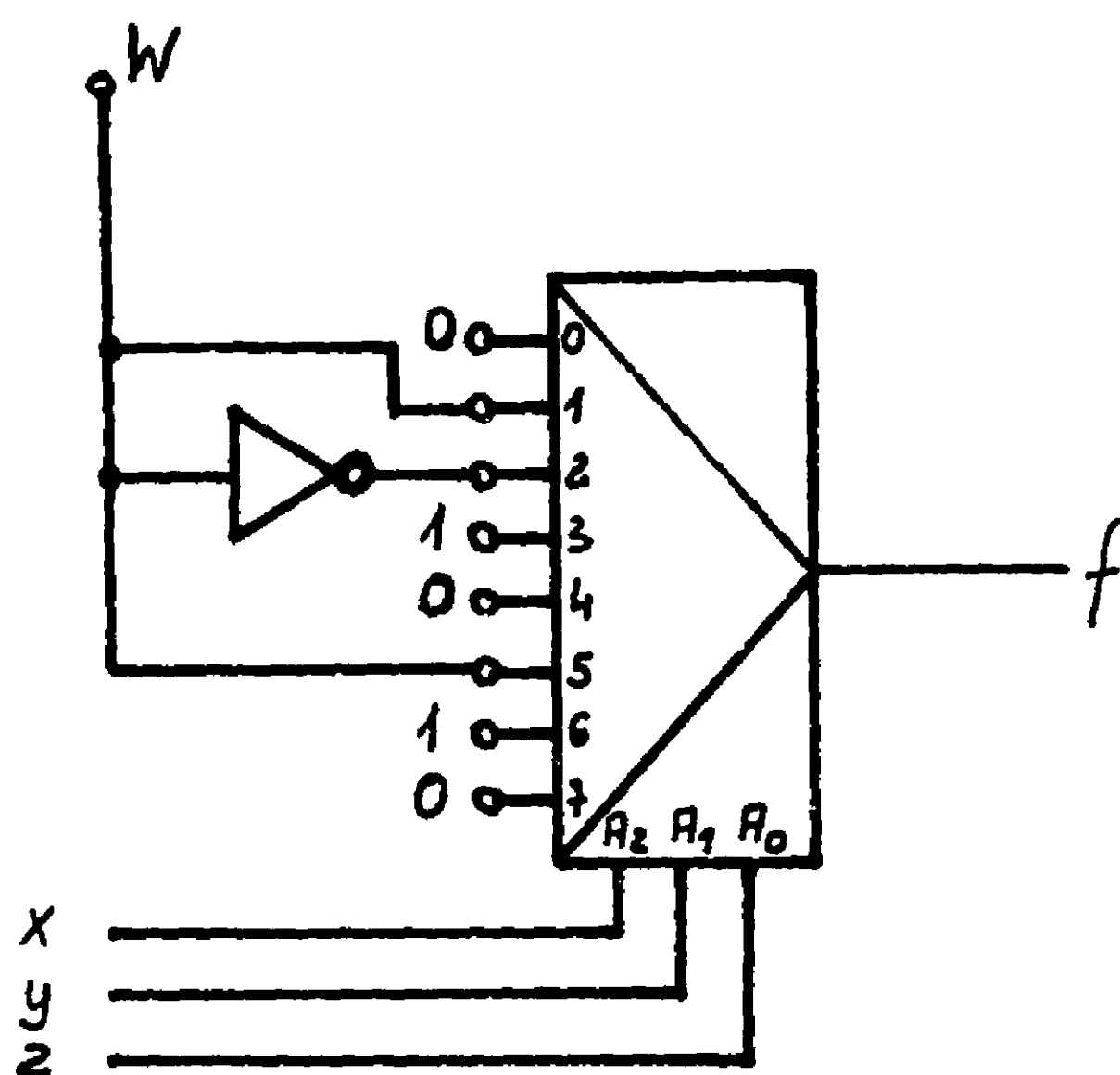


Rys. 3.28. Schemat bramkowy multipleksa o ośmiu wejściach informacyjnych (przykład 3.9)

xyz		$\bar{x}\bar{y}\bar{z}$	$\bar{x}\bar{y}z$	$\bar{x}y\bar{z}$	$\bar{x}yz$	$x\bar{y}\bar{z}$	$x\bar{y}z$	$xy\bar{z}$	xyz
w		0	1	2	3	4	5	6	7
0		0	0	1	1	0	0	1	0
1		0	1	0	1	0	1	1	0
		0	w	$\bar{w}$	1	0	w	1	0

Rys. 3.29. Funkcja z przykładu 3.9 wpisana do tablicy wejść multipleksa

ną w postaci afirmacyjnej dla wiersza $w = 1$, w postaci zanegowanej dla wiersza $w = 0$. Wynika to bezpośrednio z ustaleń wartości w podanych z lewej strony opisu tabeli. Realizację funkcji podano na rysunku 3.30. Oczywiście realizacja funkcji o mniej niż $n + 1$ zmiennych jest jeszcze prostsza. Na przykład dla n -zmiennych do wejść informacyjnych wystarczy dołączać zera i jedynki, tak jak to wynika bezpośrednio z postaci koniunkcyjnej funkcji.



Rys. 3.30. Schemat połączeń multiplexera realizującego funkcję z przykładu 3.9

3.7. Optymalna realizacja układów wielowyjściowych

3.7.1. Metoda tabulacyjna minimalizacji zespołu funkcji

Układy logiczne są opisywane przez zespoły funkcji boolowskich, jednak dotychczas poszczególne wyjścia były traktowane niezależnie i proces poszukiwania formuł minimalnych przeprowadzano dla każdego wyjścia osobno. Pojawiające się niekiedy identyczne człony wykorzystywano jako wspólne, dopiero na etapie tworzenia schematu logicznego (rys. 2.30b, rys. 3.20). Intuicyjnie można wyczuć, że istnieje pewien kompromis między redukcją formuł poszczególnych wyjść a zachowaniem elementów dla nich wspólnych. Rozwiązanie kompromisowe można znaleźć, jeśli minimalizację zespołu funkcji przeprowadzić metodą tabulacyjną

(p. 2.12), z zaznaczeniem przynależności występujących mintermów (makstermów) do poszczególnych funkcji. Kolumnę implikantów zredukowanych o jeden literał można tworzyć tylko wtedy, gdy mintermy są wspólne, przy czym wygenerowany implikant pokrywa tylko mintermy tych funkcji, które zostały sklejone.

Przykład 3.10. Zrealizować z bramek NAND dwupoziomowy układ logiczny opisany funkcjami:

$$f_a(w,x,y,z) = \sum (5,6,7,13,14,15)$$

$$f_b(w,x,y,z) = \sum (5,6,13,14).$$

Funkcja f_b jest sumą mintermów będących podzbiorem mintermów funkcji f_a . Pozwala to przypuszczać, że minimalizacja zespołu funkcji będzie bardziej korzystna niż minimalizacja każdej funkcji osobno. Pierwszą kolumnę mintermów tworzymy wg podziałów na grupy indeksowe z zaznaczeniem dodatkowo literami przynależności do poszczególnych funkcji:

$$\begin{array}{l} 5 \text{ a b} \\ 6 \text{ a b} \\ \hline 7 \text{ a} \\ 13 \text{ a b} \\ 14 \text{ a b} \\ \hline 15 \text{ a} \end{array}$$

Drugą i następne kolumny tworzymy wg reguł podanych w punkcie 2.12.2, z tym że wygenerowany implikant pokrywa tylko te mintermy, dla których jest wspólny. Przykładowo 5 a b i 7 a dają rezultat 5, 7 (2)a. Otrzymany implikant pokrywa tylko minterm 7 a (tylko a jest wspólne dla obu mintermów) i tylko ten rezultat wolno zaznaczyć w kolumnie pierwszej. Minterm 5 a b zostanie sklejony dopiero w następnym etapie z mintermem 13 a b. Ostatecznie otrzymamy:

$$\begin{array}{lll} \begin{array}{l} \vee 5 \text{ a b} \\ \vee \underline{6 \text{ a b}} \\ \vee 7 \text{ a} \\ \vee 13 \text{ a b} \\ \vee \underline{14 \text{ a b}} \\ \vee 15 \text{ a} \end{array} & \begin{array}{l} \vee 5,7 (2) \text{ a} \\ 5, 13 (8) \text{ a b A} \\ \vee 6, 7 (1) \text{ a} \\ \underline{\vee 6, 14 (8) \text{ a b B}} \\ \vee 7,15 (8) \text{ a} \\ \vee 13,15 (2) \text{ a} \\ \vee 14,15 (1) \text{ a} \end{array} & \left. \begin{array}{l} 5,7,13,15 (2, 8) \text{ a} \\ 5,13,7,15 (8, 2) \text{ a} \end{array} \right\} \text{C} \\ & & \left. \begin{array}{l} 6,14,7,15 (8, 1) \text{ a} \\ 6,7,14,15 (1, 8) \text{ a} \end{array} \right\} \text{D} \end{array}$$

		f_a						f_b			
		5	6	7	13	14	15	5	6	13	14
A	5, 13 (8) $a b$	X			X			X		X	
B	6, 14 (8) $a b$		X			X			X		X
C	5, 7, 13, 15 (2, 8) a	X		X	X		X				
D	6, 7, 14, 15 (1, 8) a		X	X		X	X				

Rys. 3.31. Tablica implikantów prostych zespołu funkcji z przykładu 3.10

Tablicę implikantów prostych tworzymy wspólnie dla obu funkcji (rys. 3.31). Do pokrycia wszystkich mintermów obu funkcji wystarczą tylko trzy wspólne implikanty: A, B, C (lub A, B, D). Aby określić schemat logiczny układu dokonujemy deszyfracji zapisu wybranych implikantów:

$$\begin{array}{llll}
 & & w & x & y & z \\
 5, 13, (8) & a & b & \longrightarrow & - & 1 & 0 & 1 & \longrightarrow & x & \bar{y} & z \\
 6, 14 & (8) & a & b & \longrightarrow & - & 1 & 1 & 0 & \longrightarrow & x & y & \bar{z} \\
 5, 13, (8) & a & b & \longrightarrow & - & 1 & 0 & 1 & \longrightarrow & x & \bar{y} & z \\
 6, 14 & (8) & a & b & \longrightarrow & - & 1 & 1 & 0 & \longrightarrow & x & y & \bar{z} \\
 5, 7, 13, 15 & (2, 8) & a & \longrightarrow & - & 1 & - & 1 & \longrightarrow & x & z \\
 [6, 7, 14, 15 & (1, 8) & a & \longrightarrow & - & 1 & 1 & - & \longrightarrow & x & y]
 \end{array}$$

Dodatkowym rezultatem postępowania minimalizacji jest stwierdzenie zbędności wprowadzenia zmiennej w . Obie funkcje w swej postaci wyjściowej są tylko pozornie zależne od w .

Dokonując faktoryzacji zgodnie z regułami podanymi w punkcie 2.13, można napisać:

$$f_a = x \bar{y} z + x y \bar{z} + x y = \overline{x \bar{y} z} \overline{x y \bar{z}} \overline{x y}$$

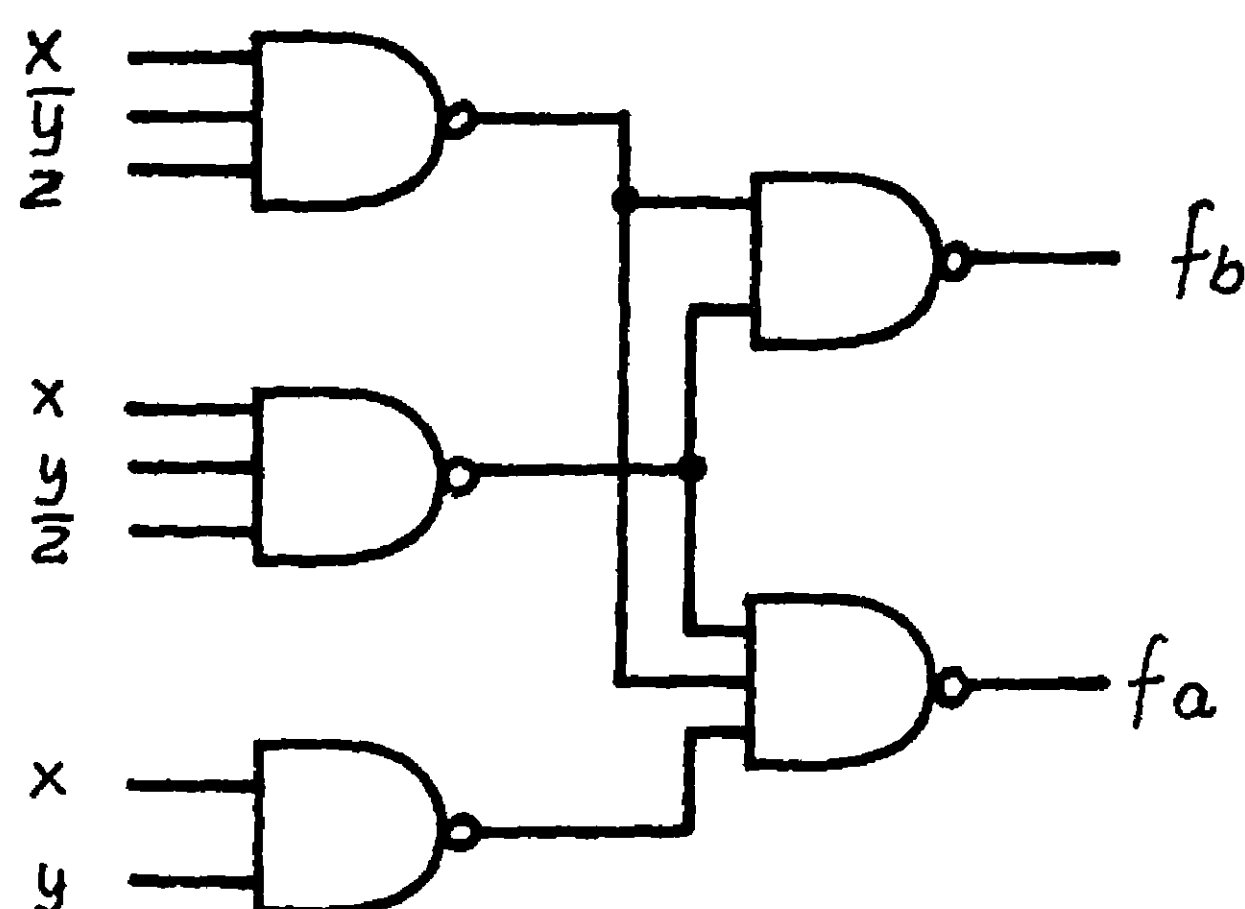
$$f_b = x \bar{y} z + x y \bar{z} = \overline{x \bar{y} z} \overline{x y \bar{z}}$$

Schemat zastępczy układu przedstawiono na rysunku 3.32. Zakładając, że zmienne są dane zarówno w postaci afirmacyjnej, jak i zanegowanej otrzymujemy realizację dwupoziomową i wskaźnik kosztu wynosi $\langle 13,5 \rangle$. Gdyby przeprowadzać minimalizację każdej funkcji osobno otrzymalibyśmy następujący rezultat:

$$f_a = xz + xy = \overline{\overline{xz} \overline{xy}}$$

$$f_b = x\bar{y}z + xy\bar{z} = \overline{\overline{x\bar{y}z} \overline{xy\bar{z}}}$$

W ostatnim przypadku wskaźnik kosztu jest większy i wynosi $\langle 14,6 \rangle$.



Rys. 3.32. Schemat logiczny układu z przykładu 3.10 zrealizowany z bramek NAND

Jeszcze prostszą postać układu można otrzymać rezygnując z realizacji dwupoziomej (normalnej). Wystarczy zauważyć, że funkcja f_a jest sumą funkcji f_b oraz implikantu $x y$.

$$f_a = f_b + x y.$$

Wskaźnik kosztu obniży się wówczas do wartości $\langle 12,5 \rangle$, ale tylko dla realizacji na bramkach sumy i iloczynu. Dla bramek NAND z uwagi na konieczność zanegowania f_b , wspomniana faktoryzacja nie ma sensu.

Przykład 3.11. Dla funkcji podanych w przykładzie 3.10, określić minimalny dwupoziomowy układ logiczny zrealizowany z bramek NOR.

Najbardziej korzystną postacią normalną funkcji jest tutaj postać koniunkcyjna, z której bezpośrednio wynika formuła dostosowana do dwupoziomowej realizacji na bramkach NOR. Z rezultatów otrzymanych w przykładzie 3.10 wynika, że obie funkcje są tylko pozornie zależne od zmiennej w , do obliczeń przyjmijmy więc już ich postać jako funkcji trzech zmiennych. Otrzymamy wówczas:

$$f_a(x, y, z) = \prod (0, 1, 2, 3, 4),$$

$$f_b(x, y, z) = \prod (0, 1, 2, 3, 4, 7).$$

Zastosowanie metody tabulacyjnej daje następujący rezultat:

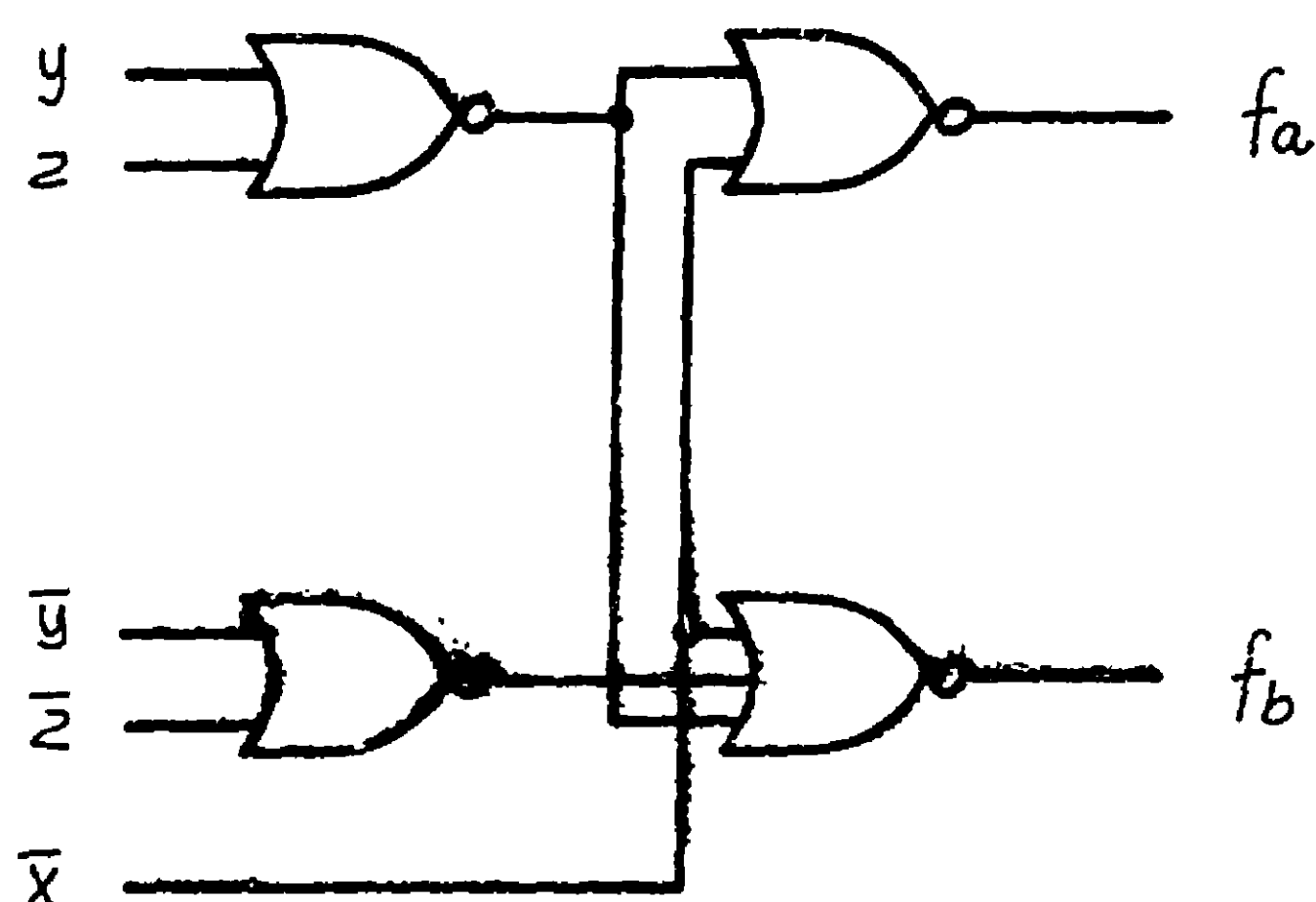
$\vee$ <u>0 a b</u>	$\vee$ 0,1 (1) a b	0,1,2,3 (1,2) a b	}	C
$\vee$ 1 a b	$\vee$ 0,2 (2) a b	0,2,1,3 (2,1) a b		
$\vee$ 2 a b	<u>0,4 (4) a b A</u>			
$\vee$ <u>4 a b</u>	$\vee$ 1,3 (2) a b			
$\vee$ <u>3 a b</u>	$\vee$ <u>2,3 (1) a b</u>			
$\vee$ 7 b	3,7 (4) b B			

Z bezpośredniego porównania postaci kanonicznych funkcji i otrzymanych implikantów wspólnych wynika, że wszystkie one są niezbędne, gdyż jako jedyne pokrywają pewne makstremy. Po deszyfracji, można napisać:

$$f_a = x (y + z),$$

$$f_b = x (y + z)(\bar{y} + \bar{z}).$$

Schemat logiczny układu przedstawiono na rysunku 3.33. Wskaźnik kosztu wynosi tu $\langle 9, 4 \rangle$, a więc jest znacznie mniejszy niż w przypadku dwupoziomowej realizacji na bramkach NAND. Minimalizacja każdej funkcji osobno, doprowadza do identycznego rezultatu. Oba rozwiązania należy więc uznać za minimalne.



Rys. 3.33. Schemat logiczny układu z przykładu 3.10 zrealizowany z bramek NOR

Przykład 3.12. Zaprojektować z bramek NOR układ kombinacyjny opisany funkcjami:

$$f_a(w, x, y, z) = \prod (7, 8, 11, 12, 15),$$

$$f_b(w, x, y, z) = \prod (7, 8, 9, 13, 15),$$

$$f_c(w, x, y, z) = \prod (8, 11, 12, 13).$$

Dla powyższego zespołu funkcji można napisać

<u>8 a b c A</u>	8, 9 (1) b D
V 9 b	<u>8, 12 (4) a c E</u>
V <u>12 a c</u>	9, 13 (4) b F
V 7 a b	<u>12, 13 (1) c G</u>
11 a c B	7, 15 (8) a b H
<u>13 b c C</u>	11, 15 (4) a I
V 15 a b	13, 15 (2) b J

Z drugiej kolumny nie da się wygenerować więcej wspólnych implikentów, w związku z czym należy wybrać te, które są niezbędne do pokrycia wszystkich makstermów zespołu funkcji (rys. 3.34). W tabeli pokazanej na rysunku 3.34, trzy implikenty B, E, H są niezbędne, gdyż jako jedyne

pokrywają określone makstermy. Pozostają jeszcze do pokrycia makstermy 8,9,13 funkcji f_b oraz maksterm 13 funkcji f_c . Wybierając implicanty C oraz D uzyskujemy pełne pokrycie wszystkich makstermów. W wyniku de-
szyfracji otrzymamy:

		w	x	y	z	
1 1 a c	→	1	0	1	1	→ $\bar{w} + x + \bar{y} + \bar{z}$
13 b c	→	1	1	0	1	→ $\bar{w} + \bar{x} + y + \bar{z}$
8,9 (1) b	→	1	0	0	-	→ $\bar{w} + x + y$
8,12 (4) a c	→	1	-	0	0	→ $\bar{w} + y + z$
7,15 (8) a b	→	-	1	1	1	→ $\bar{x} + \bar{y} + \bar{z}$

		f_a					f_b					f_c				
		7	8	11	12	15	7	8	9	13	15	8	11	12	13	
A	8 a b c		X					X				X				
B	11 a c			X									X			*
C	13 b c									X					X	V
D	8,9(1) b							X	X							V
E	8,12(4) a c		X		X							X		X		*
F	9,13(4) b								X	X						
G	12,13(1) c													X	X	
H	7,15(8) a b	X				X	X				X					*
I	11,15(4) a			X		X										
J	13,15(2) b									X	X					
		*	*	*	*	*	*	V	V	V	*	*	*	*	V	

Rys. 3.34. Tablica implicantów dla zespołu funkcji z przykładu 3.12

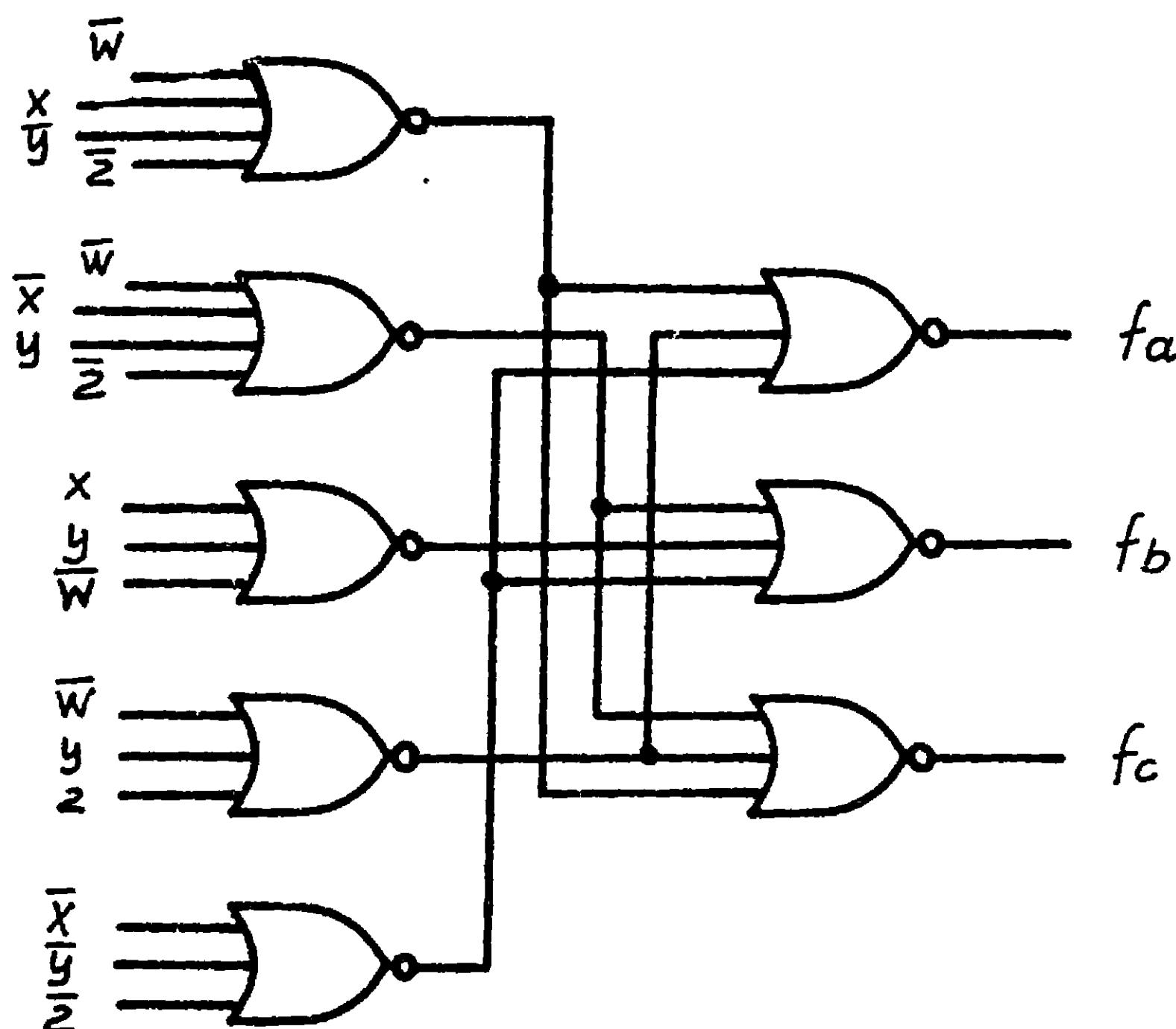
Poszczególne funkcje opiszą formuły:

$$f_a = (\bar{w} + x + \bar{y} + \bar{z})(\bar{w} + y + z)(\bar{x} + \bar{y} + \bar{z}),$$

$$f_b = (\bar{w} + \bar{x} + y + \bar{z})(\bar{w} + x + y)(\bar{x} + \bar{y} + \bar{z}),$$

$$f_c = (\bar{w} + x + \bar{y} + \bar{z})(\bar{w} + \bar{x} + y + \bar{z})(\bar{w} + y + z).$$

Realizację układu na bramkach NOR przedstawiono na rysunku 3.35. Współczynnik kosztu układu wynosi $\langle 26,8 \rangle$. Współczynnik kosztów dla przypadku indywidualnej minimalizacji wszystkich funkcji i następnie wykorzystaniu wspólnych składników, wyniósłby $\langle 31,10 \rangle$.



Rys. 3.35. Realizacja z bramek NOR układu z przykładu 3.12

3.7.2. Realizacja zespołu funkcji z zastosowaniem dekodera

Wykonywane w postaci układów średniej skali integracji dekodery scalone, mogą być wykorzystywane w połączeniu z odpowiednimi bramkami do realizacji układów wielowyjściowych. Idea takiego wykorzystania dekodera wiąże się z faktem występowania jedynki tylko na wyjściu, odpowiadającym jednej wartości wektora wejściowego. Wyjście to można potraktować jako wyjście generujące ściśle określony minterm. Dysponując postacią kanoniczną funkcji, można ją zrealizować bezpośrednio, łącząc wyjścia generujące wymagane mintermy do wejść bramki sumy.

Przykład 3.13. Zrealizować za pomocą dekodera układ opisany zespołem funkcji:

$$f_a(x, y, z) = \sum (0, 1, 6, 7),$$

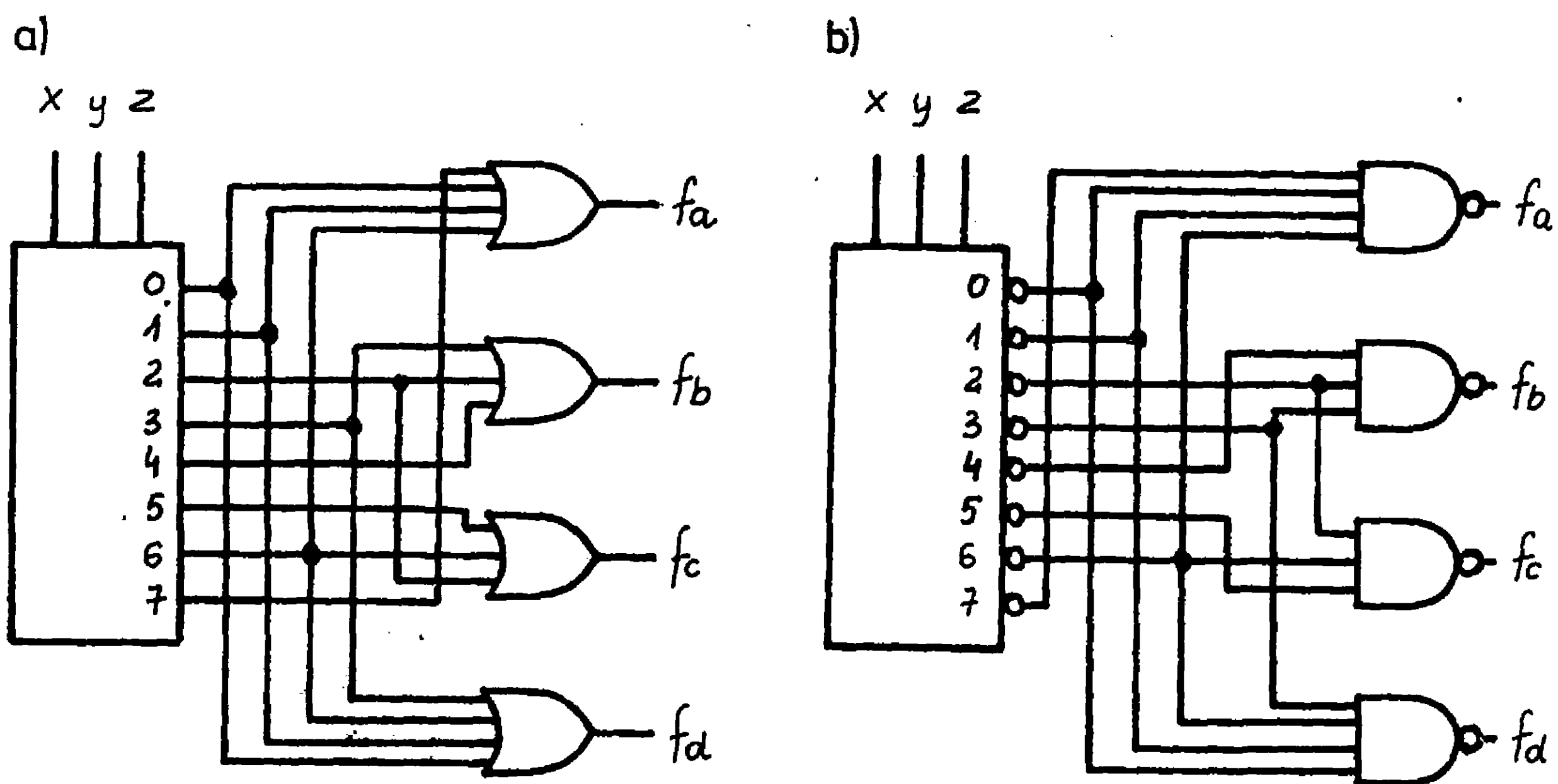
$$f_b(x, y, z) = \sum (2, 3, 4),$$

$$f_c(x, y, z) = \sum (2, 5, 6),$$

$$f_d(x, y, z) = \sum (0, 1, 3, 6).$$

Na rysunku 3.36a przedstawiono schemat układu otrzymany bezpośrednio z równań wyjściowych. Dekodery scalone mają często wyjścia zanegowane. Wtedy zamiast bramek sumy należy użyć bramek NAND, uzyskując dokładnie ten sam rezultat. Przykładowo, dla funkcji f_b , otrzymamy bowiem

$$\begin{aligned} f_b &= \sum (2, 3, 4) = \overline{\overline{x} y \bar{z}} + \overline{\overline{x} y z} + \overline{x \bar{y} \bar{z}} = \\ &= \overline{\overline{x} y \bar{z} \cdot \overline{\overline{x} y z} \cdot x \bar{y} \bar{z}}. \end{aligned}$$



Rys. 3.36. Dwa warianty realizacji układu z przykładu 3.13: a) dekodery z wyjściami afirmacyjnymi, b) dekodery z wyjściami zanegowanymi

Otrzymujemy więc w wyniku negację iloczynu zanegowanych mintermów, które są realizowane na wyjściu omawianego dekodera. Na rysunku 3.36b przedstawiono realizację zespołu funkcji z użyciem dekodera z wyjściami zanegowanymi i z bramkami NAND.

3.8. Minimalizacja funkcji nie w pełni określonych

3.8.1. Minimalizacja funkcji o małej nieokreśloności

Z funkcjami, których wartości dla pewnych mintermów były nieokreślone mieliśmy już do czynienia w omówionych uprzednio przykładach, przy czym wykorzystywanie nieokreśloności ilustrowane było za pomocą tablic Karnaugh. W przypadku metod tabulacyjnych mintermy (makstermy), dla których funkcja jest nieokreślona włączamy do kolumny z podziałami na grupy indeksowe i w ten sposób zwiększamy ilość wygenerowanych implikantów (implicentów), co poszerza możliwości redukcji minimalizowanej formuły. W tablicy implikantów bądź implicentów prostych nieokreślone mintermy lub makstermy już nie występują, natomiast wygenerowane przez nie implikanty (implicenty) proste ułatwiają dokonanie dalszej redukcji formuły. Procedurę wykorzystania nieokreśloności funkcji ilustruje przykład 3.14.

Przykład 3.14. Określić minimalną formułę dla następującej funkcji boolowskiej

$$f(w, x, y, z) = \sum((0), 2, (3), (5), 7, (8), 10, 11, 13, 15).$$

W nawiasach podano mintermy, dla których funkcja jest nieokreślona.

Powyższe zadanie rozwiążemy dla ilustracji metodą tabulacyjną, choć ze względu na małą liczbę zmiennych można by tego dokonać za pomocą tablicy Karnaugh. Otrzymujemy następujące kolumny implikantów:

V <u>0</u>	V 0,2 (2)	0,2,8,10 (2,8)	}	A
V 2	V <u>0,8 (8)</u>	<u>0,8,2,10 (8,2)</u>		
V <u>8</u>	V 2,3 (1)	2,3,10,11 (1,8)	}	B
V 3	V 2,10 (8)	<u>2,10,3,11 (8,1)</u>		
V 5	V <u>8,10 (2)</u>	3,7,11,15 (4,8)	}	C
V <u>10</u>	V 3,7 (4)	<u>3,11,7,15 (8,4)</u>		
V 7	V 3,11 (8)	5,7,13,15 (2,8)	}	D
V 11	V 5,7 (2)	<u>5,13,7,15 (8,2)</u>		
V <u>13</u>	V 5,13 (8)			
V 15	V <u>10,11 (1)</u>			
	V 7,15 (8)			
	V 11,15 (4)			
	V 13,15 (2)			

Tablicę implikantów prostych (rys. 3.37) tworzymy bez mintermów, dla których funkcja jest nieokreślona. Do pokrycia wymienionych w tabeli mintermów wystarczą dwa implikanty. Implikant D, który jako jedyny pokrywa minterm 13 oraz implikant B. Po deszyfracji otrzymamy ostatecznie

$$f(w,x,y,z) = \bar{x} y + x z.$$

		*	*	*	*	*	*
		2	7	10	11	13	15
A	0, 2, 8, 10(2, 8)	X		X			
B	2, 3, 10, 11(1, 8)	X		X	X		
C	3, 7, 11, 15,(4, 8)		X		X		X
D	5, 7, 13, 15(2, 8)		X			X	X

Rys. 3.37. Tablica implikantów prostych dla funkcji z przykładu 3.14

3.8.2. Minimalizacja funkcji silnie nieokreślonych

Z funkcjami silnie nieokreślonymi mieliśmy już do czynienia w przykładzie 3.8, gdzie omawiano układ koder 1 z 10 na kod B C D. Funkcje wyjściowe były określone jedynie dla dziesięciu spośród 1024 możliwych mintermów. Wprowadźmy metodę podaną w poprzednim podpunkcie ma charakter uniwersalny i nadaje się do poszukiwania formuł minimalnych i w tym przypadku, jednak z punktu widzenia obliczeń, jest bardzo nieefektywna z uwagi na konieczność przeszukiwania długich kolumn generowanych implikantów. W tej sytuacji najlepsze rezultaty daje metoda porównywania zbioru mintermów, dla których funkcja przyjmuje wartość 1 ze zbiorem mintermów, dla którego funkcja ta przyjmuje wartość 0, a następnie określenia formuł minimalnych wystarczających do odróżnienia obydwu zbiorów. W analogiczny sposób postępujemy, jeśli funkcja jest określona dla zbiorów makstermów. Przykładowo, jeśli funkcję $F(a,b,c,d,e)$, określają następujące zbiory jedynek F^1 oraz zer F^0 :

$$F^1 = \left\{ \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \end{bmatrix} \right\}; \quad F^0 = \left\{ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \right\},$$

przy czym mintermy (makstermy) zostały zapisane za pomocą ciągów zero-jedynkowych, to formuła

$$F = d + e$$

całkowicie wystarcza do odróżnienia zbioru jedynek od zbioru zer, gdyż wszystkie ciągi zbioru F^0 mają na tych pozycjach zera. Postępowanie powyższe można bardziej sformalizować; ilustruje to przykład 3.15.

Przykład 3.15. Znaleźć minimalną formułę boolowską dla funkcji słabookreślonej $F(a,b,c,d,e,f)$ zadanej zbiorami

$$F^1 = \left\{ \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \right\}; \quad F^0 = \left\{ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \right\}.$$

Procedurę poszukiwań wygodnie jest zacząć od tabeli umożliwiającej przeprowadzenie rejestracji różnic między poszczególnymi składnikami zer i jedynek. Tabelę taką pokazano na rysunku 3.38. W wierszach umieszczono ciągi reprezentujące jedyneki funkcji F , w kolumnach - ciągi dla zer. Na przecięciu odpowiednich wierszy i kolumn zaznaczono pozycje, na których ciągi te się różnią, przy czym kierunek numeracji przebiega od lewej do prawej strony ciągu. Analiza zawartości całej tabeli umożliwia znalezienie minimalnej formuły umożliwiającej odróżnienie ciągów ze zbioru jedynek od ciągów ze zbioru zer. Poszukiwania te są żmudne i wymagają w zasadzie zastosowania komputera. W większości zastosowań wystarczy jednak ograniczyć się do poszukiwań quasioptymalnych, dla których algorytm jest bardzo prosty, a rozwiązanie najczęściej optymalne albo bliskie optymalnego. Algorytm ten sprowadza się do poszukiwania w kolumnach zer i w wierszach jedynek minimalnych zbiorów numerów pozycji umożliwiających:

- dla kolumny, odróżnienie danego zera od wszystkich jedynek,
- dla wiersza, odróżnienie danej jedynki od wszystkich zer.

$F^1 \backslash F^0$	000001	010001	100000		
001011	3,5	2,3,5	1,3,5,6	3;5	$\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}$
010100	2,4,6	4,6	1,2,4	4	$\bar{1}\bar{1}\bar{1}\bar{1}$
001000	3,6	2,3,6	1,3	3	$\bar{1}\bar{1}\bar{1}\bar{1}$
	2,3 ; 3,4 3,6 ; 5,6	2,6;2,4;3,4 3,6 ; 5,6	1		
	-0 0- - - - - 00 - - - - 0- - 1 - - - - 01	-1 - - - 1 -1 - 0 - - - - 00 - - - - - - 01 - - 0 - - 1	1 - - - - -		

Rys. 3.38. Tablica porównawcza dla funkcji z przykładu 3.15

Zbiory te zaznaczono dla kolumn w dolnej części tabeli, dla wierszy - z jej prawej strony. Drugi etap polega na wypisaniu w postaci jawnej tych pozycji ciągu danego zera, na których różni się on od ciągu wszystkich jedynek dla kolumn, oraz pozycji ciągu danej jedynki, na których różni się on od ciągów wszystkich zer (dla wierszy). W następnym etapie należy znaleźć minimalny zbiór tak zredukowanych ciągów, pozwalający na odróżnienie wszystkich zer od wszystkich jedynek. W rozpatrywanym przypadku, minimalny zbiór będzie obejmował:

- dla kolumn, ciągi: $[(--00--), (1-----)],$
 $[(----01), (1-----)], [(--0--1), (1-----)],$
- dla wierszy, ciąg: $[(--1--), (---1--)].$

Deszyfrację dla kolumn (zer) przeprowadzamy w konwencji makstermowej. Otrzymamy trzy następujące formuły:

$$F = (c + d) \bar{a},$$

$$F = (e + \bar{f}) \bar{a},$$

$$F = (c + \bar{f}) \bar{a}.$$

Formuły te zawierają jednakową liczbę literałów i każda z nich pozwala na jednoznaczne odróżnienie wszystkich zer od wszystkich jedynek funkcji. Nie stanowią one jeszcze rozwiązania ostatecznego - brakuje wyników deszyfracji wierszy. W tym przypadku jedynki funkcji deszyfrację przeprowadzamy w konwencji mintermów. Otrzymujemy następujący rezultat

$$F = c + d.$$

Jest to formuła o najmniejszej liczbie literałów i przyjmujemy ją jako ostateczny rezultat minimalizacji funkcji. Rezultat ten uzyskano w wyniku postępowania quasioptymalnego; można jednak wykazać, że w pełni optymalna procedura minimalizacji doprowadziłaby również do tej samej formuły. W większości przypadków efekty obliczeń quasioptymalnych i optymalnych pokrywają się, ewentualnie występuje najczęściej nieistotna różnica w liczbie literałów uzyskanych formuł. Jednocześnie dla procedury w pełni optymalnej konieczne byłoby podanie wszystkich możliwych zbiorów numerów pozycji, na których występują różnice, co znacznie wydłuża tok obliczeń i komplikuje algorytm.

3.9. Zjawisko hazardu

3.9.1. Hazard statyczny

Na pierwszym poziomie w układach realizujących normalną postać funkcji jest pojedyncza, najczęściej wielowejsciowa bramka sumy lub iloczynu. Do bramki tej dochodzą sygnały z bramek poziomu drugiego, które z kolei są sterowane sygnałem zmiennych wejściowych lub ich negacji. Z uwagi na skończoną szybkość propagacji sygnałów fizycznych, może się zdarzyć, że sygnały dochodzące do bramki, np. iloczynu pierwszego poziomu, reprezentujące daną zmienną w postaci afirmacyjnej i negacji, są względem siebie wyraźnie przesunięte w czasie (rys. 3.39). W tej sytuacji jeśli wysterowanie ewentualnych pozostałych wejść bramki iloczynu osiągnie stan 1, to nie jest spełniona formuła tożsamościowa

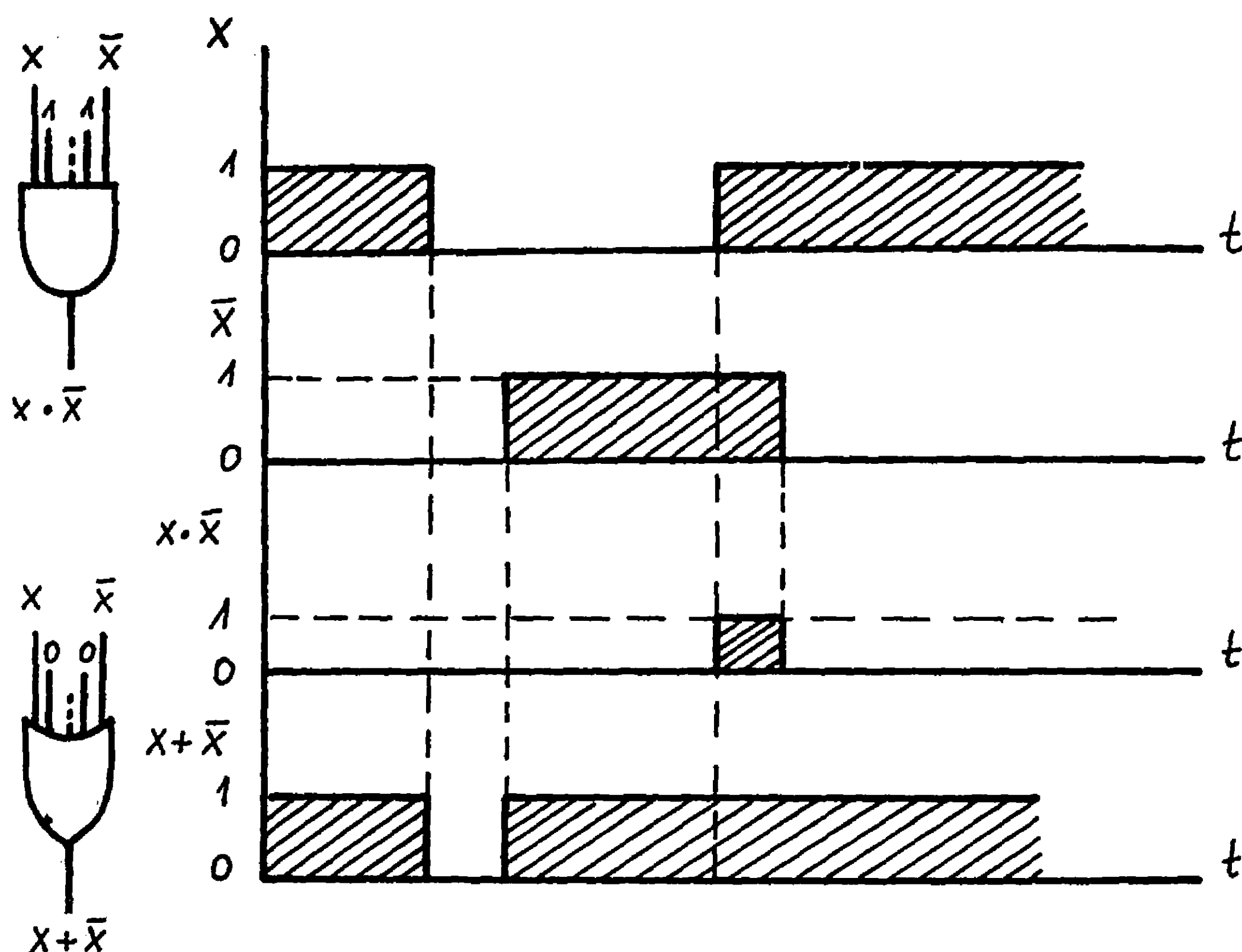
$$x \cdot \bar{x} = 0.$$

Na wyjściu zostanie wygenerowana w okresie przejściowym "krótka jedynka" (rys. 3.39), co może doprowadzić do wadliwego wysterowania dalszych bloków systemu cyfrowego. Analogiczna sytuacja będzie miała miej-

sce, jeśli na pierwszym poziomie występuje bramka sumy. Przyjęcie przez ewentualne pozostałe wejścia stanu logicznego 0 powoduje, że na wyjściu może się pojawić "krótkie zero", z uwagi na nie spełnienie warunku:

$$x + \bar{x} = 1.$$

Omawiane zjawisko nosi nazwę hazardu statycznego i jest charakterystyczne dla normalnych postaci realizowanych funkcji boolowskich.



Rys. 3.39. Wykresy wyjaśniające powstawanie zjawiska hazardu statycznego

- Hazard statyczny względem określonej zmiennej można wykryć w analitycznej postaci funkcji. Jeśli funkcja dana w alternatywnej postaci normalnej względem zmiennej x przyjmuje następującą formę zapisu

$$f = A x + B \bar{x} + C, \quad (1)$$

to dla przypadku gdy $A = 1$, $B = 1$ i $C = 0$, ($C = 0$ jest odpowiednikiem zerowego wystawienia wejść bramki pierwszego poziomu, z wyjątkiem wejść x oraz $\bar{x}$) może wystąpić hazard typu $x + \bar{x} = 1$, a więc "krótkie zero". Wyrazy A, B, C są funkcjami pozostałych zmiennych. Przykładowo, funkcję

$$f(w,x,y,z) = x \bar{y} \bar{z} + \bar{w} \bar{y} z + w x y,$$

można zapisać względem zmiennej z w postaci

$$f = (\bar{w} \bar{y}) z + (x \bar{y}) \bar{z} + w x y.$$

Jeśli $w = 0$, $x = 1$, $y = 0$, to funkcja ma hazard względem z , bowiem

$$f(0,1,0,z) = z + \bar{z}.$$

Dla omawianej funkcji, wyrażenia A, B, C wynoszą odpowiednio:

$$A = \bar{w} \bar{y}$$

$$B = x \bar{y}$$

$$C = w x y.$$

W podobny sposób można powyższą funkcję przekształcić względem zmiennych y i w .

Dla zmiennej y

$$f(w,x,y,z) = (w x) y + (x \bar{z} + \bar{w} z) \bar{y};$$

hazard wystąpi, gdy $w = 1$, $x = 1$, $z = 0$, bowiem

$$f(1,1,y,0) = y + \bar{y}.$$

Wyrażenia A, B, C wyniosą odpowiednio:

$$A = w x$$

$$B = x \bar{z} + \bar{w} z$$

$$C = 0.$$

Dla zmiennej w

$$f(w,x,y,z) = (x y) w + (\bar{y} z) \bar{w} + x \bar{y} \bar{z},$$

hazard nie wystąpi, bowiem gdy $x = 1$, $y = 1$, to $\bar{y} = 0$ i nie da się funkcji sprowadzić do postaci $w + \bar{w}$, tzn. nie może być jednocześnie: $A = x y = 1$, $B = \bar{y} z = 1$ oraz $C = x \bar{y} \bar{z} = 0$, zawsze $A B \bar{C} \neq 1$. Na rysunku 3.40 przedstawiono tablicę Karnaugh omawianej funkcji. Wynika z niej, że hazard występuje w tych przypadkach, gdy sklejone pary mają kratki sąsiednie nie objęte sklejaniem. Gdyby dodatkowo skleić te minitermy, to hazard zostałby wyeliminowany, gdyż jak łatwo się przekonać zawsze byłoby

$$A B \bar{C} \neq 1.$$



Dodatkowe sklejenia zaznaczono na rysunku liniami przerywanymi. Analitycznie oznacza to wprowadzenie do ogólnej postaci funkcji (1)

$$f = A x + B \bar{x} + C,$$

wx \ yz	00	01	11	10
00		1		
01	1	1		
11	1		1	1
10				

Rys. 3.40. Tablica Karnaugh funkcji z hazardem względem zmiennych z oraz y

składnika łączącego łańcuch sąsiednich mintermów (jedynek w tablicy Karnaugh) funkcji i wynoszącego $A \cdot B$. Jest to intuicyjnie zrozumiałe, bowiem skoro $A = 1$ i $B = 1$, to również $A \cdot B = 1$ i nie jest możliwe wygenerowanie "krótkiego zera". Postać funkcji bez hazardu względem zmiennej x będzie więc następująca

$$f = A x + B \bar{x} + C + A B.$$

Składnik $A B$ nie zmienia wartości funkcji, bowiem

$$\begin{aligned} f &= A x + B \bar{x} + C = A x(B + \bar{B}) + B \bar{x}(A + \bar{A}) + C = \\ &= A B x + A B \bar{x} + A x + B \bar{x} + C = A x + B \bar{x} + C + A B. \end{aligned} \quad (2)$$

Wracając do rozpatrywanej funkcji, hazard względem z eliminujemy, dodając człon

$$A B = (\bar{w} \bar{y})(x \bar{y}) = \bar{w} x \bar{y} \quad (\text{rys. 3.40}),$$

natomiast hazard względem y , dodając człon:

$$A B = (w x)(x \bar{z} + \bar{w} z) = w x \bar{z}.$$

Ostatecznie postać funkcji pozbawiona hazardu będzie następująca

$$f = x \bar{y} \bar{z} + \bar{w} \bar{y} z + \bar{w} x \bar{y} + w x \bar{z} + w x y.$$

Taki sam rezultat otrzymalibyśmy poddając postać kanoniczną funkcji skracaniu metodą tabulacyjną Quine'a McCluskey'a, bez redukcji w tablicy implikantów (implimentów) prostych. Postacie skrócone funkcji są więc postaciami nie zawierającymi hazardu. Wynika to bezpośrednio z algorytmu przedstawionego w punkcie 2.12.2, gdyż kolumny kolejnych implikantów implimentów są generowane z zapasem przez sklejenie grupy z indeksem " $i - 1$ " z grupą o indeksie " i " oraz grupy o indeksie

"i" z grupą o indeksie "i + 1". W ten sposób mamy gwarancję łańcuchowego sklejanie wszystkich wyrażeń sąsiednich. Łańcuch ten zostaje dopiero "rozerwany" w procesie poszukiwania formuły minimalnej w tablicy implikantów (implicentów) prostych, a więc eliminowania członów typu $A B (1)$, zbędnych z punktu widzenia tablicy wartości logicznych funkcji (2).

Na zasadzie dualności, analogiczne rozważania można przeprowadzić dla koniunkcyjnej postaci normalnej. Tak więc hazard typu "krótka jedynka" (rys. 3.39) wystąpi w funkcji względem zmiennej x , jeśli da się ją zapisać w postaci

$$f = (A + x)(B + \bar{x})C \quad (3)$$

oraz gdy wyrażenia A , B , C mogą przyjąć wartości: $A = 0$, $B = 0$, $C = 1$. Wtedy

$$f = x \bar{x}.$$

Eliminację hazardu typu "krótka jedynka" przeprowadzamy wprowadzając dodatkowo iloczyn $A + B$, tzn. przekształcając funkcję do postaci

$$f = (A + x)(B + \bar{x}) C = (A + x)(B + \bar{x}) C (A + B). \quad (4)$$

Wtedy gdy $A = 0$ i $B = 0$, $A + B = 0$ i nie może wystąpić efekt "krótkiej jedynki".

Przykład 3.16. Wyeliminować hazard z funkcji $f(w,x,y,z)$

$$f = (\bar{w} + y + z)(\bar{x} + y + \bar{z})(\bar{w} + x + \bar{y}).$$

Funkcję powyższą daje się wprowadzić do postaci (3) dla dwóch zmiennych: z oraz y . Otrzymamy bowiem:

- dla z

$$f = [(\bar{w} + y) + z][(\bar{x} + y) + \bar{z}](\bar{w} + x + \bar{y}),$$

przy czym hazard wystąpi, gdy $w = 1$, $x = 1$, $y = 0$ gdyż

$$f(1,1,0,z) = z \bar{z},$$

- dla y (korzystając z prawa rozdzielności)

$$f = [(\bar{w} + z)(\bar{x} + z) + y][(\bar{w} + x) + \bar{y}],$$

przy czym hazard wystąpi, gdy $w = 1$, $x = 0$, $z = 0$, gdyż

$$f(1,0,y,0) = y \bar{y}.$$

Dla zmiennej x również można uzyskać postać (3), ale jak łatwo sprawdzić nie da się tu osiągnąć warunku $A = 0$, $B = 0$ i $C = 1$.

Eliminację hazardu uzyskujemy uzupełniając funkcję członem $A + B$ (4). Dla zmiennej z wynosi on

$$(\bar{w} + y) + (\bar{x} + y) = \bar{w} + \bar{x} + y,$$

natomiast dla zmiennej y

$$(\bar{w} + z)(\bar{x} + z) + (\bar{w} + x) = \bar{w} + x + z.$$

Ostateczna formuła zapisu funkcji, nie zawierająca hazardu, będzie miała postać

$$f = (\bar{w} + y + z)(\bar{x} + y + \bar{z})(\bar{w} + x + \bar{y})(\bar{w} + \bar{x} + y)(\bar{w} + x + z).$$

Łatwo sprawdzić, że postać skrócona funkcji uzyskana metodą Quine'a McCluskey'a będzie taka sama.

3.9.2. Hazard dynamiczny

Poddając postać normalną funkcji boolowskiej procesowi faktoryzacji (p. 2.13), na ogół zwiększa się liczbę poziomów propagacji dla sygnałów wejściowych. W konsekwencji, sygnał do bramki wyjściowej może się dostać co najmniej trzema torami opóźnień: dwa z nich są identyczne jak dla omawianego już hazardu statycznego, trzeci i dalsze wiążą się z procesem faktoryzacji. Obecność hazardu dynamicznego można wykryć, jeśli dla pewnej kombinacji argumentów, funkcję da się sprowadzić do postaci szczególnej, typu:

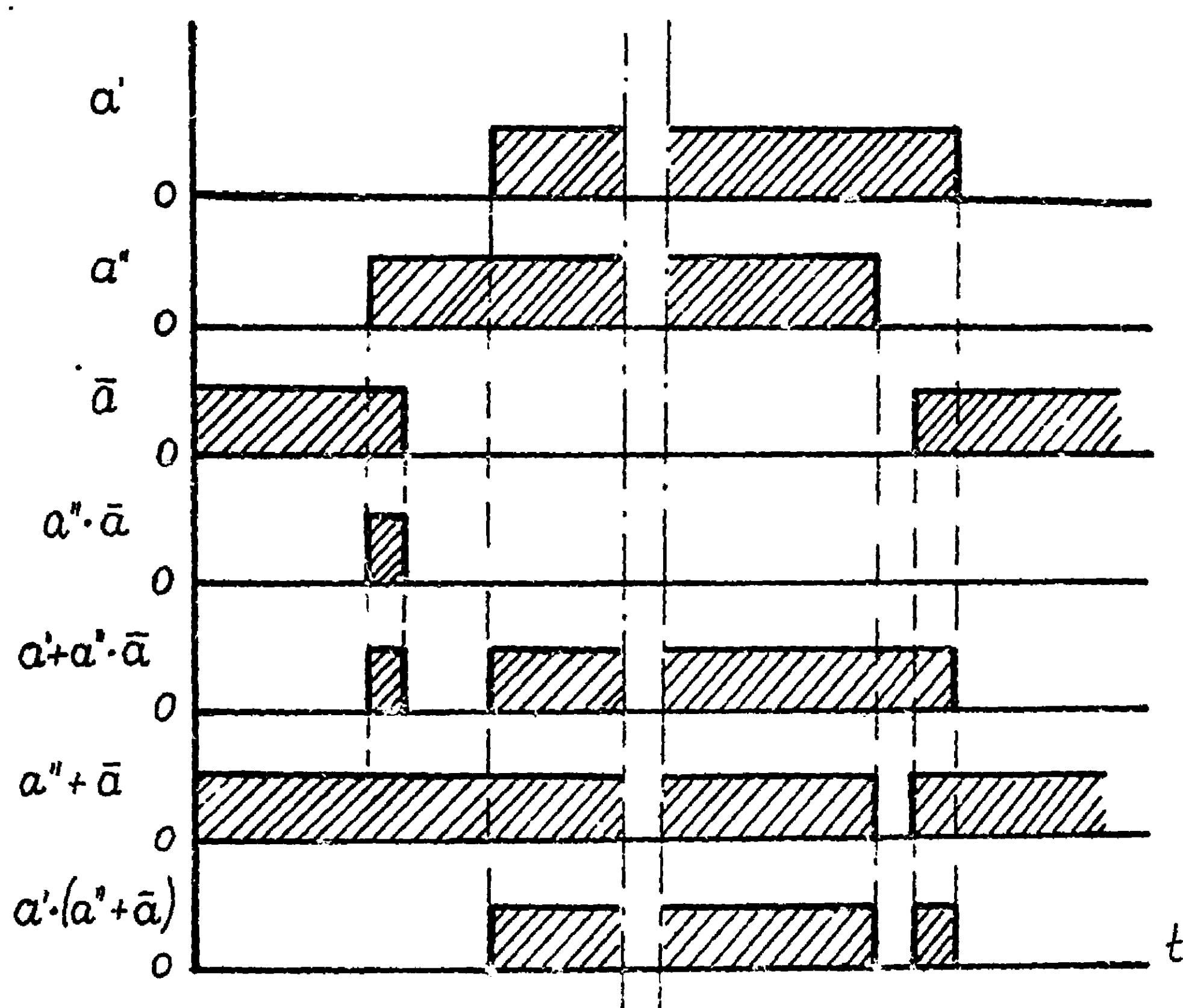
$$x + x \bar{x} \quad \text{lub} \quad x(x + \bar{x}), \quad (5)$$

ewentualnie:

$$\bar{x} + x \bar{x} \quad \text{lub} \quad \bar{x}(x + \bar{x}). \quad (6)$$

Postacie (5) i (6) są równoważne i przechodzą wzajemnie w siebie w wyniku podstawienia $y = \bar{x}$, w związku z czym nie ma potrzeby osobnego ich rozpatrywania. Na rysunku 3.41 przedstawiono przebiegi czasowe wy-

jaśniające efekt powstawania zjawiska hazardu dynamicznego. Sygnał wejściowy a jest podawany na wyjście dwoma drogami o różnych opóźnieniach, co zaznaczono symbolicznie jako a' oraz a'' . W wyniku koniunkcji sygnału a'' z sygnałem $\bar{a}$, może być wygenerowana "krótka jedynka" jak w przypadku hazardu statycznego. Z kolei tak wygenerowana "krótka jedynka" dodając się na wyższym poziomie do sygnału a' , daje hazard dynamicznego charakterystyczny przy przejściu sygnału a z wartości 0 na 1. Przejście to w sekwencji czasowej daje serię impulsów 1 0 1, przy czym ostatnia jedynka jest stabilna (rys. 3.41 dla $a' + a'' \bar{a}$).



Rys. 3.41. Przebiegi czasowe wyjaśniające powstawanie zjawiska hazardu dynamicznego

Suma sygnału a'' oraz $\bar{a}$, tzn. $a'' + \bar{a}$ może wprowadzić hazard typu "krótkie zero". Podając tę sumę na bramkę koniunkcji (iloczynu) z sygnałem a' , uzyskamy efekt hazardu dynamicznego charakterystyczny dla zmiany sygnału a z 1 na 0. Zmiana ta w sekwencji czasowej daje serię impulsów 0 1 0, przy czym ostatnie zero jest stabilne (rys. 3.41, dla $a'(a'' + \bar{a})$).

W ogólnym przypadku dla układów wielopoziomowych, w wyniku hazardu dynamicznego, zmiana sygnału wejściowego z 0 na 1 może wygenerować serię "krótkich jedynek", poprzedzających stan stabilnej jedynki, natomiast przejście sygnału z 1 na 0 może wygenerować serię "krótkich zer", poprzedzających stan stabilnego zera. Zjawiska te są na ogół bardzo szkodliwe, dlatego też faktoryzację funkcji należy przeprowadzać, mając na uwadze wymienione efekty.

Nie ma ogólnych metod pozwalających eliminować hazard z wyrażeń faktoryzowanych. Stwierdzono jedynie, że stosowanie następujących przekształceń:

- 1) prawo łączności,
- 2) prawo przemienności,
- 3) prawo rozdzielności,
- 4) prawa de Morgana,
- 5) $A + A B = A$, $A(A + B) = A$,
- 6) $A + \bar{A} B = A + B$, $A(\bar{A} + B) = A B$,

nie zmienia właściwości funkcji ze względu na hazard, rozumiany niezależnie od tego czy ma on charakter statyczny, czy też dynamiczny. Pozwala to na faktoryzację wyrażeń boolowskich bez ryzyka wprowadzania hazardu.

3.10. Testowanie układów kombinacyjnych

3.10.1. Warunki testowania układu

Testowanie układu kombinacyjnego w ogólnym przypadku polega na doświadczalnym sprawdzaniu zgodności jego funkcjonowania z tabelą wartości logicznych. Poszczególne wartości wektora wejściowego noszą przy tym nazwę testów. Ogólnie dla n wejść można podać 2^n testów, gdyż tyle różnych wartości ma n -wymiarowy wektor o składowych binarnych. Zakładając typy uszkodzeń, liczbę testów niezbędnych do ich wykrycia można zredukować. Weźmy przykładowo dwuwejściową bramkę NAND. Zbiór testów zawiera 4 elementy reprezentowane przez wartości 00, 01, 10, 11; ale po to by sprawdzić, czy jedno z wejść jest zwarte do masy, co odpowiada trwałej wartości zero, wystarczy podać na nie sygnał 1, przy jednoczesnym wystereowaniu drugiego wejścia również sygnałem 1. Jeśli na wyjściu nie pojawi się zero, to bramka jest uszkodzona. Test 11 może być traktowany jako test sprawdzający uszkodzenia wejść typu: trwałe zero. Z kolei trwałą jedynkę na wejściu można wykryć przykładając do tego wejścia sygnał zero, przy jednoczesnym przyłożeniu 1 do wejścia sąsied-



niego. Test 01 wykrywa więc uszkodzenie typu: trwała jedynka dla jednego z wejść, natomiast test 10 - dla drugiego. Jeśli założymy, że możliwe jest również uszkodzenie wyjścia bramki typu trwałe zero lub trwała jedynka, to po to, by to sprawdzić należy wysterować jej wejścia stosownie do wymagań detekcji uszkodzenia, tzn. tak by na wejściu bramki dobrej był sygnał przeciwny do założonego w uszkodzeniu. A więc precyzując prawdopodobne typy uszkodzeń i ograniczając się do nich, można zmniejszyć liczbę testów niezbędnych do oceny poprawności pracy układu. Wyłaniają się przy tym dwa warunki konieczne do tego, by test był skuteczny:

- 1) sygnał w miejscu uszkodzenia musi mieć wartość przeciwną do tej, jaka wynika z założonego uszkodzenia,
- 2) przynajmniej jedno z wyjść układu musi reagować na zmianę sygnału w miejscu uszkodzenia.

Obydwa warunki wynikają z przesłanek intuicyjnie wyczuwalnych: inaczej uszkodzenie nie byłoby wykrywalne. Określenie testu dla konkretnego uszkodzenia może być dokonane poprzez analizę funkcji boolowskich opisujących układ, bądź też bezpośrednio ze schematu logicznego. W pierwszym przypadku wykorzystuje się formułę zwaną różnicą boolowską, w drugim - dokonuje się tego tzw. metodą pobudzania ścieżek. Niżej zostaną rozpatrzone obydwie metody.

3.10.2. Metoda różnic boolowskich

Operator EXCLUSIVE-OR jest również nazywany operatorem nierównoważności, gdyż przyporządkowuje dwóm argumentom wartość jedynki logicznej tylko wówczas, gdy są one różne. Jeśli argumentami będą funkcje boolowskie, to dla wyróżnionej zmiennej x_i operacja nierównoważności pozwoli określić, czy funkcje te są rozróżnialne czy też nie. Może to być wykorzystane w procesie detekcji uszkodzeń, gdyż jeśli jako argumentów operacji nierównoważności użyjemy tej samej funkcji, ale niezależnie raz dla $x_i = 0$ i drugi raz dla $x_i = 1$, to uzyskamy formułę boolowską o właściwościach detekcyjnych ze względu na tę zmienną. Dla tych kombinacji wartości pozostałych zmiennych, dla których formuła ta przyjmuje wartość 1, zmiana x_i wywołuje zawsze zmianę wartości funkcji, natomiast gdy wartość formuły wynosi 0, funkcja jest niezależna od zmian x_i . Właściwość ta umożliwia określenie testu dla danego uszkodzenia. Przedstawioną formułę przyjęto nazywać różnicą boolowską, a jej postać analityczną podaje definicja 3.1.

Definicja 3.1. Różnicą boolowską funkcji $f(x_{n-1}, \dots, x_i, \dots, x_0)$ względem zmiennej x_i ($i = 0, 1, \dots, n-1$), nazywamy operację EXCLUSIVE-OR, wykonaną na argumentach: $f(x_{n-1}, \dots, x_i = 1, \dots, x_0)$, $f(x_{n-1}, \dots, x_i = 0, \dots, x_0)$. Zapisujemy to następująco:

$$\frac{df}{dx_i} = f(x_{n-1}, \dots, x_i = 1, \dots, x_0) \oplus f(x_{n-1}, \dots, x_i = 0, \dots, x_0).$$

Do oznaczenia różnicy boolowskiej zastosowano symbol pochodnej funkcji. Wynika to z pewnej analogii, gdyż można sprawdzić, że:

$$\frac{d(x_1 x_2)}{dx_1} = x_2,$$

$$\frac{d(x_1 + x_2)}{dx_1} = 1,$$

$$\frac{dx}{dx} = 1,$$

$$\frac{dk}{dx} = 0 \quad (k \text{ nie zależy od } x).$$

Rezultaty te są analogiczne jak w przypadku działań w klasycznym obliczaniu pochodnej iloczynu czy też sumy. Łatwo sprawdzić, że różnica boolowska ma następujące własności:

$$\frac{d\bar{f}}{dx_i} = \frac{df}{dx_i},$$

$$\frac{df}{d\bar{x}_i} = \frac{df}{dx_i}.$$

Stąd wynika, że:

$$\frac{d\bar{f}}{d\bar{x}_i} = \frac{df}{d\bar{x}_i} = \frac{d\bar{f}}{dx_i} = \frac{df}{dx_i}. \quad (1)$$

Różnica boolowska jest więc niezmiennicza ze względu na operację dopełnienia zarówno funkcji, jak i zmiennej. Ponadto w procesie wyznaczania testów pomocne są wyrażenia określające różnice boolowskie operacji wykonywanych na funkcjach boolowskich g , h tych samych zmiennych:

$$\frac{d}{dx_i} (g \cdot h) = g \frac{dh}{dx_i} \oplus h \frac{dg}{dx_i} \oplus \frac{dg}{dx_i} \frac{dh}{dx_i}, \quad (2)$$

$$\frac{d}{dx_i} (g + h) = \bar{g} \frac{dh}{dx_i} \oplus \bar{h} \frac{dg}{dx_i} \oplus \frac{dg}{dx_i} \frac{dh}{dx_i}, \quad (3)$$

$$\frac{d}{dx_i} (g \oplus h) = \frac{dg}{dx_i} \oplus \frac{dh}{dx_i}. \quad (4)$$

Formuły powyższe łatwo otrzymać bezpośrednio z definicji różnicy boolowskiej. Również bezpośrednio z definicji wynika prawdziwość następującej zależności

$$\frac{d}{dx_i} \left(\frac{df}{dx_j} \right) = \frac{d}{dx_j} \left(\frac{df}{dx_i} \right). \quad (5)$$

Jak już wspomniano, wartość 1 różnicy boolowskiej oznacza, że funkcja zmienia swą wartość zależnie od zmiany rozpatrywanej zmiennej. W skrajnym przypadku, jeśli $\frac{df}{dx_i} = 1$ niezależnie od stanu pozostałych zmiennych, to i funkcja zmienia swą wartość zawsze przy zmianie wartości x_i , niezależnie od stanu innych zmiennych. Analogicznie, jeśli różnica boolowska $\frac{df}{dx_i} = 0$ - niezależnie od stanu pozostałych zmiennych, to i funkcja jest od x_i niezależna. Z przypadkiem takim spotkaliśmy się już podczas omawiania minimalizacji zespołu funkcji (przykład 3.10, zmienna w). Różnicę boolowską można więc ogólnie traktować jako wrażliwość binarną danej funkcji na zmiany argumentu x_i . Koniunkcja tej wrażliwości ze zmienną w postaci stosownej do typu uszkodzenia wejścia (stała jedynka, lub stałe zero), określa formułę, której mintermy są testami dla tego uszkodzenia. Postać zmiennej wiąże się z koniecznością spełnienia warunków wykrywalności uszkodzenia sprecyzowanych w podpunkcie 3.10.1. Oznaczając uszkodzenie wejścia typu: stałe zero przez $s = 0$, natomiast typu: stała jedynka przez $s = 1$, otrzymamy następujące postacie ogólne formuł testujących:

$$F_0 = x_i \frac{df}{dx_i} \quad - \text{ dla uszkodzenia } s = 0, \quad (6)$$

$$F_1 = \bar{x}_i \frac{df}{dx_i} \quad - \text{ dla uszkodzenia } s = 1. \quad (7)$$

Zakładamy przy tym, że uszkodzenia dotyczą tylko przewodów bezpośrednio dostępnych dla zmiennych x_i , czyli wejść. Jeśli chcemy określić formuły testujące dla uszkodzeń w przewodach wewnętrznych i niedostępnych przez zaciski wejściowe musimy wyrazić wartość sygnału w punkcie uszkodzenia jako funkcję argumentów wejściowych i obliczyć różnicę boolowską dla tak wprowadzonej zmiennej zależnej. Funkcja $f(x_{n-1}, \dots, x_0)$ musi być przekształcona do postaci: $f[x_{n-1}, \dots, x_0; g(x_{n-1}, \dots, x_0)]$, gdzie g oznacza nowowprowadzoną zmienną, zależną od tych samych argumentów. Formuły testujące przyjmą wówczas postać analogiczną do postaci (6) i (7):

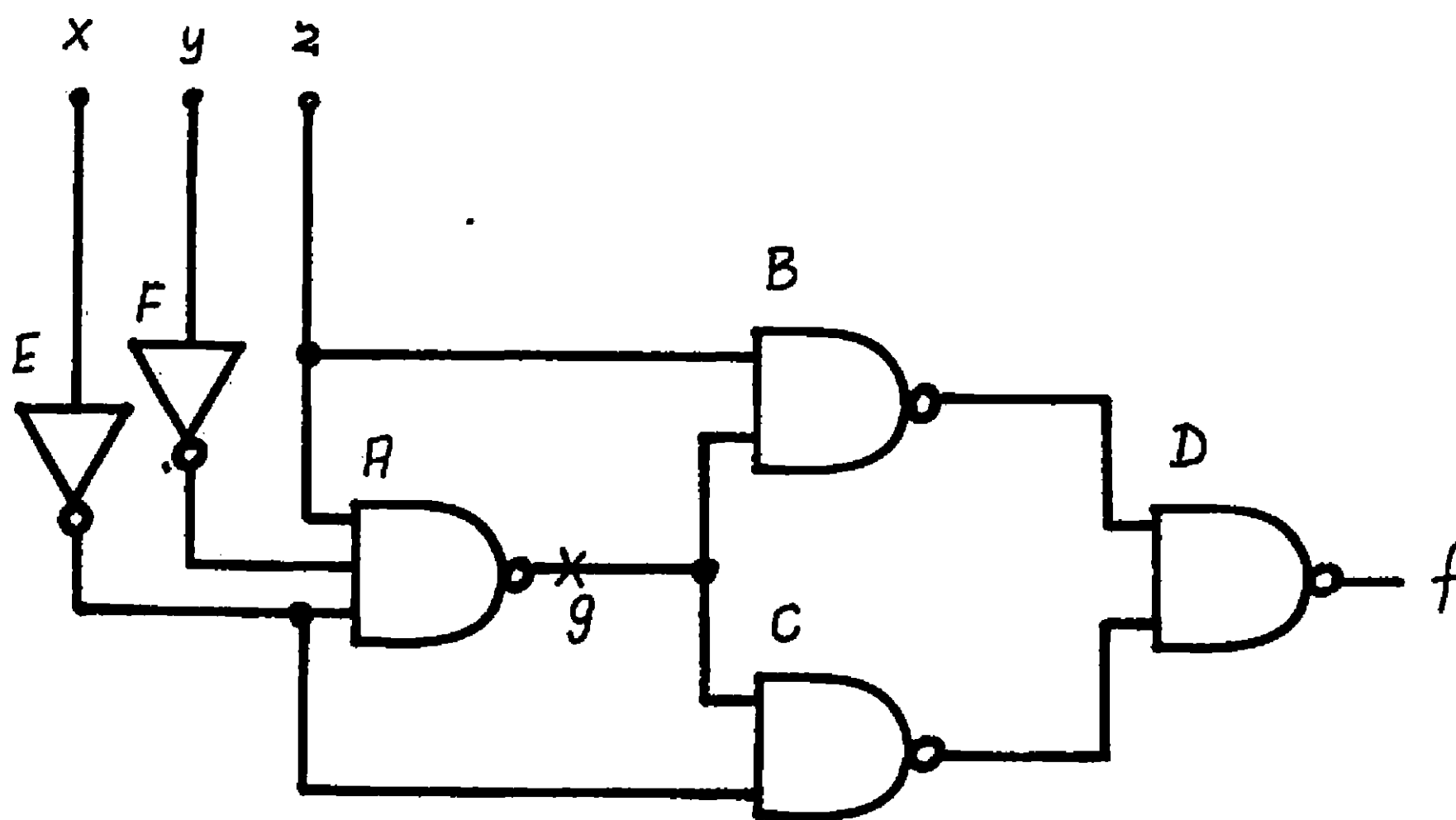
$$F_0 = g \frac{df}{dg} \quad - \text{ dla uszkodzenia } s = 0, \quad (8)$$

$$F_1 = \bar{g} \frac{df}{dg} \quad - \text{ dla uszkodzenia } s = 1. \quad (9)$$

Mintermy formuł (6) i (7) lub (8) i (9) są testami odpowiednich uszkodzeń.

Przykład 3.17. Określić testy dla uszkodzeń typu $s = 0$ i $s = 1$ w punkcie g , schematu przedstawionego na rysunku 3.42 metodą różnic boolowskich. Układ realizuje funkcję

$$f = xz + yz + \bar{x}\bar{y} + \bar{x}z.$$



Rys. 3.42. Schemat układu z przykładu 3.17

Zmienna zależna w punkcie g wyrazi się wzorem

$$g = x + y + \bar{z}.$$

Funkcja $f(x, y, z, g)$ przyjmie postać

$$f = z g + \bar{x} g = (\bar{x} + z)g.$$

Różnicę boolowską $\frac{df}{dg}$ można wyznaczyć korzystając z wyrażenia (2) dla iloczynu funkcji. Wprowadźmy oznaczenia:

$$A = \bar{x} + z, B = g.$$

Dla tych oznaczeń, można napisać:

$$\frac{dA}{dg} = 0, \frac{dB}{dg} = 1,$$

i stąd:

$$\frac{df}{dg} = A \frac{dB}{dg} \oplus B \frac{dA}{dg} \oplus \frac{dA}{dg} \frac{dB}{dg} = \bar{x} + z.$$

Formuły określające testy będą mieć postać:

- dla uszkodzenia $s = 0$

$$F_0 = g \frac{df}{dg} = (x + y + \bar{z})(\bar{x} + z) =$$

$$= xz + \bar{x}y + yz + \bar{x}\bar{z} = \sum (0, 2, 3, 5, 7),$$

- dla uszkodzenia $s = 1$

$$F = \bar{g} \frac{df}{dg} = \bar{x} \bar{y} z (\bar{x} + z) = \bar{x} \bar{y} z = \sum (1).$$

Uszkodzenie $s = 0$, wykrywa więc zbiór złożony z pięciu testów:

$$F_0 = \left\{ \begin{array}{ccc} x & y & z \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{array} \right\},$$

natomiast uszkodzenie $s = 1$, zbiór jednoelementowy $F_1 = \{0 \ 0 \ 1\}$.

Formuły testujące można również zdefiniować w konwencji makstermowej, stosując operator równoważności. Postępowanie jest wówczas dualne do omówionego wyżej, przy czym różnicę boolowską zastępujemy jej dopełnieniem. Dopełnienie różnicy boolowskiej będzie mieć następującą postać

$$\overline{\left[\frac{df}{dx_i}\right]} = \frac{Df}{Dx_i} = f(x_{n-1}, \dots, x_i = 1, \dots, x_0) \odot f(x_{n-1}, \dots, x_i = 0, \dots, x_0),$$

natomiast formuły testujące dla zmiennych wejściowych przyjmą postać:

$$\begin{aligned} F_1 &= x_i + \frac{Df}{Dx_i} && - \text{dla uszkodzeń } s - 1, \\ F_0 &= \bar{x}_i + \frac{Df}{Dx_i} && - \text{dla uszkodzeń } s - 0. \end{aligned}$$

Analogicznie dla przewodów wewnętrznych:

$$\begin{aligned} F_1 &= g + \frac{Df}{Dg} && - \text{dla uszkodzeń } s - 1, \\ F_0 &= \bar{g} + \frac{Df}{Dg} && - \text{dla uszkodzeń } s - 0. \end{aligned}$$

Przykład 3.18. Określić testy dla uszkodzeń układu podanego w przykładzie 3.17, metodą dopełnienia różnicy boolowskiej.

Korzystając z uprzednio otrzymanych rezultatów można napisać:

$$\frac{Df}{Dg} = \overline{\bar{x} + z} = x \bar{z}.$$

Formuła testująca uszkodzenie $s - 1$ będzie mieć postać

$$F_1 = g + \frac{Df}{Dg} = x + y + \bar{z} + x \bar{z} = x + y + \bar{z} = \Pi(1),$$

natomiast formuła dla uszkodzenia $s - 0$, wyniesie

$$F_0 = \bar{g} + \frac{Df}{Dg} = \bar{x} \bar{y} z + x \bar{z} = \Pi(0, 2, 3, 5, 7).$$

Po deszyfracji według konwencji makstermowej otrzymamy następujące zbiory testów:

$$F_1 = \begin{matrix} & x & y & z \\ & 0 & 0 & 1 \end{matrix}; \quad F_0 = \begin{matrix} & x & y & z \\ \begin{matrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{matrix} & \begin{matrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \end{matrix} & \begin{matrix} 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{matrix} & \begin{matrix} 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{matrix} \end{matrix}.$$

Otrzymany rezultat jest identyczny z uzyskanym z wykorzystaniem różnicy boolowskiej.

3.10.3. Metoda pobudzania ścieżek

Metoda pobudzania ścieżek jest układową adaptacją metody różnic boolowskich, poszerzoną o możliwości wynikające ze znajomości topologii łączy. Zasadnicza przewaga tej metody nad metodą analityczną polega na możliwości uwzględniania w procesie testowania wielu wyjść na raz. Nie było to bezpośrednio możliwe poprzednio, a operowanie zbiorem różnic boolowskich określanych dla każdego wyjścia nie daje użytecznych rezultatów z uwagi na duże komplikacje analityczne. Generalnie, metoda ta sprowadza się do takiego wysterowania wejść poszczególnych bramek, by umożliwić sterowanie sygnałem testującym uszkodzenie przynajmniej jednego wyjścia. Drogi propagacji sygnału testującego są określane jako ścieżki, a wysterowanie bramek umożliwiające, tę propagację jest nazwane pobudzeniem ścieżek. Przykładowo jeśli sygnał jest propagowany przez wejście x_i wielowejsciowej bramki NOR, to wszystkie pozostałe wejścia muszą mieć wartość zero, gdyż inaczej zmiana sygnału x_i nie wywoła zmian na wyjściu tej bramki:

$$\overline{x_{n-1} + \dots + x_i + \dots + x_0} \rightarrow \overline{0 + \dots + x_i + \dots + 0}.$$

Pobudzenie ścieżki utożsamia się w tym fragmencie z nadaniem pozostałym wejściom stanu logicznego 0. Jeśli to nie jest możliwe konieczne jest wybranie innej ścieżki. Na przykład dla układu pokazanego na rysunku 3.42, test $F_0 = 1 \ 1 \ 1$ uniemożliwia propagację sygnału z punktu g poprzez bramkę C, a więc ścieżka musi biec przez bramkę B, natomiast, testy dla których $z = 0$ eliminują ze ścieżki bramkę B; tym razem aktywną dla sygnału z punktu g musi być bramka C.

Reasumując można stwierdzić, że poszukiwanie testów wykrywających założone uszkodzenie wymaga:

- 1) wymuszenia w miejscu uszkodzenia sygnału przeciwnego do tego jaki wynika z założonego uszkodzenia,
- 2) określenia wszystkich możliwych ścieżek, którymi sygnał może być propagowany do wyjść układu,
- 3) pobudzenia tych ścieżek, umożliwiającego propagację sygnału z miejsca uszkodzenia do przynajmniej jednego wyjścia,
- 4) wyboru testów umożliwiających propagację sygnału z miejsca uszkodzenia do wyjść, a więc testów niesprzecznych z warunkami pobudzenia przynajmniej jednej ścieżki.

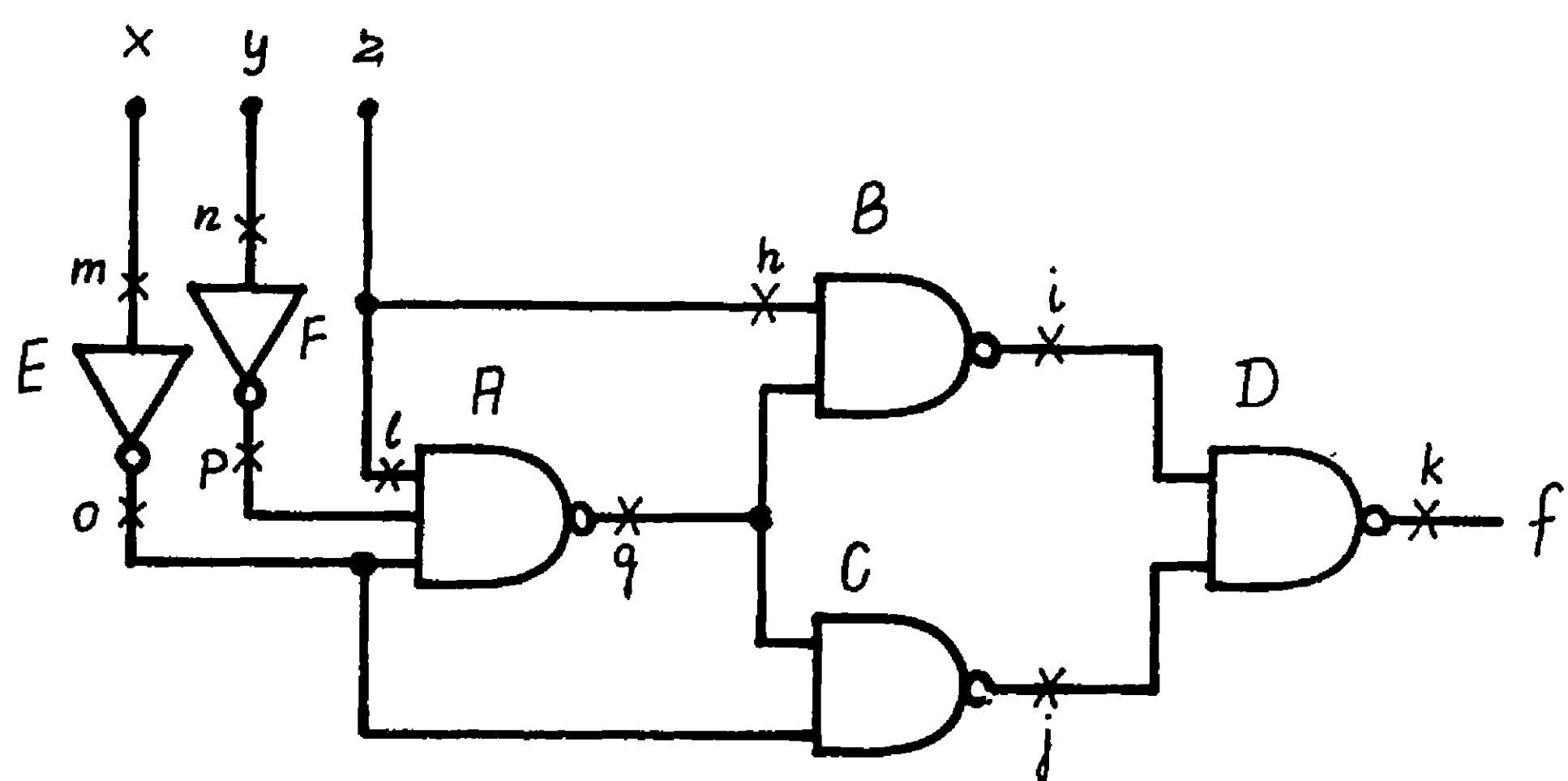
Przykład 3.19. Określić testy dla uszkodzenia typu $s - 0$ i $s - 1$ w punkcie g schematu podanego na rysunku 3.42.

Zacznijmy od określenia ścieżek dla testu uszkodzenia typu $s - 1$. Należy wymusić w punkcie g sygnał logiczny przeciwny, a więc 0. Jest to możliwe tylko dla jednej kombinacji sygnałów wejściowych $x y z$: 0 0 1. Jednocześnie zostają pobudzone ścieżki $g - B - D - f$ oraz $g - C - D - f$, gdyż test wymusza sygnał 1 na pozostałych wejściach bramek NAND C i B .

Znalezienie testów dla uszkodzeń typu $s - 0$, wymaga wymuszenia jedynki na wyjściu bramki A . Jedynkę wymusza logiczne 0 na dowolnym wejściu tej bramki. Jednak jak już wspomniano, testy dla których $z = 0$ eliminują ścieżkę $g - B - D - f$. Propagacja sygnału odbywa się wówczas ścieżką $g - C - D - f$. Wartość 0 zmienna z przyjmuje dla testów: 0 0 0 i 0 1 0. Dla testu 0 1 1, zero w punkcie g wymusza wejście $\bar{y}$, natomiast propagacja sygnału odbywa się obydwoma ścieżkami. Test 1 0 1, wymusza 1 w punkcie g na wejściu $\bar{x}$ i jest propagowany ścieżką $g - B - D - f$, podobnie jak test 1 1 1, dla którego sygnał 1 w punkcie g wymuszają wejścia $\bar{x}$ oraz $\bar{y}$. Test 1 1 0 wprowadzie wymusza w punkcie g sygnał 1, ale jednocześnie występuje blokada wszystkich ścieżek, a więc nie jest to test dla omawianego uszkodzenia. Dotyczy to również testu 1 0 0. Ostatecznie uzyskujemy zbiory testów identyczne z tymi jakie otrzymano w przykładach 3.7 i 3.18.

W ten sposób można określić zbiory testów dla wszystkich przewidywanych uszkodzeń w układzie. Na rysunku 3.43 powtórzono schemat z rysunku 3.42 z zaznaczonymi punktami uszkodzeń. Założono, że najbardziej prawdopodobnymi są uszkodzenia na wejściach zewnętrznych układu x, y, z oraz na wyjściach bramek, przy czym uszkodzenia wejść są wywołane przerwaniem połączenia $s - 1$, natomiast na wyjściach bramek uszkodzenie polega na zwarcie do masy $s - 0$. W tabeli pokazanej na rysunku 3.44 przedstawiono pełny wykaz testów, przy czym symbolem 1 zaznaczono uszkodze-

nie wykrywane przez dany test, kreską - niewykrywane. Rodzaj uszkodzenia precyzuje indeks przy nazwie punktu uszkodzenia: 0 - dla $s = 0$, 1 - dla $s = 1$. Z tabeli tej można określić minimalny zbiór testów pozwalający stwierdzić, czy układ jest dobry, czy też uszkodzony. Są dwa takie zbiory, przy czym zawierają one tylko trzy testy: 0 0 0, 0 0 1, 1 0 0 oraz 0 0 0, 0 0 1, 1 1 0. Kryterium wyboru zbioru testów jest proste: muszą one wykrywać wszystkie przewidywane uszkodzenia, a więc ich jedynki w tabeli muszą pokrywać wszystkie kolumny uszkodzeń. Jeśli za każdym razem wynik testu jest pozytywny, tzn. wyjście ma wartość wynikającą z realizowanej funkcji, to w układzie nie wystąpiło żadne z wymienionych w tabeli uszkodzeń, a więc układ jest dobry.



Rys. 3.43. Schemat układu z przykładu 3.17 z zaznaczonymi punktami prawdopodobnych uszkodzeń

test			g_0	h_1	$\bar{l}_0$	j_0	k_0	l_1	m_1	n_1	o_0	p_0
x	y	z										
0	0	0	1	—	—	—	1	1	1	—	1	—
0	0	1	—	—	1	1	—	—	1	1	1	1
0	1	0	1	—	—	—	1	—	1	—	1	—
0	1	1	1	—	—	—	1	—	—	—	—	—
1	0	0	—	1	1	1	—	—	—	—	—	—
1	0	1	1	—	—	—	1	—	—	—	—	—
1	1	0	—	1	1	1	—	—	—	—	—	—
1	1	1	1	—	—	—	1	—	—	—	—	—

Rys. 3.44. Tablica uszkodzeń dla testów układu pokazanego na rysunku 3.43

3.10.4. Diagnostyka układów kombinacyjnych

Analiza tabeli uszkodzeń pozwala nie tylko na ocenę typu: zły, dobry, ale również umożliwia przeprowadzenie lokalizacji konkretnego uszkodzenia, ewentualnie lokalizację grupy uszkodzeń. Przykładowo, jeśli wejścia układu pokazanego na rysunku 3.43ysterować testem 0 0 0, to zła wartość funkcji wyjściowej może być wywołana uszkodzeniami: L_1 , m_1 , 0_0 , g_0 , k_0 (tabela z rys. 3.44). Jeśli teraz po teście 0 0 0ysterujemy wejścia testem 0 1 0, a uzyskany wynik na wyjściu f jest dobry, to widzimy, że wystąpiło zwarcie do zasilania na wejściu bramki A (L_1), gdyż tylko kolumna L_1 nie może być oceniona przez test 0 1 0 jako zła, a dla testu 0 0 0 występuje tutaj 1. W ten sposób można zbudować tzw. adaptacyjny test diagnostyczny, w którym podając w określonej kolejności odpowiednie testy na wejście, zależnie od rezultatu na wyjściu, przeprowadzamy lokalizację uszkodzenia. Oprócz testu adaptacyjnego można stworzyć bezwarunkowy test diagnostyczny. Uzyskuje się go również z tabeli uszkodzeń, tworząc kolumny rozróżnialnych par uszkodzeń i następnie dla każdego testu wpisujemy 1, jeśli rozróżnia on tylko jedno z nich, natomiast 0, jeśli nie wykrywa żadnego lub wykrywa obydwa jednocześnie. W oparciu o taką tabelę, minimalny zbiór testów otrzymujemy wybierając pełne pokrycie jedynekami wszystkich par tworzących kolumny, analogicznie jak przy poszukiwaniu pokrycia mintermów prostymi implikantami. Wielką wadą tabeli testu bezwarunkowego jest szybki wzrost liczby kolumn wraz ze wzrostem liczby punktów uszkodzeń. Dla n punktów, liczba par uszkodzeń wynosi: $n(n - 1)/2$. Dla $n = 10$ liczba kolumn tabeli testu bezwarunkowego wyniesie już 45. W tej sytuacji znacznie wygodniejsze jest operowanie testem adaptacyjnym. Mechanizm konstruowania testu adaptacyjnego prześledźmy na przykładzie.

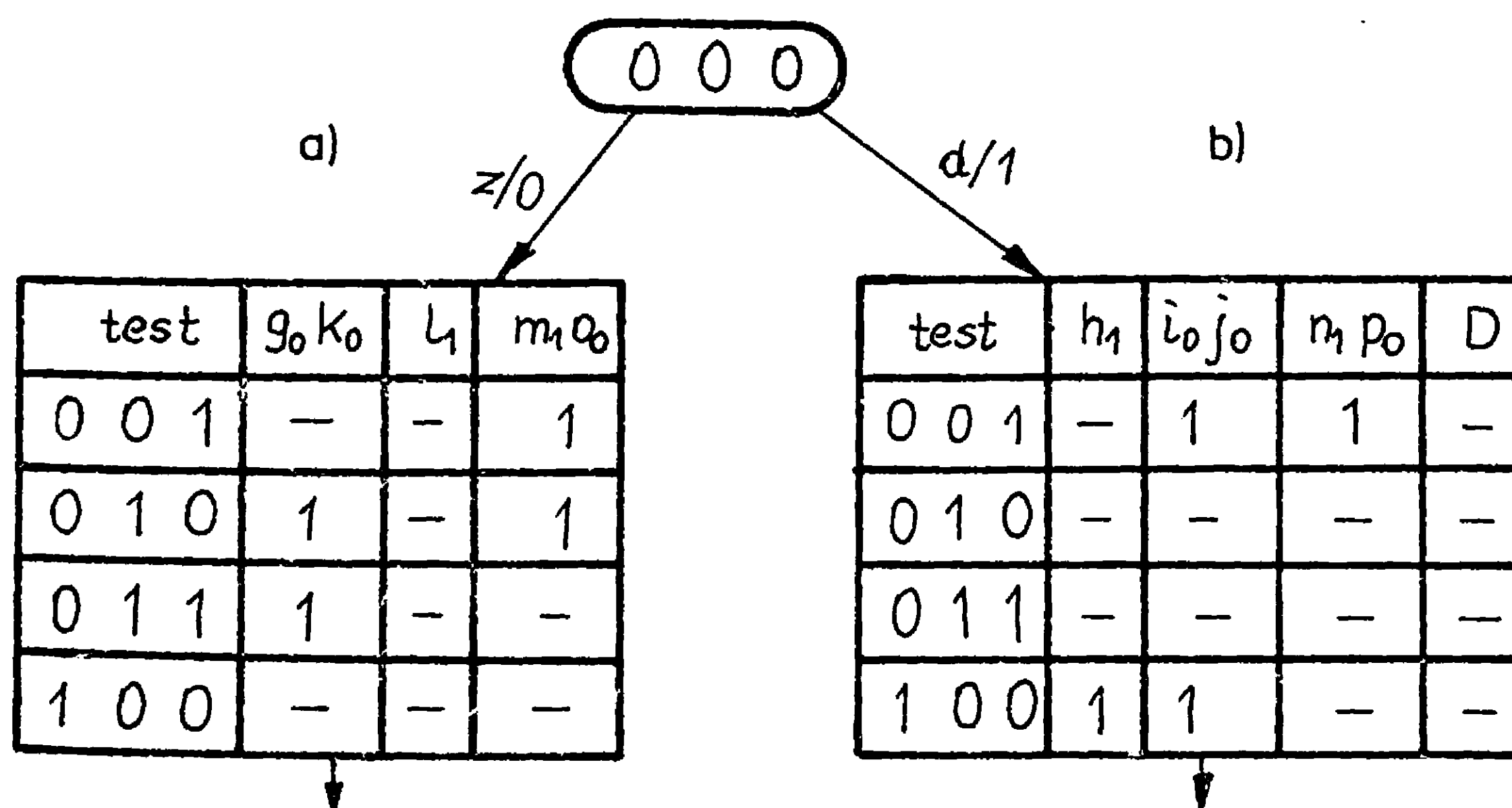
Przykład 3.20. Określić adaptacyjny test diagnostyczny dla układu przedstawionego na rysunku 3.43.

W celu utworzenia testu adaptacyjnego tabelę uszkodzeń przedstawioną na rys. 3.44, przekształćmy w ten sposób, że kolumny i wiersze, które mają taki sam rozkład jedynek połączmy razem. Otrzymaną w ten sposób, zredukowaną tablicę przedstawiono na rysunku 3.45. Do tabeli zredukowanej dołączono kolumnę D, reprezentującą brak uszkodzeń. Będzie to pomocne w procesie tworzenia drzewa diagnostycznego. Tworzymy je wychodząc z dowolnego testu wiersza, z tym że aby otrzymać możliwie symetryczne rozgałęzienia, dobrze jest dobrać taki test, który ma zbli-

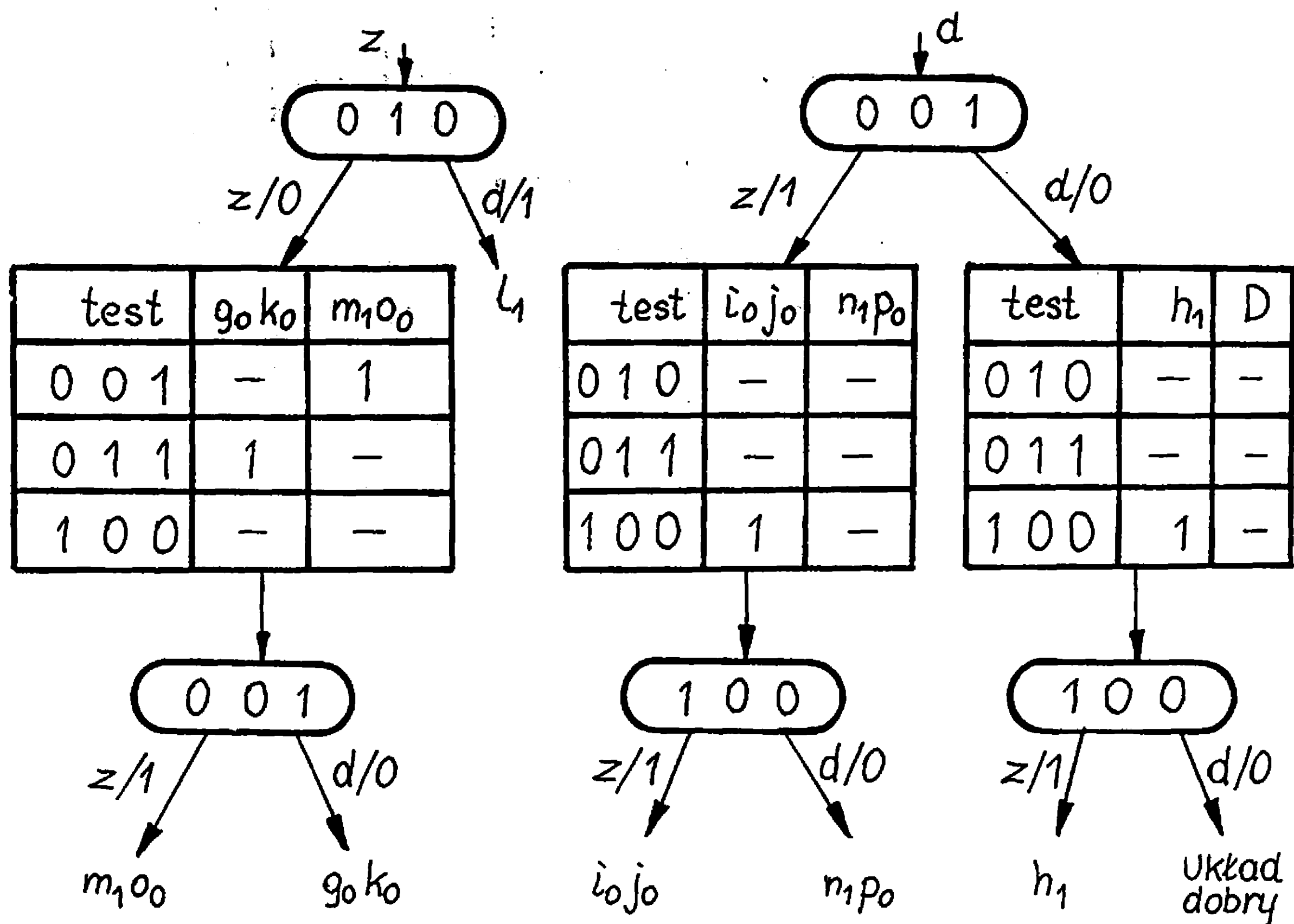
test x y z	$g_0 k_0$	h_1	$i_0 j_0$	l_1	$m_1 o_0$	$n_1 p_0$	D
0 0 0	1	-	-	1	1	-	-
0 0 1	-	-	1	-	1	1	-
0 1 0	1	-	-	-	1	-	-
0 1 1	1	-	-	-	-	-	-
1 0 0	-	1	1	-	-	-	-

Rys. 3.45. Zredukowana tablica uszkodzeń układu przedstawionego na rys. 3.43 (przykład 3.20)

zoną liczbę jedynek i kresek. Każdy test dzieli tabelę na dwie części. Do pierwszej należą te wszystkie uszkodzenia, które ten test wykrywa, do drugiej - wszystkie niewykrywalne. Utwórzmy te tabele dla testu 0 0 0, pomijając wiersz tego testu. Przedstawiono je na rysunku 3.46. Tabela a) zawiera te uszkodzenia, dla których na wyjściu układu uzyskamy wynik niewłaściwy (zły), tabela b) te uszkodzenia, dla których sygnał na wyjściu układu jest prawidłowy (dobry) oraz kolumnę D. Każdą z tak otrzymanych tabel można w sposób niezależny dzielić dalej aż do momentu, gdy uzyskamy tabele jednokolumnowe, a więc z konkretnie sprecyzowanym uszkodzeniem. Na rysunku 3.47 przedstawiono kolejny podział tabeli a) z rysunku 3.46 dla testu 0 1 0 oraz tabeli b) dla testu 0 0 1.



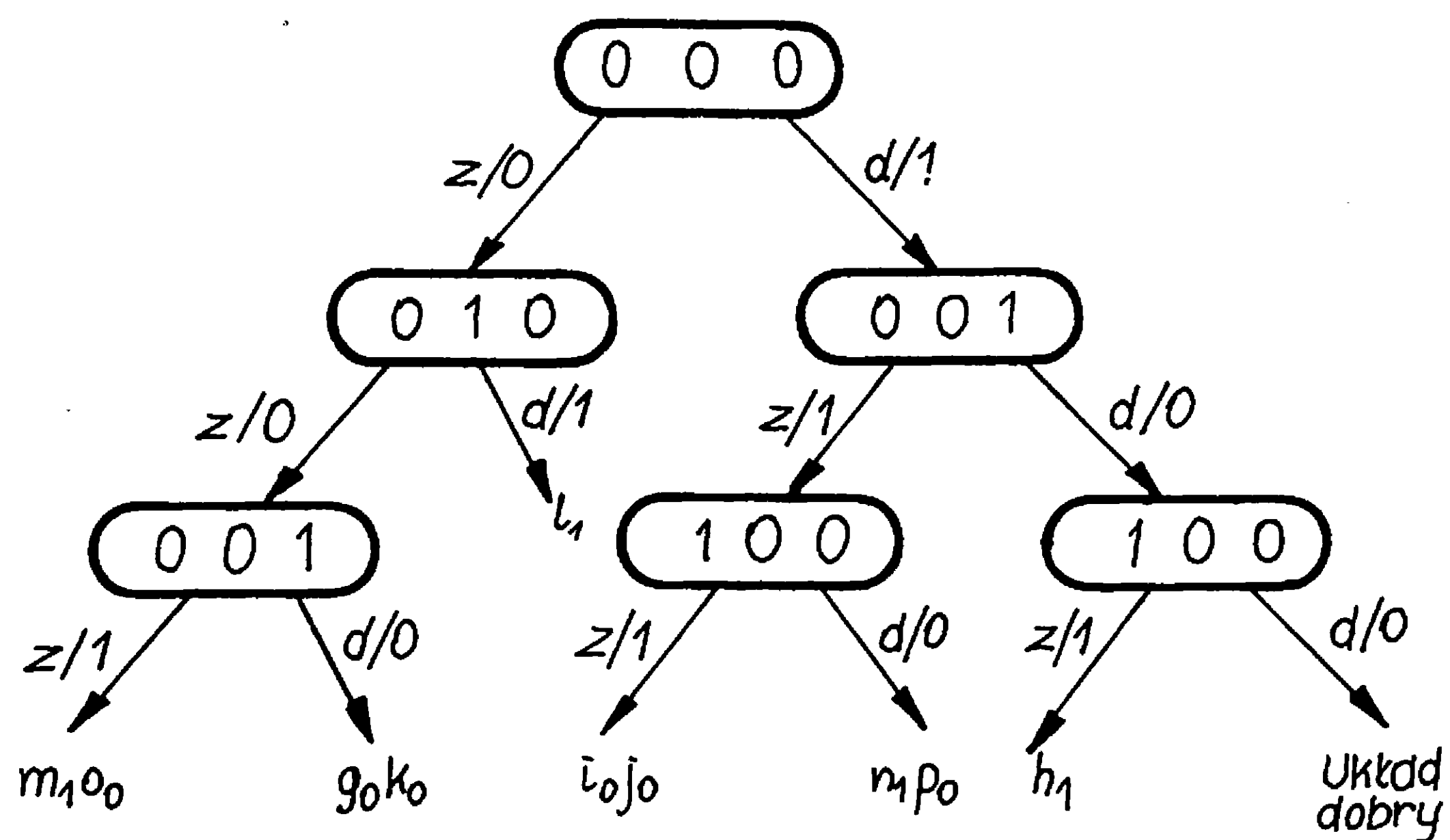
Rys. 3.46. Podział tabeli uszkodzeń względem testu 000 (przykład 3.20 (a,b))



Rys. 3.47. Schemat kolejnego podziału tabel (przykład 3.20)

Następny podział doprowadza do tabel jednokolumnowych, co zaznaczono na rysunku w formie określonego uszkodzenia. W procesie podziału tabel powstają wiersze o identycznym rozkładzie jedynek, które należy łączyć razem oraz wiersze bez jedynek, tylko z kreskami. Są one bezużyteczne w dalszym procesie podziału i można je pominąć. Fakt występowania identycznych rozkładów jedynek dla różnych testów powoduje określoną swobodę w wyborze testu. Końcowe wysterowania przedstawione na rys. 3.47 nie są więc jedynym możliwym zestawem testów drzewa diagnostycznego.

W oparciu o uzyskane rezultaty można przystąpić do tworzenia kompleksowego drzewa diagnostycznego. Wariant drzewa odpowiadający tablicom pokazanym na rysunku 3.44÷3.47, przedstawiono na rysunku 3.48. Literami przy przejściach do kolejnych testów oznaczono symbolicznie rezultat na wyjściu: z - zły, d - dobry, jednocześnie podano wartość wyjścia odpowiadającego podanemu rezultatowi. Należy podkreślić, że w oparciu o przedstawiony w przykładzie algorytm, można zbudować szereg innych drzew diagnostycznych związanych z danym układem, wszystkie jednak z punktu widzenia detekcji uszkodzeń są sobie równoważne.



Rys. 3.48. Jedno z możliwych drzew diagnostycznych układu pokazanego na rys. 3.43

4. UKŁADY LOGICZNE W UJĘCIU TEORII AUTOMATÓW

4.1. Uwagi ogólne

Wartości wektorów charakteryzujące układ logiczny są reprezentowane przez ciągi kombinacji zero-jedynkowych związanych ze stanami poszczególnych składowych tych wektorów. Stany logiczne 0 i 1 reprezentują z kolei pewne efekty fizyczne charakterystyczne dla urządzenia, którego pracę modeluje układ logiczny. Dalszym etapem zwiększającym stopień abstrakcji procesu modelowania, jest przypisanie wartościom występującym wektorów pewnych nazw, niezależnych od konkretnych reprezentacji tych wartości. Zbiór tych nazw nosi nazwę alfabetu, a wartości wektorów układu logicznego im odpowiadające są tylko szczególnym przypadkiem ich reprezentacji. Jest to jednocześnie jeden z nieskończenie wielu możliwych takich przypadków. Pojęcie alfabetu leży u podstaw definicji charakterystycznych dla teorii automatów. Teoria układów logicznych jest więc tylko wariantem szczegółowym teorii automatów, ale (i to trzeba wyraźnie podkreślić) wariantem o dużym znaczeniu użytkowym.

Badania w dziedzinie teorii automatów mają głównie charakter poznawczy, pewne jednak działy znajdują zastosowanie w procesie syntezy projektowej sekwencyjnych układów logicznych (patrz p. 1.2). Dotyczy to głównie tzw. automatów skończonych, dla których zarówno alfabet charakteryzujący wejście, jak i wyjście oraz stany wewnętrzne, zawiera ograniczoną liczbę nazw (liter). Automat pracuje w dyskretnej skali czasu, tzn. zmiany liter wejściowych, wyjściowych i stanów wewnętrznych odbywają się w ściśle określonych chwilach czasowych. Zbiór tych chwil jest zbiorem przeliczalnym, a więc można je ponumerować kolejnymi liczbami naturalnymi. W następnym punkcie podano definicje automatów mających swoje bezpośrednie odpowiedniki w konkretnych realizacjach układów logicznych.

4.2. Automaty Mealy'ego i Moore'a

Podając na wejście automatu określoną sekwencję liter wejściowych, w k-tej chwili otrzymamy określoną literę wyjściową

$$\dots, x_{k-3}, x_{k-2}, x_{k-1}, x_k \rightarrow y_k.$$

Nie jest jednak możliwe scharakteryzowanie w podany sposób pracy całego automatu, gdyż nawet dla skończonych licznosci alfabetów wejściowych, można tworzyć nieskończenie wiele sekwencji liter (słów wejściowych). Sekwencje te tworzą historie automatu, które są z kolei reprezentowane przez stany wewnętrzne zwane alfabetem wewnętrznym. Skończony alfabet wewnętrzny reprezentuje skończoną liczbę wszystkich możliwych historii danego automatu. Pojęcie tak rozumianego stanu wewnętrznego leży u podstaw większości definicji automatów, a w szczególności definicji wykorzystywanych na potrzeby układów logicznych.

Definicja 4.1. Automatem Mealy'ego nazywamy uporządkowaną "piątkę symboli"

$$A = \langle X, S, Y, \delta, \lambda \rangle.$$

gdzie: $X = \{x_0, x_1, x_2, \dots, x_{m-1}\}$ - alfabet wejściowy
 $S = \{s_0, s_1, \dots, s_{l-1}\}$ - alfabet wewnętrzny (zbiór stanów wewnętrznych),
 $Y = \{y_0, y_1, \dots, y_{n-1}\}$ - alfabet wyjściowy,
 δ - funkcja przejść,
 λ - funkcja wyjść.

Funkcja przejść δ przyporządkowuje każdej parze $\langle x_k, s_k \rangle$ stan następny s_{k+1}

$$\delta : \langle x_k, s_k \rangle \rightarrow s_{k+1}.$$

Zakładamy przy tym, że pary te tworzą podzbiór iloczynu kartezjańskiego $X \times S$ oznaczony przez D_δ

$$\langle x_k, s_k \rangle \in D_\delta \subset X \times S.$$

Analogicznie, funkcja wyjść λ przyporządkowuje każdej parze $\langle x_k, s_k \rangle$ literę wyjściową y_k

$$\lambda : \langle x_k, s_k \rangle \rightarrow y_k.$$

Zakładamy przy tym, że pary te tworzą podzbiór iloczynu kartezjańskiego $X \times S$ oznaczony przez D_λ

$$\langle x_k, s_k \rangle \in D_\lambda \subset X \times S.$$

Innymi słowy, aby zdefiniować automaty Mealy'ego należy:

- podać alfabet wejściowy,
- podać alfabet wewnętrzny,
- podać alfabet wyjściowy,
- określić funkcję przejść δ ,
- określić funkcję wyjść λ .

Przykładowo, jeśli przyjmiemy:

$$\begin{aligned} X &= \{a, b, c\}, \\ S &= \{q, r, s\}, \\ Y &= \{y, z\}, \end{aligned}$$

to iloczyn kartezjański $X \times S$, wyniesie

$$X \times S = \{aq, ar, as, bq, br, bs, cq, cr, cs\}.$$

Przyjmując:

$$D_\delta = \{aq, ar, bq, bs, cq, cr\} \subset X \times S,$$

$$D_\lambda = \{ar, as, br, bs, cq\} \subset X \times S$$

oraz definiując funkcję przejść:

$$\begin{aligned} \delta : \quad & \langle a, q \rangle \rightarrow r, \\ & \langle a, r \rangle \rightarrow r, \\ & \langle b, q \rangle \rightarrow q, \\ & \langle b, s \rangle \rightarrow r, \\ & \langle c, q \rangle \rightarrow s, \\ & \langle c, r \rangle \rightarrow s \end{aligned}$$

i funkcję wyjść

$$\lambda : \begin{aligned} &\langle a, r \rangle \rightarrow z, \\ &\langle a, s \rangle \rightarrow y, \\ &\langle b, r \rangle \rightarrow y, \\ &\langle b, s \rangle \rightarrow z, \\ &\langle c, q \rangle \rightarrow z, \end{aligned}$$

zdefiniowaliśmy automat Mealy'ego.

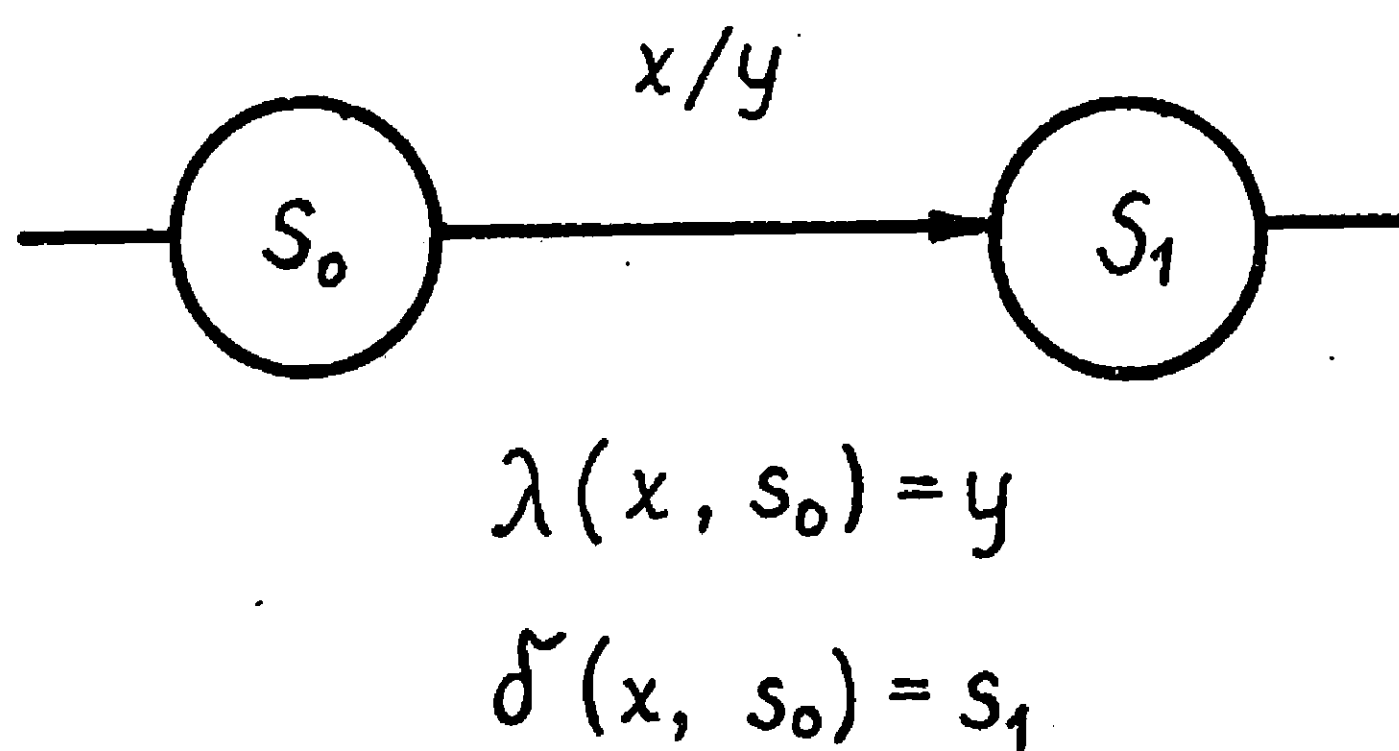
	a	b	c		a	b	c
q	r	q	s		-	-	z
r	r	-	s		z	y	-
s	-	r	-		y	z	-

Rys. 4.1. Przykładowa tablica przejść - wyjść automatu Mealy'ego

Przedstawiona forma opisu automatu jest niewygodna, dlatego częściej stosuje się zapis definicyjny w postaci tzw. tablic przejść - wyjść lub też w postaci grafu. Tablice przejść - wyjść zawierają dwa zestawy kolumn, w których dla każdej pary liter wejściowych i liter alfabetu wewnętrznego jest przypisana litera alfabetu wewnętrznego (tabela przejść) lub litera alfabetu wyjściowego (tabela wyjść). Dla omawianego przykładu tabela przejść - wyjść przyjmie postać przedstawioną na rysunku 4.1. Pary nie należące do zbiorów D_δ nie określają stanu następnego, w związku z czym w tabeli przejść zaznaczono to kreskami. Analogicznie, dla par nie należących do zbioru D_λ litery wyjściowe są nieokreślone. Zaznaczono to również kreskami w tabeli wyjść. Automat zawierający kreski w tablicy przejść - wyjść jest często nazywany automatem niezupełnym. Kresek nie byłoby wówczas, gdyby zostały zdefiniowane stany następne i litery wyjściowe dla wszystkich par iloczynu kartezjańskiego $X \times S$, tzn. gdyby

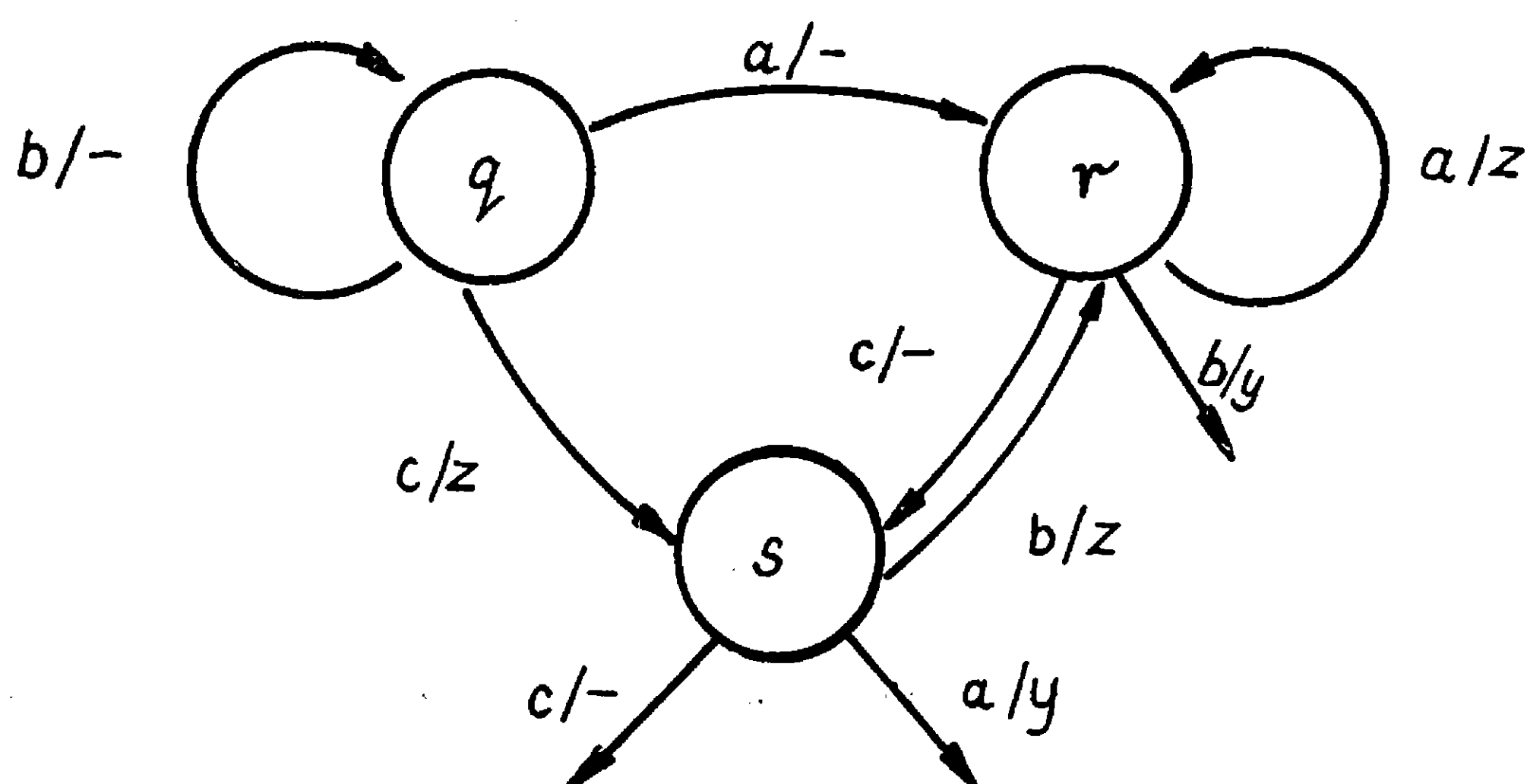
$$D_\delta = D_\lambda = X \times S. \quad (1)$$

Automat Mealy'ego, dla którego zachodzi zależność (1) jest nazywany automatem zupełnym. Topologiczną formą opisu automatu jest jego graf. Na rysunku 4.2 przedstawiono konwencję zapisu funkcji przejść i funkcję wyjść dla automatu Mealy'ego. Litera x oznacza literę wejściową, która w parze ze stanem s_0 powoduje przejście do stanu s_1 oraz generuje literę wyjściową y . Graf automatu opisanego tabelą pokazaną na rysunku 4.1 przedstawiono na rysunku 4.3. Zawiera on te same treści co i tabela, opis jest jednak tutaj bardziej obrazowy. Jeśli stanem następnym przy pobudzeniu określoną literą jest ten sam stan, mamy do czynienia z pętlą własną grafu. Dla omawianego automatu pętla własna powstanie przy pobudzeniu literą wejściową b w stanie q oraz a w stanie r . Pobudzenie literą b w stanie r oraz a i c w stanie s powoduje wyjście poza graf (odpowiedniki kresek w tabeli przejść). Wyjście takie może jednak łączyć się z wygenerowaniem litery wyjściowej. Ma to miejsce dla stanu r i pobudzenia literą b oraz dla stanu s i pobudzenia literą a . Łatwo sprawdzić, że w przypadku automatu zupełnego, wyjście poza graf nie jest możliwe.



Rys. 4.2. Graficzna konwencja zapisu przejść i wyjść automatu Mealy'ego

Drugim typem automatu mającym powszechne zastosowanie w układach logicznych jest automat typu Moore'a. Definicja tego automatu niewiele różni się od definicji poprzedniej.



Rys. 4.3. Graf automatu opisanego tablicą pokazaną na rys. 4.1

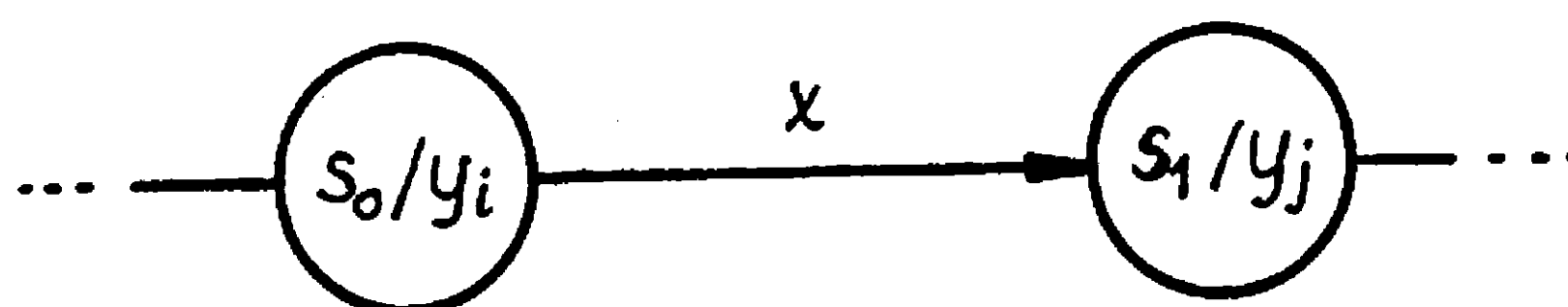
Definicja 4.2. Automatem Moore'a nazywamy uporządkowaną "piątkę symboli"

$$A = \langle X, S, Y, \delta, \lambda \rangle,$$

w której X, S, Y, δ , są zdefiniowane identycznie jak w definicji 4.1, natomiast funkcję wyjściową λ określa się wyłącznie dla zbioru stanów wewnętrznych, tzn:

$$\lambda : s_k \rightarrow y_k, \quad \text{gdzie: } s_k \in S.$$

Litery wyjściowe automatu Moore'a są więc przypisane wyłącznie stanom wewnętrznym. Tablica wyjść automatu zawiera tylko jedną kolumnę, natomiast konwencję tworzenia grafu przedstawiono na rysunku 4.4. Różnica między konwencją dla automatu Mealy'ego polega na przypisaniu liter wyjściowych bezpośrednio stanom, poprzez umieszczenie ich pod kreską symbolu stanu. Na rysunku 4.5 przedstawiono tablicę przejść - wyjść oraz graf automatu typu Moore'a. Jest to automat zupełny. Stan q automatu posiada aż dwie pętle własne: dla liter a oraz c .

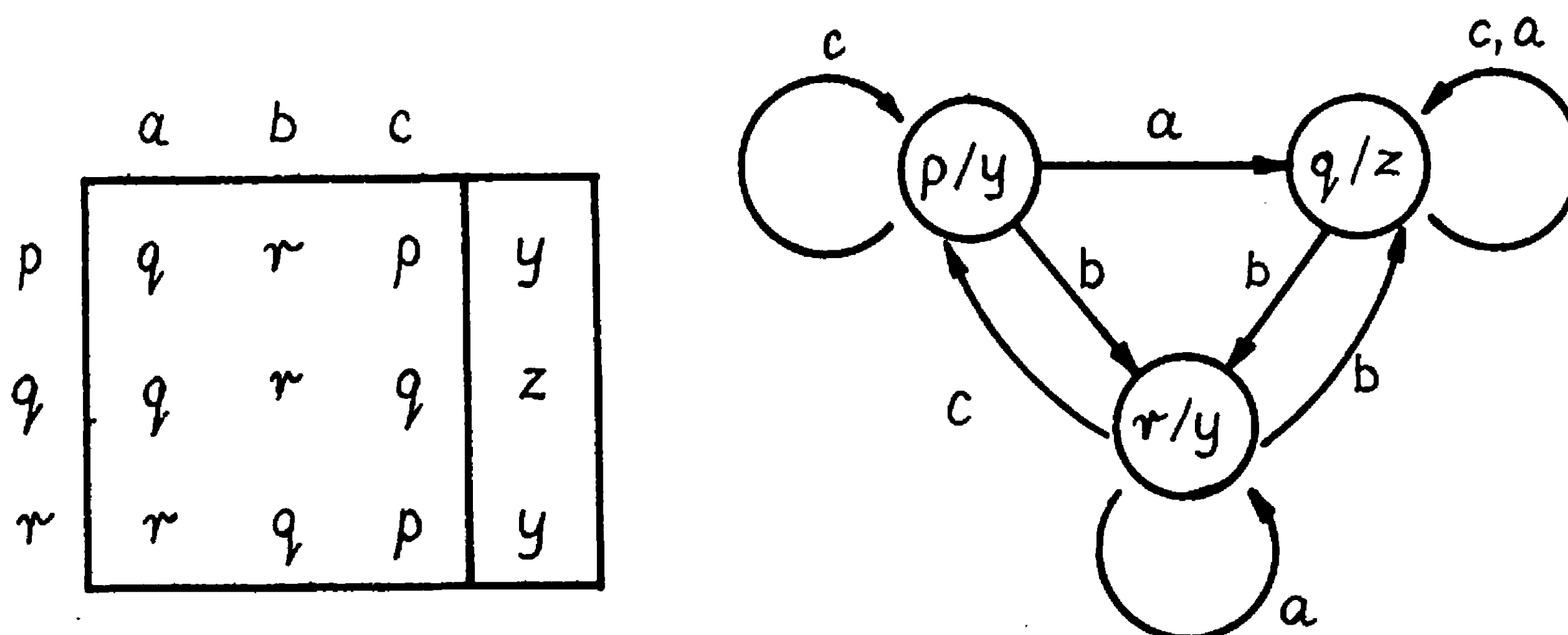


$$\lambda(s_0) = y_i$$

$$\lambda(s_1) = y_j$$

$$\delta(x, s_0) = s_1$$

Rys. 4.4. Graficzna konwencja zapisu przejść i wyjść automatu Moore'a



Rys. 4.5. Przykładowa tablica przejść - wyjść i graf automatu Moore'a

4.3. Podstawowe definicje

Duża liczba zadań realizowanych przez układy logiczne jest sformułowana w kategoriach wejście - wyjście, bez narzucania liczby stanów wewnętrznych. Pozwala to na swobodę w doborze liczby stanów, jednak względy konstrukcyjne najczęściej wymagają by była ona jak najmniejsza. Teoria automatów dostarcza narzędzia umożliwiającego osiągnięcie tego celu; dla operowania nim konieczne jest jednak wprowadzenie szeregu definicji, precyzujących reguły pobudzania oraz umożli-

wiających dokonywanie porównywania automatów między sobą. Poniżej podane definicje mają formę dostosowaną do wymagań realizacji postawionego zadania.

Definicja 4.3. Słowem wejściowym automatu A nazywamy dowolną sekwencję liter alfabetu wejściowego. Słowo oznaczamy zwykle symbolem $\tilde{x}$. Jeśli w wyniku pobudzenia automatu będącego w stanie s słowem $\tilde{x}$, końcowa litera wyjściowa jest określona, to słowo to nazywamy odpowiednimi dla stanu s , co zapisujemy symbolem $\tilde{x} \parallel s$.

Brak określoności końcowej litery wyjściowej charakteryzuje słowo nieodpowiednie. Dla automatu Mealy'ego pokazanego na rysunku 4.3, słowami nieodpowiednimi dla stanu q są np. słowa jednoliterowe a , b oraz wieloliterowe ac , cbc , bba , itd. Słowem odpowiednim jest natomiast słowo acb wygenerowujące końcową literę wyjściową z . Można więc napisać: $(acb) \parallel q$. Dla stanu r słowem nieodpowiednim jest słowo cc , ale słowo aca jest odpowiednie, mimo iż następny stan jest nieokreślony, gdyż została wygenerowana określona litera końcowa. Na marginesie ostatniego przypadku wyłania się różnica między automatem Moore'a i Mealy'ego. Dla automatu Moore'a z uwagi na bezpośrednie powiązanie między stanami wewnętrznymi i literami wyjściowymi, nieokreśloność stanu pociąga za sobą nieokreśloność końcowej litery wyjściowej. Dla automatu Mealy'ego tak nie jest, w związku z czym zachodzi niekiedy potrzeba wyodrębnienia ze słowa pobudzającego ostatniej litery wejściowej, po podaniu której określona litera wyjściowa zostanie wygenerowana, natomiast stan końcowy może być nieokreślony.

Dalszych kilka definicji zostało wprowadzonych po to, by można było dokonywać porównania dwóch różnych automatów, pobudzanych tymi samymi słowami. Dotyczą one zarówno automatu Moore'a, jak i Mealy'ego, w związku z czym nie będzie to dokładnie zaznaczane.

Definicja 4.4. Mówimy, że automat $A_1 = \langle X, S_1, Y, \delta_1, \lambda_1 \rangle$ w stanie s_1 pokrywa automat $A_2 = \langle X, S_2, Y, \delta_2, \lambda_2 \rangle$ w stanie s_2 , jeśli każde słowo odpowiednie dla stanu s_2 jest odpowiednie dla stanu s_1 i dla każdego takiego słowa końcowe litery wyjściowe obu automatów są sobie równe

$$\lambda(\tilde{x}, s_2) = \lambda(\tilde{x}, s_1).$$

Zapisujemy te symbole $s_1 \supseteq s_2$.

Należy zwrócić uwagę na fakt, że słowa odpowiednie dla stanu s_1 automatu A_1 mogą nie być odpowiednimi dla stanu s_2 automatu A_2 , natomiast wszystkie słowa odpowiednie dla s_2 są odpowiednie dla s_1 , a wygenerowane wówczas przez oba automaty końcowe litery wyjściowe są sobie równe. Oba automaty zostały określone dla tego samego alfabetu wejściowego i wyjściowego.

Definicja 4.5. Mówimy, że automat $A_1 = \langle X, S_1, Y, \delta_1, \lambda_1 \rangle$, pokrywa automat $A_2 = \langle X, S_2, Y, \delta_2, \lambda_2 \rangle$, co zapisujemy $A_1 \geq A_2$, jeśli dla każdego stanu s_2 automatu A_2 istnieje przynajmniej jeden stan s_1 automatu A_1 taki, że $s_1 \geq s_2$. Można to zapisać następująco

$$A_1 \geq A_2 \iff \bigwedge_{s_2 \in S_2} \bigvee_{s_1 \in S_1} s_1 \geq s_2.$$

Automat A_1 niejako zawiera w sobie funkcję automatu A_2 , jest więc od niego operatywniejszy. Przypadkiem granicznym właściwości pokrywania automatów, jest właściwość ich nieodróżnialności [2]. W odniesieniu do konkretnych stanów, nieodróżnialność automatów można zdefiniować następująco:

Definicja 4.6. Automat $A_1 = \langle X, S_1, Y, \delta_1, \lambda_1 \rangle$ w stanie s_1 jest nieodróżnialny od automatu $A_2 = \langle X, S_2, Y, \delta_2, \lambda_2 \rangle$ w stanie s_2 , jeśli każde słowo odpowiednie dla s_2 jest także odpowiednie dla s_1 i odwrotnie, przy czym końcowe litery wyjściowe są sobie równe. Zapisujemy to symbolem $s_1 \equiv s_2$.

Właściwość nieodróżnialności dla konkretnych stanów tym się różni od właściwości pokrywania, że jest symetryczna. Można ją rozszerzyć na automaty będące w dowolnych stanach wyjściowych.

Definicja 4.7. Automat $A_1 = \langle X, S_1, Y, \delta_1, \lambda_1 \rangle$ jest nieodróżnialny od automatu $A_2 = \langle X, S_2, Y, \delta_2, \lambda_2 \rangle$, jeśli automaty te wzajemnie się pokrywają. Zapisujemy to symbolem $A_1 \equiv A_2$.

Definicje pokrywania i nierozróżnialności automatów są bardzo użyteczne w procesie poszukiwania automatu, który realizując wszystkie funkcje wejścia - wyjścia automatu pierwotnego, posiada mniejszą liczbę stanów wewnętrznych. Prowadzi to do definicji automatu zredukowanego.

Definicja 4.7. Automatem zredukowanym $A_r = \langle X, S_r, Y, \delta_r, \lambda_r \rangle$ dla automatu $A = \langle X, S, Y, \delta, \lambda \rangle$ nazywamy taki automat, który go pokrywa, tzn. $A_r \geq A$ i jednocześnie liczba stanów automatu zredukowanego $|S_r|$ nie jest większa od liczby stanów automatu A $|S| : |S_r| \leq |S|$.

Dany automat może mieć wiele automatów zredukowanych, może ich też nie mieć wcale. Wtedy sam dla siebie jest automatem zredukowanym. W zbiorze automatów zredukowanych znajduje się przynajmniej jeden o najmniejszej możliwej liczbie stanów wewnętrznych. Znalezienie go jest celem zasadniczym naszych poszukiwań. Aby to osiągnąć konieczne jest wyszukanie w automacie pierwotnym wszystkich stanów niesprzecznych, wzajemne ich pogrupowanie i zastąpienie jednym stanem automatu zredukowanego. Temu celowi służą następujące definicje.

Definicja 4.8. Relacją niesprzeczności stanów automatu $A = \langle X, S, Y, \delta, \lambda \rangle$ nazywamy zbiór wszystkich takich par jego stanów $\langle s_1, s_2 \rangle$, dla których słowo odpowiednie dla jednego stanu, jeśli jest również odpowiednie dla drugiego, to wygenerowane z tych stanów końcowe litery wyjściowe są sobie równe, tzn. $\lambda(\tilde{x}, s_1) = \lambda(\tilde{x}, s_2)$. Można to zapisać następująco:

$$R_{\sim} = \{ \langle s_1, s_2 \rangle \in S \mid \tilde{x} \parallel s_1 \cap \tilde{x} \parallel s_2 \rightarrow \lambda(\tilde{x}, s_1) = \lambda(\tilde{x}, s_2) \}.$$

Stany niesprzeczne oznaczamy symbolem: $s_1 \approx s_2$.

Stanami niesprzecznymi są więc takie stany, dla których słowo odpowiednie dla jednego stanu powoduje przy pobudzeniu nim automatu znajdującego się w drugim stanie, albo wygenerowanie końcowej litery wyjściowej takiej samej jak dla stanu pierwszego, albo litera ta jest nieokreślona. Innymi słowy, dla stanów niesprzecznych końcowe litery wyjściowe, jeśli są obie określone, to muszą być równe. Stany nie spełniające tego warunku są stanami sprzecznymi. Można dla nich zdefiniować w sposób dualny relację sprzeczności oznaczaną przez $R_{\not\sim}$. Relacją sprzeczności nazywamy zbiór par stanów $\langle s_1, s_2 \rangle$ takich, że jeśli słowo $\tilde{x}$ jest odpowiednie dla s_1 oraz dla s_2 , to wygenerowane końcowe litery muszą być różne, tzn. $\lambda(\tilde{x}, s_1) \neq \lambda(\tilde{x}, s_2)$. Relacja ta ma mniejsze znaczenie i częściej operuje się relacją niesprzeczności.

Relacja niesprzeczności odgrywa istotną rolę w procesie poszukiwań automatu minimalnego, bowiem w oparciu o nią można połączyć w zbiory wszystkie stany tworzące pary niesprzeczne i zastąpić jednym, re-

prezentującym je stanem zastępczym. W związku z tym wprowadza się definicję zbioru stanów niesprzecznych, maksymalnego zbioru stanów niesprzecznych i rodziny tych zbiorów.

Definicja 4.9. Zbiorem stanów niesprzecznych Q nazywamy taki zbiór, którego dowolne dwa elementy tworzą parę stanów niesprzecznych. Maksymalnym zbiorem stanów niesprzecznych $Q_{\max}$, nazywamy taki zbiór Q , do którego dodanie jednego elementu powoduje utratę właściwości niesprzeczności stanów.

Definicja 4.10. Zbiór wszystkich dla danego elementu zbiorów maksymalnych stanów niesprzecznych nazywamy rodziną maksymalnych stanów zbiorów niesprzecznych i oznaczamy symbolem $\{Q_{\max}\}$.

Pojęcie rodziny maksymalnych zbiorów stanów niesprzecznych odgrywa kluczową rolę w procesie poszukiwania automatu minimalnego. Wyznacza się ją najczęściej bezpośrednio z relacji niesprzeczności.

W analogiczny sposób jak w definicji relacji niesprzeczności, można zdefiniować relację nieodróżnialności stanów $R \equiv$.

Definicja 4.11. Relacją nieodróżnialności stanów automatu $A = \langle X, S, Y, \delta, \lambda \rangle$ nazywamy zbiór wszystkich par $\langle s_1, s_2 \rangle$ stanów nieodróżnialnych, tzn. takich, dla których słowo odpowiednie dla jednego z nich jest jednocześnie odpowiednim dla drugiego i odwrotnie, a wygenerowane z tych słów końcowe litery wyjściowe są sobie równe $\lambda(\tilde{x}, s_1) = \lambda(\tilde{x}, s_2)$.

Innymi słowy, para stanów nieodróżnialnych różni się co najwyżej tylko oznaczeniem, a z punktu widzenia wyjścia stany te są nieodróżnialne. Dla stanów tych można również zdefiniować zbiór stanów nieodróżnialnych i maksymalny zbiór stanów nieodróżnialnych.

Definicja 4.12. Zbiorem stanów nieodróżnialnych B nazywamy taki zbiór, którego dowolne dwa elementy tworzą parę stanów nieodróżnialnych. Maksymalnym zbiorem stanów nieodróżnialnych $B_{\max}$ nazywamy taki zbiór B , do którego dodanie jednego elementu powoduje utratę właściwości nieodróżnialnych stanów.

Definicja 4.13. Zbiór wszystkich dla danego automatu zbiorów maksymalnych stanów nieodróżnialnych, nazywamy rodziną maksymalnych zbiorów stanów nieodróżnialnych i oznaczamy symbolem $\{B_{\max}\}$.

Pojęcie rodziny maksymalnych stanów nieodróżnialnych odgrywa dość ważną rolę w procesie poszukiwania zredukowanego automatu minimalnego wówczas, gdy automat pierwotny posiada dużą liczbę stanów. Tworząc rodzinę maksymalnych zbiorów stanów nieodróżnialnych należy pamiętać o tym, że każdy stan jest sam z sobą nieodróżnialny. Musi to znaleźć odbicie w rezultacie końcowym, tzn. w rodzinie muszą się znaleźć jako zbiory jednoelementowe te wszystkie stany, które nie tworzą stanów nieodróżnialnych z innymi stanami automatu. Ta sama uwaga dotyczy maksymalnej rodziny stanów niesprzecznych, gdyż szczególnym przypadkiem niesprzeczności jest nieodróżnialność.

4.4. Określanie automatu minimalnego

4.4.1. Algorytmy wyznaczania relacji i zbiorów stanów niesprzecznych i nieodróżnialnych

Jak już powiedziano, relacja niesprzeczności jest potrzebna do określania rodziny maksymalnych zbiorów stanów niesprzecznych, odgrywa więc istotną rolę w procesie poszukiwania automatu minimalnego. Praktycznie nie jest możliwe znalezienie tej relacji metodą prób i błędów, gdyż każdy stan najczęściej posiada nieskończenie wiele słów odpowiednich. Poszukiwanie jednak można prowadzić w sposób systematyczny poruszając się od pobudzania automatu słowem jednoliterowym. Takich słów jest tylko tyle, ile zawiera liter alfabet wejściowy. W wyniku otrzymamy dwa zbiory par stanów: jeden par sprzecznych, drugi - par warunkowo niesprzecznych, przy założeniu pobudzenia jedną literą. Zbiór par stanów sprzecznych, a więc takich, dla których gdy obie litery końcowe są określone to są różne, dalej nas nie interesuje. Oznaczmy pierwszy zbiór par warunkowo niesprzecznych przez R_0 . Dokonajmy teraz przeglądu tego zbioru, określając dla wszystkich liter alfabetu wejściowego stany następne otrzymane dla każdej pary stanów warunkowo niesprzecznych. Rezultat może być następujący:

a) stany następne są albo obydwa nieokreślone albo tylko nieokreślony jest jeden z nich,

- b) stany następne są oba takie same, tzn. $s_1 = s_2$,
- c) stany następne tworzą parę należącą do zbioru R_0 ,
- d) stany następne należą do zbioru stanów sprzecznych, a więc nie należą do R_0 .

Przypadek d) określa te stany, które wprowadzicie pobudzone jedną literą nie są sprzeczne, ale dla dwuliterowych pobudzeń wygenerowują różne końcowe litery wyjściowe. Pary te muszą więc opuścić zbiór R_0 i przejść do zbioru par sprzecznych. Otrzymujemy w ten sposób że z zbioru R_0 zbiór R_1 par warunkowo niesprzecznych. Kontynuując powyższą procedurę dojdziemy do momentu, gdy $R_{j-1} = R_j$ ($j = 1, 2, \dots$). Zbiór R_j jest częścią zasadniczą relacji niesprzeczności, która jako całość obok par zbioru R_j zawiera zbiór R'_j utworzony z R_j przez zamianę miejscami stanów w poszczególnych parach oraz część własną utworzoną ze zbioru par, jakie wszystkie stany tworzą same ze sobą. Omówiony schemat postępowania zilustrujemy na przykładzie.

Przykład 4.1. Znaleźć relację niesprzeczności stanów automatu opisanego tabelą przejść - wyjść przedstawioną na rysunku 4.6.

	a	b	c	a	b	c
1	2	-	-	z	-	-
2	4	3	6	-	z	y
3	7	-	8	z	y	-
4	2	3	-	-	z	-
5	5	-	6	-	-	y
6	-	8	-	z	-	-
7	1	8	-	-	z	-
8	-	7	1	z	y	y

Rys. 4.6. Tablica przejść - wyjść automatu z przykładu 4.1

2	24						
3	27	⊗					
4	V	V	⊗				
5	25	45	⊗ 57 68	25			
6	V	38	V	V	V		
7	12	14 38	⊗	12 38	15	V	
8	V	⊗	18	⊗	16	⊗ 78	⊗
	1	2	3	4	5	6	7

Rys. 4.7. Tablica trójkątna Unger-Paula dla określenia relacji niesprzeczności (przykład 4.1)

Nazwami stanów omawianego automatu są liczby naturalne od 1 do 8. Zbiór par R_0 tworzymy dla pobudzenia jednoliterowego. Należą do niego pary

$$R_0 = \{1,2, 1,3, 1,4, 1,5, 1,6, 1,7, 1,8, 2,4, 2,5, \\ 2,6, 2,7, 3,5, 3,6, 3,8, 4,5, 4,6, 4,7, \\ 5,6, 5,7, 5,8, 6,7, 6,8\}.$$

W celu uniknięcia pomyłki liczbę mniejszą umieszczono na pierwszym miejscu. Tego typu zasady porządkowej należy zawsze przestrzegać! Pozostałe pary są sprzeczne; zbiór ich jest następujący:

$$\{2,3, 2,8, 3,4, 3,7, 4,8, 7,8\}.$$

Zbiór R_1 otrzymamy z R_0 poprzez odrzucenie tych par, dla których stany następne należą do wymienionego zbioru stanów sprzecznych, a więc nie należą do R_0 :

$$R_1 = \{1,2, 1,3, 1,4, 1,5, 1,6, 1,7, 1,8, 2,4, 2,5, 2,6, \\ 2,7, 3,5, 3,6, 3,8, 4,5, 4,6, 4,7, 5,6, 5,7, 5,8, 6,7\}.$$

Ze zbioru R_0 odpadła tylko para 6,8, gdyż po pobudzeniu ją literą b mamy przejście do pary 7,8, która jest sprzeczna. Przechodzimy teraz do utworzenia zbioru R_2 . Zbiór R_2 posiada tylko jedną parę mniej od R_1 ; jest nią para 3,5, która po pobudzeniu literą c prowadzi do stanów 6,8, które nie należą do R_1 , a więc są sprzeczne. Łatwo sprawdzić, że zbiory dalej się nie redukują, tzn. $R_2 = R_3$. Relacja niesprzeczności będzie więc mieć postać:

$$R = \{1,2, 1,3, 1,4, 1,5, 1,6, 1,7, 1,8, 2,4, 2,5, \\ 2,6, 2,7, 3,6, 3,8, 4,5, 4,6, 4,7, 5,6, 5,7, \\ 5,8, 6,7\} \cup \{2,1, 3,1, \dots, 7,6\} \cup \{1,1, \dots, 8,8\}.$$

Przedstawiona procedura poszukiwania relacji niesprzeczności jest dość kłopotliwa. Znacznie szybciej można uzyskać rezultat, posługując się metodą zaproponowaną przez Ungera i Paulla. W tablicę trójkątną wiążącą pary stanów (rys. 4.7) wpisujemy w pierwszym etapie symbole, np. X dla par sprzecznych oraz stany następne w pozostałych kratkach.

Kratki nie oznaczone symbolem X należą do zbioru R_0 . Następne etapy selekcji są odpowiednikami poszukiwań zbiorów R_1 , R_2 itd. Dokonujemy tego przeglądając pary stanów następnych pod kątem sprawdzenia, czy nie są one sprzeczne. Parę sprzeczną, dla której dokonujemy przeglądu oznaczamy kółkiem (łącznie w kratce będzie symbol $\otimes$). Jeśli natrafimy na parę następnych stanów sprzecznych stawiamy w kratce znak X (przekreślamy ją). Procedurę kontynuujemy tak długo, aż wszystkie kratki stanów sprzecznych zostaną ujęte w kółka. Na rysunku 4.7 przedstawiono tabelę dla automatu z przykładu 4.1. Dla ułatwienia, kratki których jeden lub oba stany następne są nieokreślone lub oba stany są jednakowe, czy też stany następne są takie same jak współrzędne kratki, oznaczono symbolem V, wiadomo bowiem od razu, że należą one do relacji niesprzeczności. Wszystkie kratki nie zawierające znaku X (nieprzekreślone) należą do relacji niesprzeczności.

W analogiczny sposób można określić algorytm poszukiwania relacji nieodróżnialności stanów (def. 4.11).

A. Tworzymy zbiór R_0 par nieodróżnialnych, tzn. takich które mają identyczne wiersze (litery wyjściowe) w tablicy wyjść. Kreski możemy potraktować jako symbole dodatkowej litery alfabetu wyjściowego.

B. Dla par zbioru R_0 poszukujemy stanów następnych i jeśli nie należą one do R_0 , parę tę odrzucamy; nieodróżnialnymi mogą więc być tylko te pary, których stany następne dla każdej litery wejściowej są wszystkie określone i należą do R_0 , albo też nieokreśloności występują na tych samych miejscach tabeli przejść. Utożsamić to można z umownym potraktowaniem "kreszek" jako dodatkowego stanu wewnętrznego rozpatrywanego automatu, oznaczonego np. symbolem s. A więc aby rozpatrywana para należała do R_1 albo para stanów następnych musi należeć do R_0 , albo jest typu: $\langle s, s \rangle$ ($\langle \text{kreska}, \text{kreska} \rangle$).

C. Tworzymy zbiór R_2 z elementów zbioru R_1 w sposób analogiczny jak w punkcie B.

D. Jeśli $R_{j-1} = R_j$ ($j = 1, 2, \dots$), to R_j oraz R_{j-1} są częścią główną relacji nieodróżnialności $R_{\equiv}$.

Wszystkie wymienione etapy algorytmu mogą być realizowane w ramach tablicy trójkątnej, analogicznie jak w przypadku relacji niesprzeczności.

Przykład 4.2. Określić relację nieodróżnialności i niesprzeczności stanów automatu, którego tablicę przejść-wyjść przedstawiono na rysunku 4.8.

Z tabeli i przedstawionego algorytmu wynika, że

$$R_0 = \{1,3, 4,6\}.$$

Ze zbioru R_0 otrzymamy zbiór R_1

$$R_1 = \{1,3, 4,6\},$$

bowiem stanami następnymi dla pary 1,3 są pary 4,6 i 3,3, natomiast dla pary 4,6, pary 1,3, i 2,2. Ponieważ $R_1 = R_2$, więc relacja nieodróżnialności ma postać

$$R = \{1,3, 4,6\} \cup \{3,1, 6,4\} \cup \{1,1, \dots, 6,6\}.$$

Ten sam rezultat osiągnęlibyśmy posługując się tablicą trójkątną (rys. 4.9). Sposób ten jest jednak tutaj bardziej złożony. Należy zwrócić uwagę na fakt, że poszukiwanie relacji $R_{\equiv}$ jest znacznie prostsze niż relacji niesprzeczności $R_{\sim}$. Relację $R_{\sim}$ wyznaczmy w oparciu o tabelę trójkątną pokazaną na rysunku 4.10. W wyniku eliminacji otrzymamy

$$R_{\sim} = \{1,3, 2,4, 2,5, 2,6, 4,6\} \dots$$

W oparciu o otrzymany rezultat możemy spróbować określić rodzinę maksymalnych zbiorów niesprzecznych. Z uwagi na niezbyt dużą liczbę elementów zbioru $R_{\sim}$, wystarczy zrobić to metodą prób i błędów. Otrzymamy następujący, łatwy do sprawdzenia rezultat

$$\{Q_{\max}\} = \left\{ \{2,4,6\}, \{2,5\}, \{1,3\} \right\}.$$

Jeszcze prościej określa się rodzinę maksymalnych zbiorów stanów nieodróżnialnych

$$\{B_{\max}\} = \left\{ \{1,3\}, \{4,6\}, \{2\}, \{5\} \right\}.$$

	a	b	a	b
1	4	3	z	z
2	5	—	—	z
3	6	3	z	z
4	2	3	y	—
5	—	4	z	—
6	2	1	y	—

Rys. 4.8. Tablica przejść - wyjść automatu z przykładu 4.2

W tym przypadku konieczne było włączenie jednoelementowych zbiorów $\{2\}$ i $\{5\}$; gdyż stany 2 i 5 nie występowały w pozostałych zbiorach.

2	(X)				
3	46	(X)			
4	(X)	(X)	(X)		
5	(X)	(X)	(X)	(X)	
6	(X)	(X)	(X)	13	(X)
	1	2	3	4	5

Rys. 4.9. Tablica trójkątna dla określenia relacji nieodróżnialności automatu z przykładu 4.2

2	(~~4~~ 5)				
3	46	(~~5~~ 6)			
4	(X)	25	(X)		
5	(~~3~~ 4)	V	(~~3~~ 4)	(X)	
6	(X)	25	(X)	13	(X)
	1	2	3	4	5

Rys. 4.10. Tablica trójkątna dla określenia relacji niesprzeczności automatu z przykładu 4.2

Należy zwrócić uwagę na fakt, że zbiory wchodzące w skład rodziny $\{Q_{\max}\}$ mogą nie być rozłączne, tzn. mogą zawierać elementy wspólne. W omawianym przykładzie elementem takim jest stan 2 występujący w zbiorze $\{2,4,6\}$ oraz $\{2,5\}$. Wynika to z definicji zbiorów stanów niesprzecznych (def. 4.9) oraz z definicji relacji niesprzeczności (def. 4.8). Z kolei zbiory wchodzące w skład rodziny maksymalnych zbiorów stanów nieodróżnialnych $\{B_{\max}\}$ są zawsze rozłączne, co jest konsekwencją definicji 4.11 i 4.12.

Procedurę poszukiwania rodziny maksymalnych zbiorów stanów niesprzecznych można zalgorytmować, korzystając bezpośrednio z rezultatu otrzymanego w tablicy trójkątnej. Wystarczy zauważyć, że jeśli mamy dwie pary stanów typu:

$$a, b \quad , \quad a, c$$

i jeśli istnieje para b, c lub zbiór zawierający tę parę, to wszystkie trzy pary można zastąpić zbiorem stanów niesprzecznych $\{a, b, c\}$. Ten sam tryb postępowania daje się rozciągnąć na większe zbiory, np. $\{a, b, c\}$ i $\{a, d\}$, $\{b, d\}$, $\{c, d\} \rightarrow \{a, b, c, d\}$. Pozwala to na określenie postępowania:

w połączeniu z trzema parami: 4 5, 4 6, 4 7 przechodzi w zbiór cztero-elementowy $\{4,5,6,7\}$. W wierszu 3 dochodzą pary 3 8, 3 6, które nie dadzą się włączyć w zbiory. W wierszu 2 pary: 2 4, 2 5, 2 6, 2 7 mogą być włączone do zbioru $\{4,5,6,7\}$, dając w wyniku zbiór pięcioelementowy stanów niesprzecznych $\{2,4,5,6,7\}$. Najwięcej łączeń wystąpi dla wiersza dolnego odpowiadającego stanowi 1, gdyż tworzy on pary niesprzeczne z wszystkimi pozostałymi stanami. W związku z tym wszystkie zbiory stanów niesprzecznych, które przeszły z wiersza odpowiadającego stanowi 2 zostają powiększone o stan 1. Rezultat końcowy jest następujący:

$$\{Q_{\max}\} = \left\{ \{1,2,4,5,6,7\}, \{1,3,8\}, \{1,5,8\}, \{1,3,6\} \right\}.$$

Według identycznego schematu można szukać rodziny maksymalnych zbiorów stanów nieodróżnialnych, najczęściej jednak wystarcza do tego celu metoda prób i błędów.

4.4.2. Procedura minimalizacji liczby stanów automatu pierwotnego

Rodzina maksymalnych zbiorów stanów niesprzecznych zawiera zbiory stanów tworzących pary niesprzeczne, a więc stany te dla określonej litery wejściowej generują taką samą literę wyjściową albo też litera ta jest nieokreślona. Z charakteru procedury pozwalającej na określenie poszczególnych zbiorów składowych rodziny $\{Q_{\max}\}$ wynika jeszcze jedna bardzo istotna jej właściwość. Mianowicie, stany następne wyznaczone dla określonej litery wejściowej, wyznaczone dla wszystkich stanów dowolnego zbioru maksymalnego, są zawsze podzbiorem któregoś ze zbiorów maksymalnych stanów niesprzecznych tworzących tę rodzinę. Innymi słowy te stany następne, które dla danej litery wejściowej są określone, zawsze tworzą podzbiór zbioru maksymalnych stanów niesprzecznych. Z uwagi na to, że zbiory rodziny $\{Q_{\max}\}$ nie są rozłączne, może się zdarzyć, że stany następne są jednocześnie podzbiorem większej liczby zbiorów maksymalnych.

Przykład 4.4. Dla rodziny

$$Q_{\max} = \left\{ \{1,2,4,5,6,7\}, \{1,3,8\}, \{1,5,8\}, \{1,3,6\} \right\} = \\ = \left\{ Q_{\max}^1, Q_{\max}^2, Q_{\max}^3, Q_{\max}^4 \right\}$$

automatu określonego tabelą pokazaną na rysunku 4.6, znaleźć i sklasyfikować podzbiory stanów następnych.

Stany następne zbioru maksymalnego

$$Q_{\max}^1 = \{1,2,4,5,6,7\}$$

dla litery a, są następujące:

$$\{2,4,2,5,-,1\} = \{1,2,4,5\} \subset Q_{\max}^1$$

i dalej dla litery a:

$$Q_{\max}^2 = \{1,3,8\} \rightarrow \{2,7,-\} = \{2,7\} \subset Q_{\max}^1,$$

$$Q_{\max}^3 = \{1,5,8\} \rightarrow \{2,5,-\} = \{2,5\} \subset Q_{\max}^1,$$

$$Q_{\max}^4 = \{1,3,6\} \rightarrow \{2,7,-\} = \{2,7\} \subset Q_{\max}^1.$$

Dla litery b uzyskamy następujący rezultat:

$$Q_{\max}^1 \rightarrow \{3,8\} \subset Q_{\max}^2; Q_{\max}^3 \rightarrow \{7\} \subset Q_{\max}^1,$$

$$Q_{\max}^2 \rightarrow \{7\} \subset Q_{\max}^1; Q_{\max}^4 \rightarrow \{8\} \subset Q_{\max}^2 \text{ lub } Q_{\max}^3.$$

Dla litery c:

$$Q_{\max}^1 \rightarrow \{6\} \subset Q_{\max}^4; Q_{\max}^3 \rightarrow \{1,6\} \subset Q_{\max}^4,$$

$$Q_{\max}^2 \rightarrow \{1,8\} \subset Q_{\max}^2 \text{ lub } Q_{\max}^3; Q_{\max}^4 \rightarrow \{8\} \subset Q_{\max}^2 \text{ lub } Q_{\max}^3.$$

Może się zdarzyć, że dla jakiegoś zbioru maksymalnego wszystkie stany następne są nieokreślone, ale nigdy nie zaistnieje sytuacja, w której tylko część stanów następnych utworzy podzbiór dowolnego zbioru maksymalnego.

Omówiona wyżej własność rodziny $\{Q_{\max}\}$ nosi nazwę zamkniętości tej rodziny. Jeśli liczba zbiorów rodziny $\{Q_{\max}\}$ jest mniejsza od liczby stanów automatu pierwotnego, to własność zamkniętości pozwala na zbudowanie automatu zredukowanego o liczbie stanów równej liczbie zbiorów maksymalnych. Każdy taki zbiór bowiem może być reprezentowany przez jeden stan automatu zredukowanego.

Przykład 4.5. W oparciu o rezultat otrzymany w przykładzie 4.4 określić automat zredukowany, dla automatu opisanego tabelą pokazaną na rysunku 4.6.

Automat zredukowany $A_r = \langle X, S_r, Y, \delta_r, \lambda_r \rangle$ będzie określony dla tego samego alfabetu wejściowego i wyjściowego. Inny natomiast będzie alfabet wewnętrzny i funkcje wyjść i przejść. Zbiór stanów wewnętrznych będzie zbiorem czteroelementowym $S_r = \{q_1, q_2, q_3, q_4\}$. Poszczególne stany wewnętrzne są przypisane zbiorom rodziny $\{Q_{\max}\}$ w następujący sposób:

$$Q_{\max}^1 = \{1, 2, 4, 5, 6, 7\} \rightarrow q_1,$$

$$Q_{\max}^2 = \{1, 3, 8\} \rightarrow q_2,$$

$$Q_{\max}^3 = \{1, 5, 8\} \rightarrow q_3,$$

$$Q_{\max}^4 = \{1, 3, 6\} \rightarrow q_4.$$

Funkcję δ_r zgodnie z rezultatami uzyskanymi w przykładzie 4.4, zdefiniujemy następująco:

$$\delta_r(a, q_1) = q_1, \text{ bowiem } \delta(a, Q_{\max}^1) \subset Q_{\max}^1,$$

$$\delta_r(a, q_2) = q_1, \text{ bowiem } \delta(a, Q_{\max}^2) \subset Q_{\max}^1,$$

$$\delta_r(a, q_3) = q_1, \text{ bowiem } \delta(a, Q_{\max}^3) \subset Q_{\max}^1,$$

$$\delta_r(a, q_4) = q_1, \text{ bowiem } \delta(a, Q_{\max}^4) \subset Q_{\max}^1,$$

$$\delta_r(b, q_1) = q_2, \text{ bowiem } \delta(b, q_{\max}^1) \subset q_{\max}^2,$$

$$\delta_r(b, q_2) = q_1, \text{ bowiem } \delta(b, q_{\max}^2) \subset q_{\max}^1,$$

$$\delta_r(b, q_3) = q_1, \text{ bowiem } \delta(b, q_{\max}^3) \subset q_{\max}^1,$$

$$\delta_r(b, q_4) = q_2 \text{ lub } q_3 \text{ bowiem } \delta(b, q_{\max}^4) \subset q_{\max}^2 \text{ lub } q_{\max}^3,$$

$$\delta_r(c, q_1) = q_4, \text{ bowiem } \delta(c, q_{\max}^1) \subset q_{\max}^4,$$

$$\delta_r(c, q_2) = q_2 \text{ lub } q_3, \text{ bowiem } \delta(c, q_{\max}^2) \subset q_{\max}^2 \text{ lub } q_{\max}^3,$$

$$\delta_r(c, q_3) = q_4 \text{ bowiem } \delta(c, q_{\max}^3) \subset q_{\max}^4,$$

$$\delta_r(c, q_4) = q_2 \text{ lub } q_3, \text{ bowiem } \delta(c, q_{\max}^4) \subset q_{\max}^2 \text{ lub } q_{\max}^3.$$

Otrzymany rezultat określa tablicę przejść automatu zredukowanego. Tablicę wyjść łatwo określić, gdyż dla wszystkich stanów zgromadzonych w danym zbiorze maksymalnym literę wyjściową, odpowiadającą poszczególnym literom wejściowym, są albo nieokreślone, albo takie same. Można więc napisać:

$$\lambda_r(a, q_1) = \lambda(a, q_{\max}^1) = z,$$

$$\lambda_r(a, q_2) = \lambda(a, q_{\max}^2) = z,$$

$$\lambda_r(a, q_3) = \lambda(a, q_{\max}^3) = z,$$

$$\lambda_r(a, q_4) = \lambda(a, q_{\max}^4) = z,$$

$$\lambda_r(b, q_1) = \lambda(b, q_{\max}^1) = z,$$

$$\lambda_r(b, q_2) = \lambda(b, q_{\max}^2) = y,$$

$$\lambda_r(b, q_3) = \lambda(b, q_{\max}^3) = y,$$

$$\lambda_r(b, q_4) = \lambda(b, q_{\max}^4) = -,$$

$$\lambda_r(c, q_1) = \lambda(c, q_{\max}^1) = y,$$

$$\lambda_r(c, q_2) = \lambda(c, q_{\max}^2) = v,$$

$$\lambda_r(c, q_3) = \lambda(c, q_{\max}^3) = y,$$

$$\lambda_r(c, q_4) = \lambda(c, q_{\max}^4) = -.$$

W dwóch przypadkach otrzymaliśmy litery wyjściowe nieokreślone. Tabelę przejść - wyjść otrzymanego automatu przedstawiono na rysunku 4.12. Zaznaczono w niej alternatywy wyboru stanów następnych. Każdy wariant opisuje inny automat zredukowany. Dla omawianej rodziny $\{Q_{\max}\}$, można więc określić kilka automatów zredukowanych.

Omówione w powyższym przykładzie automaty zredukowane, w kategoriach wejścia i wyjścia realizują te same funkcje co automat pierwotny opisany tabelą pokazaną na rysunku 4.6. Powstaje pytanie, czy są to automaty minimalne, tzn. takie, z których każdy posiada najmniejszą możliwą liczbę stanów wewnętrznych. Aby odpowiedzieć na to pytanie zauważmy, że gdyby maksymalnie zredukować rodzinę $\{Q_{\max}\}$, usuwając pewne zbiory, ale w ten sposób, że:

	a	b	c		a	b	c
q ₁	q ₁	q ₂	q ₄		z	y	y
q ₂	q ₁	q ₁	q ₂ (q ₃)		z	y	y
q ₃	q ₁	q ₁	q ₄		z	-	y
q ₄	q ₁	q ₂ (q ₃)	q ₂ (q ₃)		z	y	-

Rys. 4.12. Tablica przejść - wyjść automatu zredukowanego opisanego w przykładzie 4.5

a) w pozostałych zbiorach byłyby reprezentowane wszystkie stany automatu pierwotnego, czyli byłyby spełniony tzw. warunek pełnego pokrycia stanów tego automatu,

b) tak zredukowana rodzina spełniała by warunek zamkniętości, to tak otrzymany automat w kategoriach wejścia - wyjścia spełniałby tę samą funkcję co automat pierwotny i jednocześnie posiadałby najmniejszą możliwą liczbę stanów wewnętrznych. Spełnienie funkcji wejścia - wyjścia wynika z faktu pokrycia wszystkich stanów automatu pierwot-

nego, a więc każda sytuacja wejściowa i wyjściowa jest przez ten automat reprezentowana. Jednocześnie z tego faktu oraz z warunku zamkniętości rodziny maksymalnie zredukowanej wynika, że zbudowany w oparciu o tę rodzinę automat będzie automatem minimalnym.

Maksymalnie zredukowaną rodzinę $\{Q_{\max}\}$ określamy często jako tzw. rodzinę $\{P_j\}$ [2], natomiast automaty budowane na rodzinach zbiorów stanów wewnętrznych są określane jako tzw. automaty ilorazowe. Określone w przykładzie 4.5 automaty zredukowane są więc automatai ilorazowymi. Ogólne oznaczenie automatu ilorazowego zbudowanego dla rodziny zbiorów $\{T_j\}$ jest postaci $A / \{T_j\}$, gdzie A - nazwa automatu. Dla rodzin $\{Q_{\max}\}$, $\{B_{\max}\}$, $\{P_j\}$ oznaczenia automatów ilorazowych przyjmą postać:

$$A / \{Q_{\max}\}, \quad A / \{B_{\max}\}, \quad A / \{P_j\}. \quad (1)$$

Zgodnie z tym co powiedziano uprzednio, automat ilorazowy dla rodziny $\{P_j\}$ jest automatem minimalnym.

Przykład 4.6. Określić rodzinę $\{P_j\}$ automatu opisanego tabelą pokazaną na rysunku 4.6 oraz znaleźć automat minimalny.

Rodzinę $\{Q_{\max}\}$ dla tego automatu określono w przykładzie 4.4. Ma ona postać:

$$\{Q_{\max}\} = \left\{ \{1, 2, 4, 5, 6, 7\}, \{1, 3, 8\}, \{1, 5, 8\}, \{1, 3, 6\} \right\}.$$

Dla pełnego pokrycia wszystkich stanów automatu pierwotnego wystarczy tylko zbiory: $\{1, 2, 4, 5, 6, 7\}$ i $\{1, 3, 8\}$. Powstaje pytanie czy stanowią one rodzinę $\{P_j\}$, tzn. czy spełniają jeszcze warunek zamkniętości. Aby to sprawdzić posłużmy się tabelą pokazaną na rysunku 4.13.

	1	2	4	5	6	7	1	3	8
a	2	4	2	5	-	1	2	7	-
b	-	3	3	-	8	8	-	-	7
c	-	6	-	6	-	-	-	8	1

Rys. 4.13. Tablica do testowania zamkniętości rodziny zbiorów otrzymanych z maksymalnej redukcji rodziny $\{Q_{\max}\}$ (przykład 4.6)

Zawiera ona stany następne otrzymane w wyniku pobudzenia określoną literą zbiorów rodziny zredukowanej. Dla zbioru $\{1,2,4,5,6,7\}$ oraz liter a, c stany następne stanowią jej własny podzbiór, dla litery b są podzbiorem zbioru $\{1,3,8\}$. Stan następny zbioru $\{1,3,8\}$ jest jego podzbiorem dla litery c, w pozostałych przypadkach jest podzbiorem zbioru $\{1,2,4,5,6,7\}$. Otrzymana rodzina spełnia warunek pokrycia i zamkniętości; jest więc rodziną $\{P_j\}$. Automat ilorazowy dla tej rodziny

$$A_{\min} = \langle X, S_m, Y, \delta_m, \lambda_m \rangle = A / \{P_j\}$$

określimy w sposób analogiczny jak w przykładzie 4.5. Oznaczmy:

$$\{P_j\} = \left\{ \{1,2,4,5,6,7\}, \{1,3,8\} \right\} = \{P_1, P_2\},$$

przy czym

$$P_1 = \{1,2,4,5,6,7\}; \quad P_2 = \{1,3,8\}.$$

Oznaczmy zbiór stanów wewnętrznych szukanego automatu ilorazowego przez $S_m = \{p_1, p_2\}$. Wtedy dla funkcji przejść δ_m , można napisać (rys. 4.13):

	a	b	c	a	b	c
p_1	p_1	p_2	p_1	z	z	y
p_2	p_1	p_1	p_2	z	y	y

Rys. 4.14. Tablica przejść - wyjść automatu minimalnego określonego dla automatu opisanego tabelą pokazaną na rys. 4.6 (przykład 4.6)

$$\delta_m(a, p_1) = p_1, \text{ gdyż } \delta(a, P_1) \subset P_1,$$

$$\delta_m(a, p_2) = p_1, \text{ gdyż } \delta(a, P_2) \subset P_1,$$

$$\delta_m(b, p_1) = p_2, \text{ gdyż } \delta(b, P_1) \subset P_2,$$

$$\delta_m(b, p_2) = p_1, \text{ gdyż } \delta(b, P_2) \subset P_1,$$

$$\delta_m(c, p_1) = p_1, \text{ gdyż } \delta(c, P_1) \subset P_1,$$

$$\delta_m(c, p_2) = p_2, \text{ gdyż } \delta(c, P_2) \subset P_2.$$

Funkcję wyjść opisują następujące zależności:

$$\lambda_m(a, p_1) = \lambda(a, P_1) = z,$$

$$\lambda_m(a, p_2) = \lambda(a, P_2) = z,$$

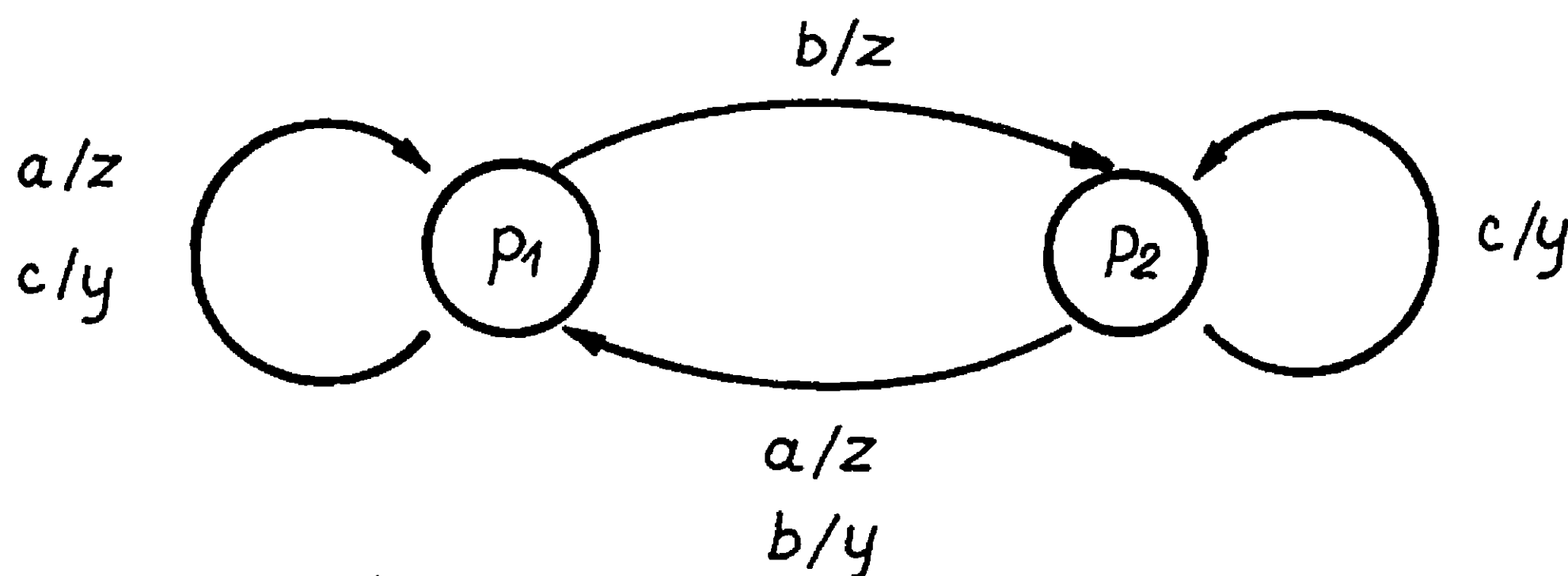
$$\lambda_m(b, p_1) = \lambda(b, P_1) = z,$$

$$\lambda_m(b, p_2) = \lambda(b, P_2) = y,$$

$$\lambda_m(c, p_1) = \lambda(c, P_1) = y,$$

$$\lambda_m(c, p_2) = \lambda(c, P_2) = y.$$

Powyższe rezultaty wpisano w tabelę pokazaną na rysunku 4.14, a na rysunku 4.15 przedstawiono graf automatu. Należy zwrócić uwagę na fakt znacznej redukcji liczby stanów w porównaniu z liczbą stanów automatu wyjściowego. Stało się to możliwe m.in. dzięki występowaniu w automacie pierwotnym dużej liczby stanów i liter wyjściowych nieokreślonych.



Rys. 4.15. Graf automatu minimalnego opisanego tablicą pokazaną na rys. 4.14 (przykład 4.6)

Przykład 4.7. Znaleźć automat minimalny dla automatu Moore'a, którego tablicę przejść - wyjść podano na rysunku 4.16.

Na rysunku 4.17 podano tabelę trójkątną oraz schemat dla określenia rodziny $\{Q_{\max}\}$. Z rysunku tego wynika, że

$$\{Q_{\max}\} = \left\{ \{3, 4, 5\}, \{2, 3, 5\}, \{1, 3, 5\} \right\}.$$

Ponieważ wszystkie zbiory są potrzebne dla pokrycia stanów automatu pierwotnego, więc $\{Q_{\max}\} = \{P_j\}$. Na rysunku 4.18 przedstawiono graf oraz tabelę dla automatu minimalnego z zaznaczeniem zbiorów, dla których zostały określone wg uprzednio podanych reguł stany następne i litery wyjściowe. Charakterystyczną rolę pełni stan p_3 . Można do niego wejść, ale nie da się z niego wyjść. Taki przypadek nosi często nazwę "studni". Wyjście ze "studni" jest możliwe tylko poprzez wymuszenie innego stanu w sposób "sztuczny". Grafy o tego typu konstrukcji są określane czasem jako grafy z osobliwościami.

	a	b	
1	-	1	0
2	1	4	0
3	-	3	-
4	-	2	1
5	1	5	-

Rys. 4.16. Tablica przejść - wyjść automatu z przykładu 4.7

a)

2	~~X~~			
3	V	34		
4	X	X	23	
5	V	45	V	25
	1	2	3	4

b)

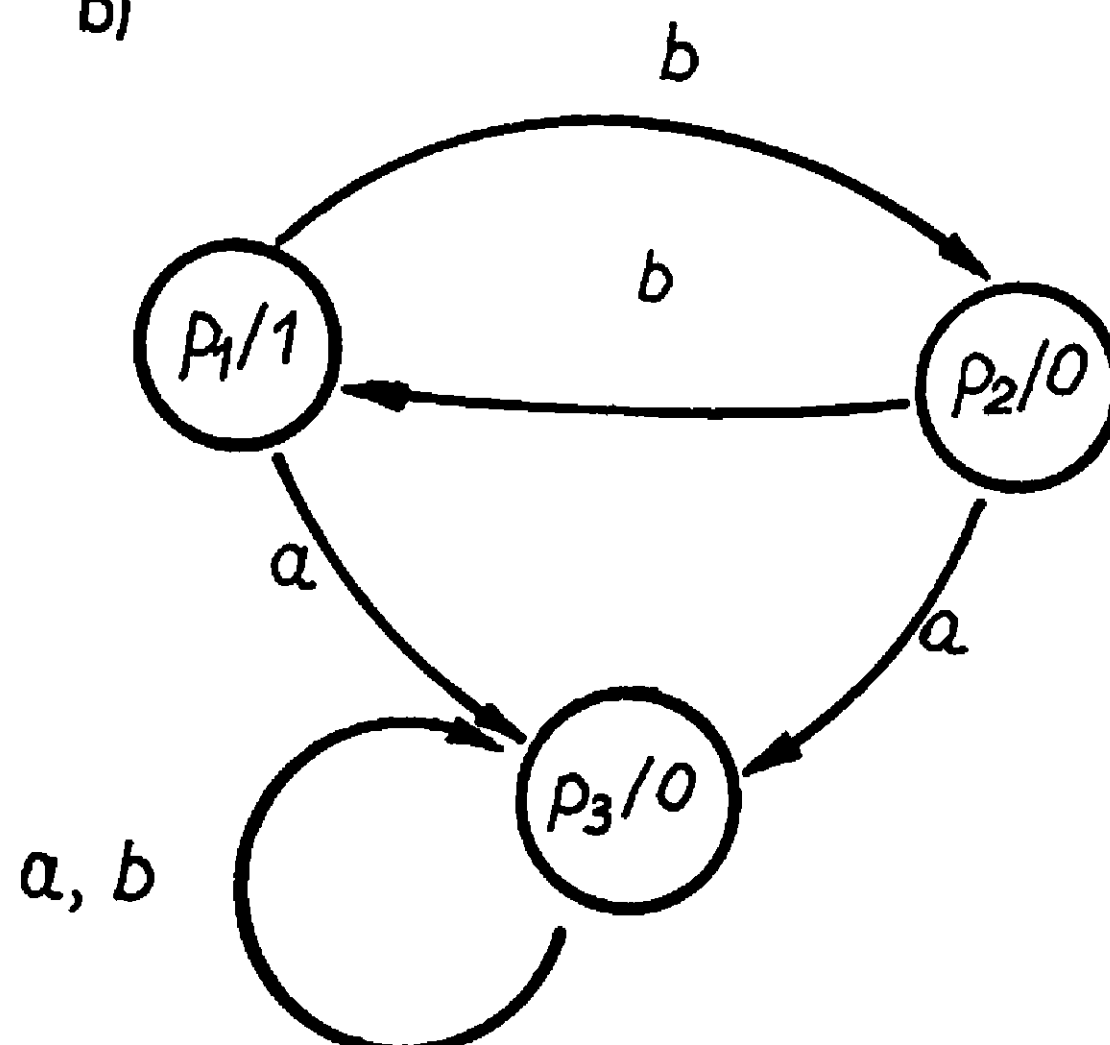
4	45				
3	~~45~~	~~34~~	~~35~~	345	
2	345	~~23~~	~~25~~	235	
1	345	235	~~13~~	~~15~~	135

Rys. 4.17. Tablica trójkątna i schemat dla określania rodziny $\{Q_{\max}\}$ automatu z przykładu 4.7 (rys. 4.16)

a)

	a	b	
$p_1(345)$	p_3	p_2	1
$p_2(235)$	p_3	p_1	0
$p_3(135)$	p_3	p_3	0

b)



Rys. 4.18. Tablica przejść - wyjść i graf automatu minimalnego z przykładu 4.7

Przykład 4.8. Dany automat jest opisany tabelą przejść - wyjść pokazaną na rysunku 4.19a. Określić automat minimalny oraz automat ilorazowy dla rodziny $\{B_{\max}\}$.

a)

	a	b	a	b
1	4	3	z	z
2	5	-	-	z
3	6	3	z	z
4	2	3	y	-
5	-	4	z	-
6	2	1	y	-

b)

	a	b	a	b
$p_1(246)$	p_2	p_3	y	z
$p_2(25)$	p_2	p_1	z	z
$p_3(13)$	p_1	p_3	z	z

c)

	a	b	a	b
$b_1(1,3)$	b_2	b_1	z	z
$b_2(4,6)$	b_3	b_1	y	-
$b_3(2)$	b_4	-	-	z
$b_4(5)$	-	b_2	z	-

d)

	a	b	a	b
$q_1(b_3b_4)$	q_1	q_2	z	z
$q_2(b_2b_3)$	q_1	q_3	y	z
$q_3(b_1)$	q_2	q_3	z	z

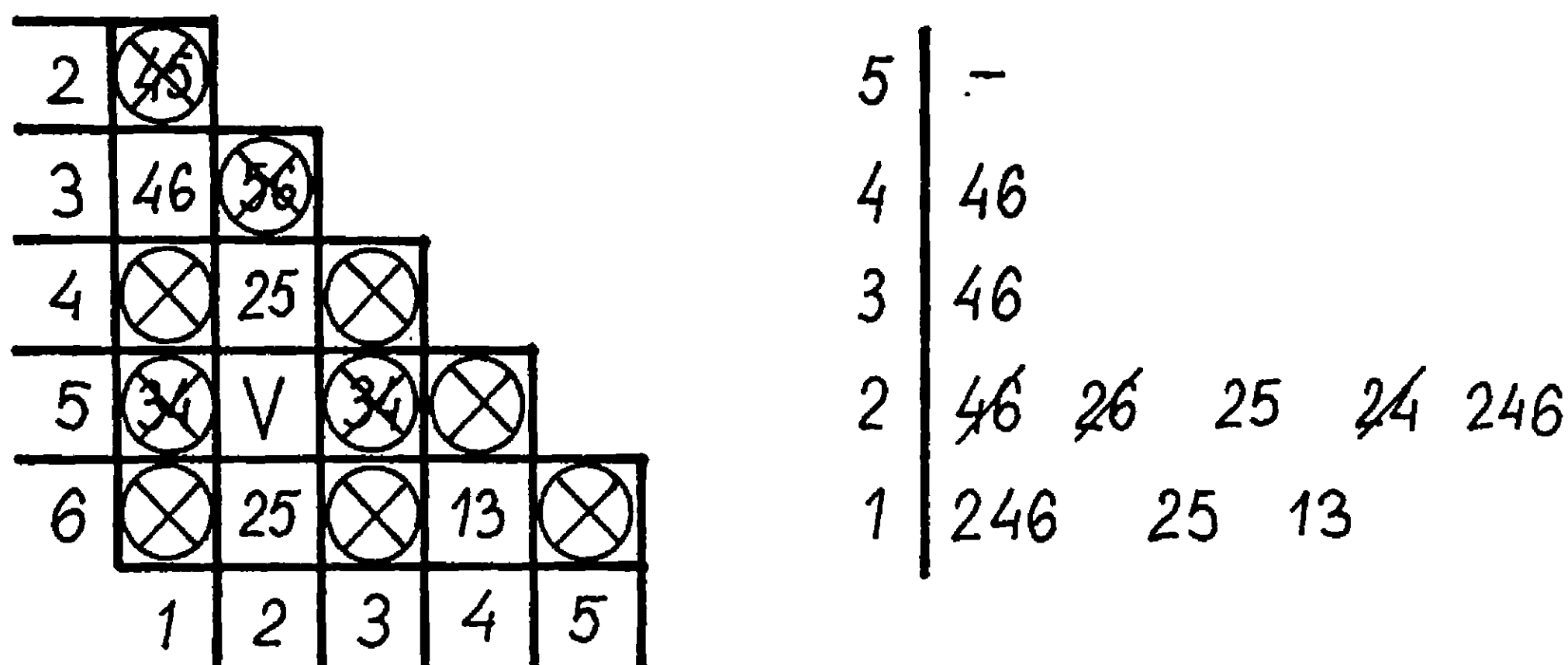
Rys. 4.19. Tablice przejść - wyjść do przykładu 4.8 (a,b,c,d)

Na rysunku 4.20 przedstawiono tabelę trójkątną oraz schemat dla określania rodziny $\{Q_{\max}\}$. Tabela trójkątna jest powtórzeniem tabeli z przykładu 4.2, gdzie dla tego samego automatu określono relację nieodróżnialności i niesprzeczności stanów. Rezultat

$$\{Q_{\max}\} = \left\{ \{2,4,6\}, \{2,5\}, \{1,3\} \right\}$$

był w przykładzie 4.2 osiągnięty intuicyjnie. Ponieważ wszystkie zbiory potrzebne są do pokrycia stanów automatu pierwotnego, a więc $\{Q_{\max}\} =$

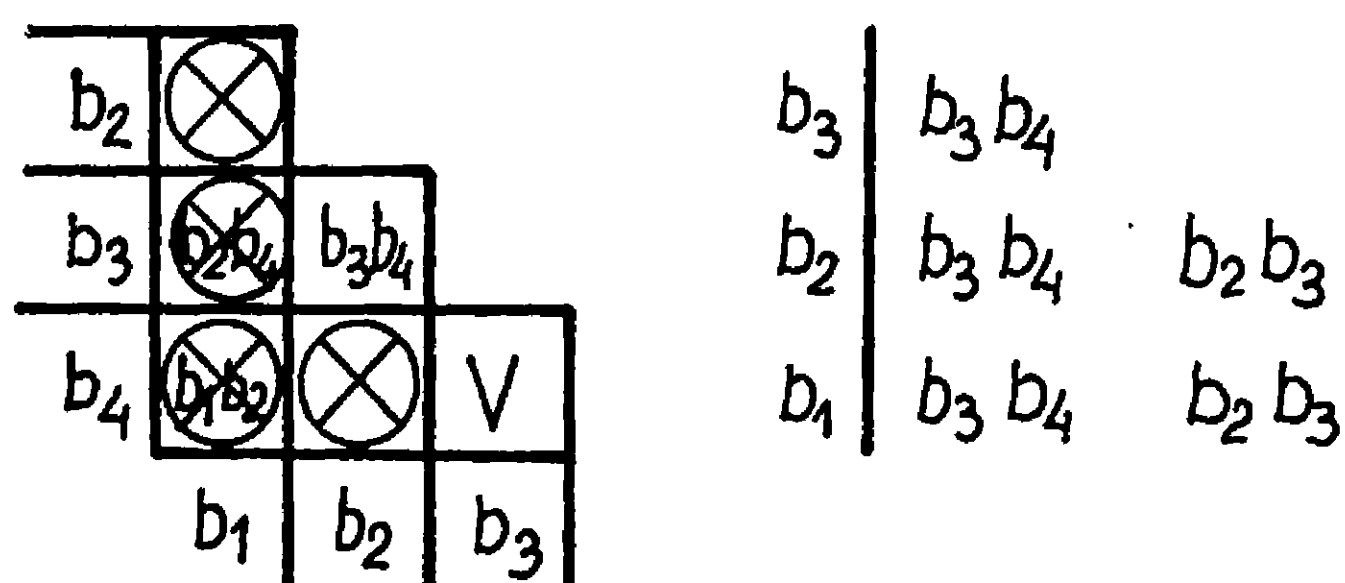
$= \{P_j\}$. Automat minimalny jest opisany tabelą podaną na rysunku 4.19b. Rodzinę $\{B_{\max}\}$ określono w przykładzie 4.2.



Rys. 4.20. Tablica trójkątna oraz schemat dla określenia rodziny $\{Q_{\max}\}$, automatu opisanego na rys. 4.19a (przykład 4.8)

$$\{B_{\max}\} = \left\{ \{1,3\}, \{4,6\}, \{2\}, \{5\} \right\}.$$

Odpowiadający tej rodzinie automat ilorazowy podano na rysunku 4.19c. Należy zwrócić uwagę na fakt, że ze względu na rozłączność zbiorów rodziny $\{B_{\max}\}$ zawsze istnieje tylko jeden automat ilorazowy $A/\{B_{\max}\}$.



Rys. 4.21. Tablica trójkątna oraz schemat dla określenia rodziny $\{Q_{\max}\}$ dla automatu $A/\{B_{\max}\}$ (przykład 4.8)

Jako dodatkowe zadanie poszukajmy automatu minimalnego do automatu $A/\{B_{\max}\}$. Na rysunku 4.21 przedstawiono tablicę trójkątną oraz schemat określający rodzinę maksymalnych zbiorów stanów niesprzecz-

nych automatu $A/\{B_{\max}\}$. Do otrzymanego rezultatu należy jeszcze dołączyć zbiór jednoelementowy $\{b_1\}$, gdyż inaczej nie byłby spełniony warunek pokrycia. Ostatecznie można napisać

$$\{Q_{\max}\}' = \left\{ \{b_3, b_4\}, \{b_2, b_3\}, \{b_1\} \right\}.$$

Rodziny tej dalej zredukować się nie da z uwagi na wspomniany już warunek pokrycia, czyli $\{Q_{\max}\}' = \{P_j\}$. Na rysunku 4.19d przedstawiono automat minimalny odpowiadający automатовi $A/\{B_{\max}\}$. Ma on identyczną liczbę stanów jak automat minimalny określony tabelą pokazaną na rysunku 4.19b. Więcej nawet, jeśli w automacie $M/\{Q_{\max}\}$, zamienić miejscami q_1 z wierszem q_2 oraz wprowadzić następującą zamianę oznaczeń:

$$q_1 \rightarrow p_2, \quad q_2 \rightarrow p_1, \quad q_3 \rightarrow p_3,$$

to staną się one identyczne. Automaty, które się różnią tylko układem tabeli i oznaczeniami nazywamy automatami izomorficznymi. Z uwagi na zdarzającą się alternatywność wyboru stanów następnych, zilustrowaną w przykładzie 4.5, nie zawsze automat minimalny do automatu $M/\{B_{\max}\}$ musi być izomorficzny z automatem otrzymanym bezpośrednio dla rodziny $\{P_j\}$, zawsze jednak będzie miał taką samą liczbę stanów wewnętrznych. Wyłania się w związku z tym możliwość przeprowadzenia minimalizacji "na raty". Najpierw określamy automat ilorazowy dla rodziny $\{B_{\max}\}$, a potem określamy dla tego automatu automat minimalny. Może okazać się to korzystne dla automatów pierwotnych o rozbudowanych tabelach przejść - wyjść, gdyż znajdowanie relacji nieodróżnialności jest znacznie prostsze niż relacje niesprzeczności.

4.5. Podobieństwo automatów Moore'a i Mealy'ego

Minimalizacja liczby stanów automatu pierwotnego odbywa się albo w klasie automatów typu Mealy'ego, albo też w klasie automatów typu Moore'a. Rytm pracy automatu Moore'a jest inny niż Mealy'ego, gdyż w ostatnim przypadku nowa litera wyjściowa jest generowana przed wejściem do stanu następnego, natomiast w automacie Moore'a dopiero z chwilą przejścia do tego stanu. Ma to konkretne następstwa funkcjonalne: w automacie Moore'a nieokreśloność stanu następnego pociąga za sobą nieokreśloność litery wyjściowej, natomiast dla automatu Mealy'ego nieokreśloność stanu następnego nie wpływa na literę wyjściową, która



jest wygenerowana przed wypadnięciem z grafu poprzez stan nieokreślony (p. 4.2). Relacje pokrywania i nieodróżnialności nie mogą więc zachodzić między tymi automatami, można natomiast mówić o podobieństwie w odniesieniu do roli jaką spełniają w określonym układzie. Ujmują to poniższe definicje.

Definicja 4.14. Automaty: Moore'a $A_1 = \langle X, S_1, Y, \delta_1, \lambda_1 \rangle$ w stanie s_1 i Mealy'ego $A_2 = \langle X, S_2, Y, \delta_2, \lambda_2 \rangle$ w stanie s_2 nazywamy podobnymi (co zapisujemy $s_1 \approx s_2$), wtedy gdy każde słowo odpowiednie dla s_1 jest także odpowiednie dla s_2 i odwrotnie oraz dla każdego takiego słowa końcowe litery wyjściowe są sobie równe

$$(x \parallel s_1 \cap x \parallel s_2) \rightarrow \lambda(x, s_1) = \lambda(x, s_2).$$

Definicja 4.15. Automaty: Moore'a A_1 i Mealy'ego A_2 nazywamy podobnymi ($A_1 \approx A_2$), jeżeli dla każdego stanu s_1 automatu A_1 istnieje przynajmniej jeden stan s_2 automatu A_2 , spełniający definicję 4.14 oraz dla każdego stanu s_2 automatu A_2 istnieje przynajmniej jeden stan s_1 automatu A_1 również spełniający definicję 4.14.

Obie definicje są analogiczne do definicji 4.6 i 4.7, określających automaty nieodróżnialne, jednak w tym przypadku można używać jedynie terminu podobieństwa.

Na podstawie definicji 4.14 i 4.15 można opracować metody umożliwiające przejście od automatu typu Moore'a do automatu typu Mealy'ego i odwrotnie. Jeśli dla każdej litery wejściowej, dla automatu Mealy'ego $A = \langle X, S, Y, \delta, \lambda \rangle$ będzie spełniona zależność

$$\lambda(x, s) = \lambda'[\delta'(x, s)], \quad (1)$$

przy czym δ' i λ' określają funkcje przejść i wyjść automatu Moore'a $A = \langle X, S, Y, \delta', \lambda' \rangle$, to jest spełniona definicja 4.15 o podobieństwie obu automatów. Zależność (1) umożliwia określenie algorytmu umożliwiającego konstruowanie automatów podobnych. Rozpatrzmy to na przykładach.

Przykład 4.9. Dla automatu Moore'a opisanego tabelą podaną na rysunku 4.22a znaleźć automat podobny typu Mealy'ego.

Zgodnie z zależnością (1) można napisać:

$$\lambda(a, 1) = \lambda'[\delta'(a, 1)] = z,$$

$$\lambda(a, 2) = \lambda'[\delta'(a, 2)] = z,$$

$$\lambda(a, 3) = \lambda'[\delta'(a, 3)] = y,$$

$$\lambda(a, 4) = \lambda'[\delta'(a, 4)] = -,$$

$$\lambda(a, 5) = \lambda'[\delta'(a, 5)] = y,$$

$$\lambda(b, 1) = \lambda'[\delta'(b, 1)] = -,$$

$$\lambda(b, 2) = \lambda'[\delta'(b, 2)] = y,$$

$$\lambda(b, 3) = \lambda'[\delta'(b, 3)] = z,$$

$$\lambda(b, 4) = \lambda'[\delta'(b, 4)] = z,$$

$$\lambda(b, 5) = \lambda'[\delta'(b, 5)] = y.$$

a)	a	b		
1	2	3	y	
2	4	5	z	
3	1	2	-	
4	3	4	z	
5	5	1	y	

b)	a	b	a	b
1	2	3	z	-
2	4	5	z	y
3	1	2	y	z
4	3	4	-	z
5	5	1	y	y

Rys. 4.22. Tablice przejść - wyjść automatów z przykładu 4.9 (a,b)

Powyższe rezultaty wpisano do tabeli przedstawionej na rysunku 4.22b. Prezentowane tam tabele przedstawiają więc automaty podobne.

Przykład 4.10. Znaleźć automat minimalny do automatu opisanego tabelą pokazaną na rysunku 4.23.

Rodzina $\{Q_{\max}\}$, wyznaczona dla omawianego automatu, jest następująca:

$$\{Q_{\max}\} = \left\{ \{1, 3, 5\}, \{2, 4\}, \{1, 6\}, \{1, 2\}, \{3, 4\} \right\}.$$

a)

	a	b	c	
1	3	4	-	-
2	4	-	3	v
3	1	2	-	u
4	2	4	1	-
5	5	2	1	u
6	5	2	6	w

b)

2	34				
3	13 24	⊗			
4	⊗ 23	24 13	12 24		
5	35 24	⊗	15	⊗ 25 24	
6	35 24	⊗	⊗	⊗ 25 24	⊗
	1	2	3	4	5

Rys. 4.23. Tablica przejść - wyjść (a) oraz tablica trójkątna automatu z przykładu 4.10(b)

Do pełnego pokrycia stanów automatu pierwotnego nie są potrzebne wszystkie zbiory maksymalne stanów niesprzecznych. Dokonajmy selekcji wybierając te, które jako jedyne pokrywają określone stany. Najmniejsza taka rodzina ma postać

$$\{P_j\} = \left\{ \{1,3,5\}, \{2,4\}, \{1,6\} \right\}.$$

a)

	a	b	c	
$p_0(135)$	p_0	p_1	p_0 p_2	u
$p_1(24)$	p_1	p_1	p_0	v
$p_2(16)$	p_0	p_1	p_2	w

b)

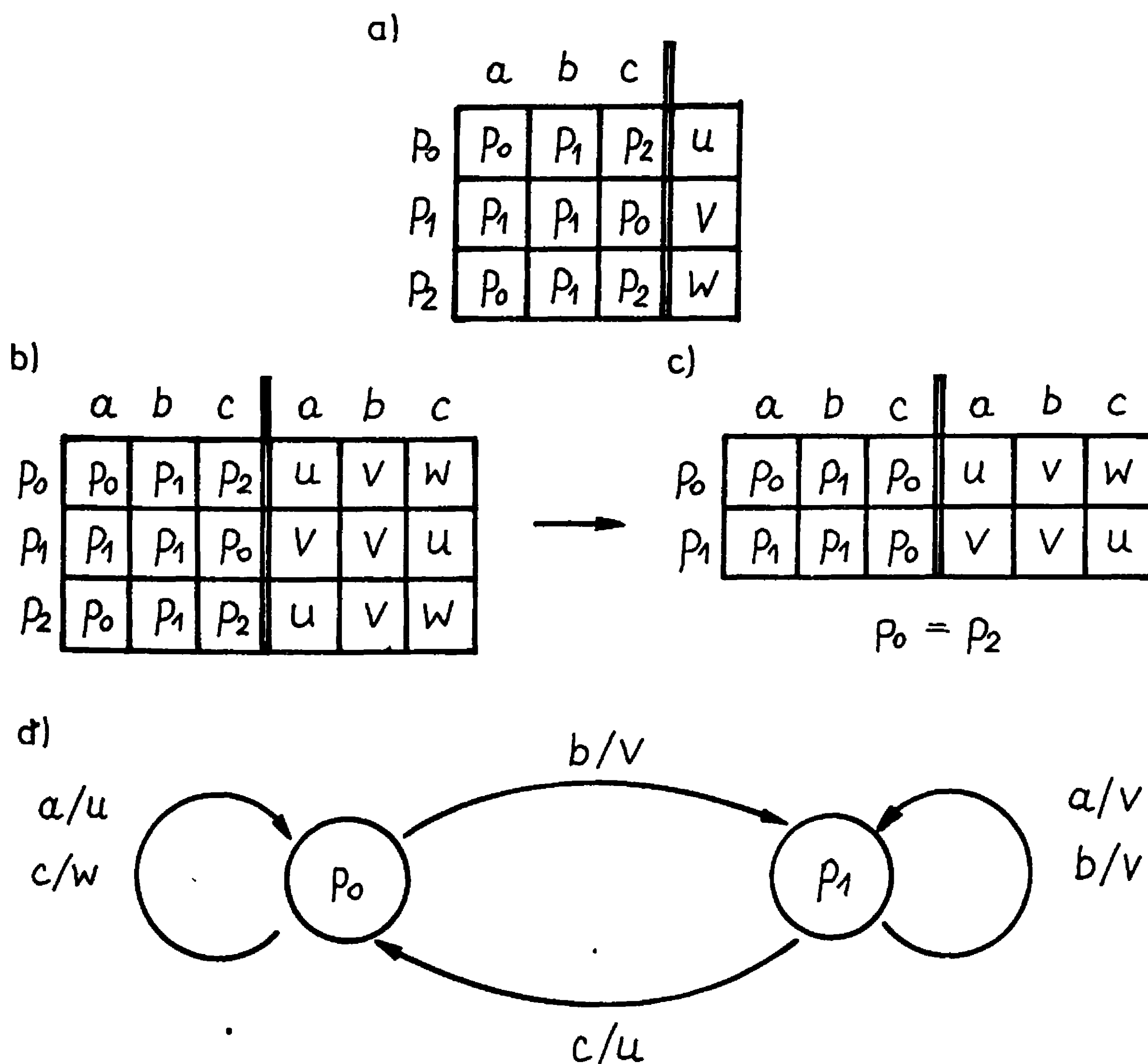
5	-								
4	-								
3	34	35							
2	34	35	24						
1	34	~~35~~	24	12	~~13~~	~~15~~	16	135	

Rys. 4.24. Tablica automatu minimalnego do automatu opisanego tablicą z rys. 4.23 (a) oraz schemat wyznaczania rodziny $\{Q_{\max}\}$ (przykład 4.10) (b)

łatwo sprawdzić, że jest spełniony warunek zamkniętości; jest to więc rodzina $\{P_j\}$. Na rysunku 4.24a przedstawiono automat ilorazowy zbudowany dla tej rodziny. Automatu tego nie da się dalej minimalizować jako automatu Moore'a. Jeśli jednak dla przejścia $\delta(c, p_0)$ przyjąć wartość

$$\delta(c, p_0) = p_2,$$

to wiersze odpowiadające stanom następnym stanów p_0 i p_2 są identyczne, różne są tylko litery wyjściowe związane z tymi stanami. Na podstawie zależności (1) można przewidzieć, że jeśli zbudujemy automat Mealy'ego podobny do opisanego na rysunku 4.24a, to wiersze dla stanów p_0 i p_2 staną się identyczne i z jednego z nich można więc zrezygnować. Na rysunku 4.25 przedstawiono automat minimalny Moore'a, automat Mealy'ego do niego podobny oraz automat z wyeliminowanym wierszem równoważnym dla stanu p_2 (rys. 4.25c).



Rys. 4.25. Tablice automatów podobnych Moore'a i Mealy'ego z przykładu 4.10 (a,b,c) oraz graf automatu z wyeliminowanym wierszem równoważnym (d)

Przedstawiona w przykładzie 4.10 procedura dalszej redukcji minimalnego automatu Moore'a poprzez sklejenie identycznych wierszy tabeli przejść - wyjść podobnego automatu Mealy'ego, może być rozwinięta na większą liczbę stanów. Wynika stąd wniosek, że automat Mealy'ego w pewnych przypadkach lepiej poddaje się redukcji niż podobny do niego automat Moore'a.

Przejście od automatu Mealy'ego do automatu Moore'a odbywa się również zgodnie z zależnością (1), jednak sama procedura jest trochę bardziej złożona, gdyż stanom i literom wyjściowym automatu Mealy'ego trzeba przypisywać stany automatu Moore'a. Rozpatrzmy to na przykładach.

Przykład 4.11. Określić automat Moore'a podobny do automatu Mealy'ego opisanego tabelą podaną na rysunku 4.26a.

a)	<table> <tr> <th></th><th>a</th><th>b</th><th>a</th><th>b</th></tr> <tr> <th>1</th><td>4</td><td>3</td><td>y</td><td>z</td></tr> <tr> <th>2</th><td>2</td><td>1</td><td>z</td><td>y</td></tr> <tr> <th>3</th><td>2</td><td>3</td><td>z</td><td>z</td></tr> <tr> <th>4</th><td>4</td><td>1</td><td>y</td><td>y</td></tr> </table>					a	b	a	b	1	4	3	y	z	2	2	1	z	y	3	2	3	z	z	4	4	1	y	y	<table> <tr> <th></th><th>a</th><th>b</th></tr> <tr> <th>1</th><td>4/y A</td><td>3/z B</td></tr> <tr> <th>2</th><td>2/z C</td><td>1/y D</td></tr> <tr> <th>3</th><td>2/z C</td><td>3/z B</td></tr> <tr> <th>4</th><td>4/y A</td><td>1/y D</td></tr> </table>					a	b	1	4/y A	3/z B	2	2/z C	1/y D	3	2/z C	3/z B	4	4/y A	1/y D	<table> <tr> <th></th><th>a</th><th>b</th></tr> <tr> <th>A</th><td>A</td><td>D</td><td>y</td></tr> <tr> <th>B</th><td>C</td><td>B</td><td>z</td></tr> <tr> <th>C</th><td>C</td><td>D</td><td>z</td></tr> <tr> <th>D</th><td>A</td><td>B</td><td>y</td></tr> </table>					a	b	A	A	D	y	B	C	B	z	C	C	D	z	D	A	B	y
	a	b	a	b																																																																			
1	4	3	y	z																																																																			
2	2	1	z	y																																																																			
3	2	3	z	z																																																																			
4	4	1	y	y																																																																			
	a	b																																																																					
1	4/y A	3/z B																																																																					
2	2/z C	1/y D																																																																					
3	2/z C	3/z B																																																																					
4	4/y A	1/y D																																																																					
	a	b																																																																					
A	A	D	y																																																																				
B	C	B	z																																																																				
C	C	D	z																																																																				
D	A	B	y																																																																				

Rys. 4.26. Tablice przejść - wyjść automatu Mealy'ego (a,b) i podobnego do niego automatu Moore'a (przykład 4.11), (c)

Procedurę określania automatu podobnego zaczniemy od modyfikacji tablicy przejść - wyjść automatu Mealy'ego pokazanej na rysunku 4.26b. W wierszach nad kreską umieszczono symbol stanu następnego odpowiadający literze wejściowej, pod kreską symbol litery wyjściowej związanej również z tą literą i stanem aktualnym. Należy zwrócić uwagę na fakt, że stan nad kreską (następny), pojawia się z opóźnieniem w stosunku do litery pod kreską (wyjściowej) równym jednej jednostce dyskretnej skali czasu. Każdej parze: stan następny / litera wyjściowa, przypisujemy nazwę pod jaką wystąpi on w automacie Moore'a, przy czym jednakowym parom przypisujemy jednakowe nazwy. Niespełnienie tego warunku nie wprowadza błędów, jednak automat Moore'a będzie niepotrzebnie roz-

budowany. Następnie konstruujemy tablicę automatu Moore'a. Stany następne w tej tabeli dobieramy w oparciu o zmodyfikowaną tabelę automatu Mealy'ego (rys. 4.26b), biorąc jako stan aktualny wartość stanu występującą nad kreską dla określonej nazwy stanu automatu Moore'a. Litera pod kreską jest jednocześnie literą wyjściową tego stanu. Przykładowo dla stanu A automatu Moore'a (rys. 4.26c) literą wyjściową jest y, natomiast stanami następnymi stan A (4/y) oraz D(1/y). Wynika to z tabeli pokazanej na rys. 4.26b dla stanu 4, który łącznie z literą y jest reprezentowany nazwą A w automacie Moore'a. W ten sposób wypełniamy całą tabelę poszukiwanego automatu.

W oparciu o powyższą ilustrację można określić ogólny algorytm określania dla automatu Mealy'ego, podobnego automatu Moore'a. Algorytm ten będzie obejmował następujące etapy:

1) dla każdej pary: (stan następny, litera wyjściowa) automatu Mealy'ego

$$A_1 = \langle X, S_1, Y, \delta_1, \lambda_1 \rangle,$$

wprowadzamy nazwę stanu wewnętrznego automatu Moore'a

$$A_2 = \langle X, S_2, Y, \delta_2, \lambda_2 \rangle,$$

co oznaczamy symbolem

$$s_2^i = \delta(x, s_1^j) / \lambda_1(x, s_1^j),$$

przy czym

$$\lambda_2(s_2^i) = \lambda_1(x, s_1^j);$$

zbiór tych nazw tworzy alfabet wewnętrzny S_2 - poszukiwanego automatu Moore'a,

2) korzystając z układu wierszy automatu Mealy'ego, podobny automat Moore'a tworzymy w następujący sposób:

- wybieramy wiersz odpowiadający stanowi następnemu z daną nazwą stanu wewnętrznego automatu Moore'a,
- dla tego wiersza stany następne określamy wg tablicy Mealy'ego, stosując nazwy stanów automatu Moore'a wyznaczone w etapie 1,

3) minimalizujemy liczbę stanów otrzymanego automatu i ewentualnie porządkujemy nazwy.

Przykład 4.12. Określić automat typu Moore'a podobny do automatu Mealy'ego opisanego tabelą pokazaną na rysunku 4.27.

a)							
		a	b		a	b	
1		3	4		y	z	
2		3	1		z	-	
3		2	3		z	z	
4		4	-		z	y	

b)							
		a			b		
1		3/y	A		4/z	B	
2		3/z	C		1/-	D	
3		2/z	E		3/z	C	
4		4/z	B		-/y	F	

Rys. 4.27. Tablice przejść - wyjść automatu Mealy'ego z przykładu 4.12 (a,b)

a)							
		a	b				
A		E	C		y		
B		B	F		z		
C		E	C		z		
D		A	B		-		
E		C	D		z		
F		-	-		y		

b)							
B		⊗					
C		⊗	⊗ BF CF				
D		⊗ AE BC	⊗ AB BF	⊗ AE BC			
E		⊗	⊗ BC BF	⊗ CE ED	⊗ AC BD		
F		V	⊗	⊗	V	⊗	
		A	B	C	D	E	

Rys. 4.28. Tablica trójkątna oraz tablica przejść - wyjść automatu Moore'a określonego w przykładzie 4.12 (a,b)

Na zmodyfikowanej tablicy przejść - wyjść automatu Mealy'ego pokazanej na rysunku 4.27b, można wyróżnić sześć różnych par $\delta(x, s)/\lambda(x, s)$, przy czym cztery z nich są w pełni określone, jedna ma nieokreśloną literę wyjściową, druga - stan następny. Oznaczmy te pary literami od A do F. Tablicę przejść - wyjść dla tak określonego automatu Moore'a przedstawiono na rysunku 4.28a. Rodzina $\{Q_{\max}\}$ dla tego automatu ma postać

$$\{Q_{\max}\} = \left\{ \{A, F\}, \{D, F\}, \{B\}, \{C\}, \{E\} \right\}.$$

	a	b	
$p_1(A,F)$	p_5	p_4	y
$p_2(D,F)$	p_1	p_3	y
$p_3(B)$	p_3	$p_1 \text{ lub } p_2$	z
$p_4(C)$	p_5	p_4	z
$p_5(E)$	p_4	p_2	z

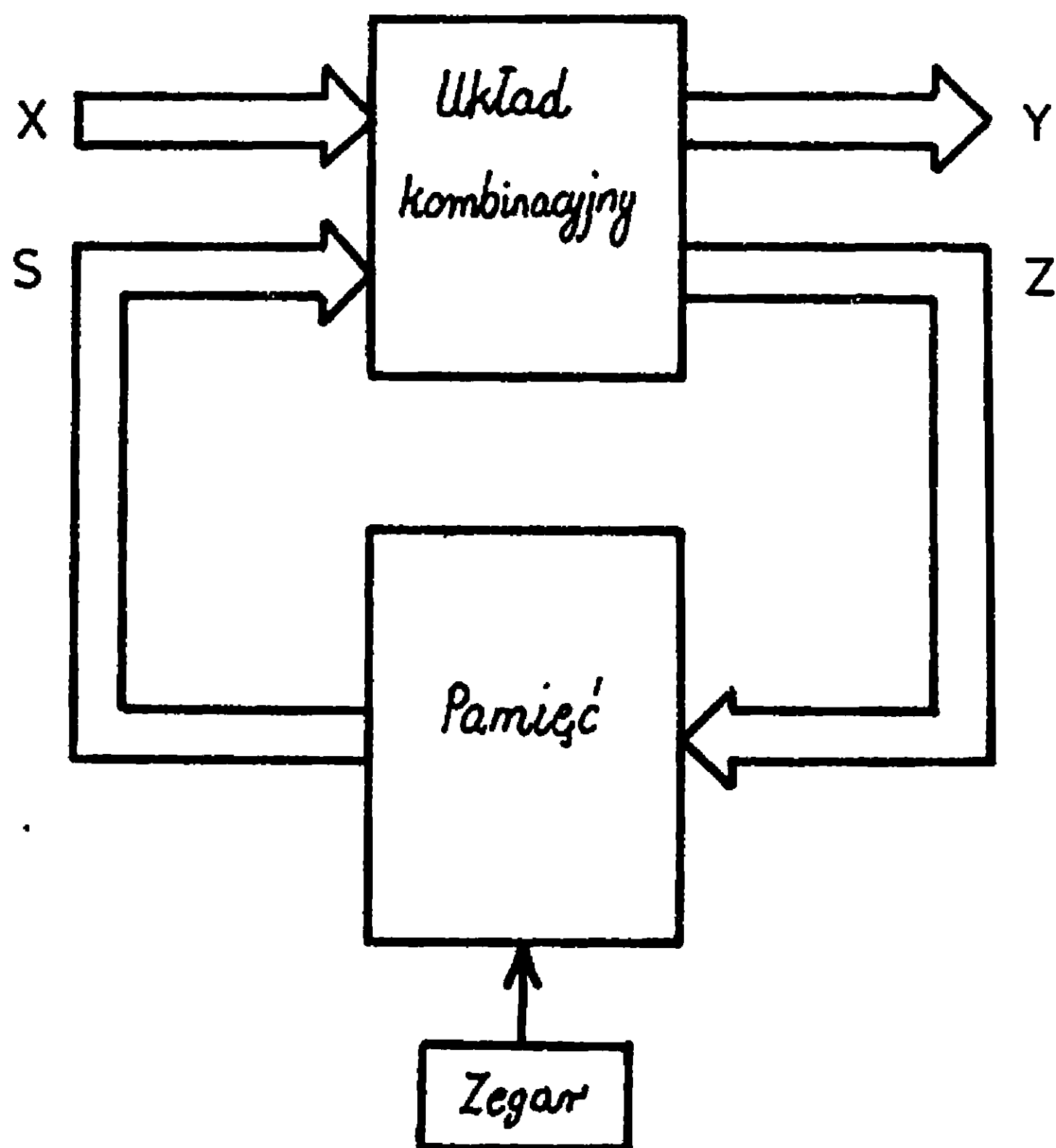
Rys. 4.29. Tablice przejść -
wyjść automatu Mealy'ego z
przykładu 4.12 (a,b)

Odpowiadający jej automat ilorazowy (minimalny) przedstawiono na rysunku 4.29. Automat ten posiada pięć stanów wewnętrznych, a więc o jeden więcej niż podobny do niego automat Mealy'ego. Stany następne odpowiadające stanom p_1 i p_4 są identyczne dla każdej litery wyjściowej. Gdyby teraz określić automat podobny Mealy'ego, to po zmianie oznaczeń otrzymamy automat wyjściowy (rys. 4.27).

5. SYNCHRONICZNE UKŁADY SEKWENCYJNE

5.1. Uwagi ogólne

Ogólny schemat sekwencyjnego układu synchronicznego przedstawiono na rysunku 5.1. Składa się on z bloku kombinacyjnego, bloku pamięci oraz z zegara. Wejściami bloku kombinacyjnego są wejścia zewnętrzne reprezentowane przez wektor X o m składowych binarnych oraz wyjścia z bloku pamięci reprezentowane przez wektor S o k składowych. Część wyjść bloku kombinacyjnego przekazuje informację na zewnątrz - reprezentuje je wektor Y o n składowych. Pozostałe wyjścia są połączone z wejściami bloku pamięci. Reprezentuje je wektor Z o l składowych.



Rys. 5.1. Schemat blokowy synchronicznego sekwencyjnego układu logicznego

Zmiany wartości wektora S są możliwe tylko w dyskretnych momentach czasowych narzuconych przez urządzenie zwane zegarem synchronizującym. Blok kombinacyjny jest układem opisanym przez zespół $(l + n)$ funkcji boolowskich $(m + k)$ zmiennych. Blok pamięci tworzą elementarne struktury zwane przerzutnikami synchronicznymi. Przerzutnik może przebywać w jednym z dwu stanów binarnych dowolnie długo, przy czym zmiana stanów w ramach cyklu pracy odbywa się wyłącznie w takt zmian impulsów zegara synchronizującego i jest uwarunkowana wartościami logicznymi występującymi na jego wejściach przed wystąpieniem tego impulsu. Na wejścia układu kombinacyjnego są podawane sygnały z wyjść k przerzutników (wektor S) oraz n -sygnałów zewnątrznych (wektor X). Metody analizy i syntezy projektowej bloku zostały omówione w rozdziale 3. Pozostają do omówienia zagadnienia związane z realizacją wektora stanów S reprezentującego k wyjść przerzutników tworzących blok pamięci oraz sformułowanie warunków sterowania wejść tego bloku przez wyjścia układu kombinacyjnego tworzące wektor Z i l składowych binarnych.

5.2. Przerzutniki

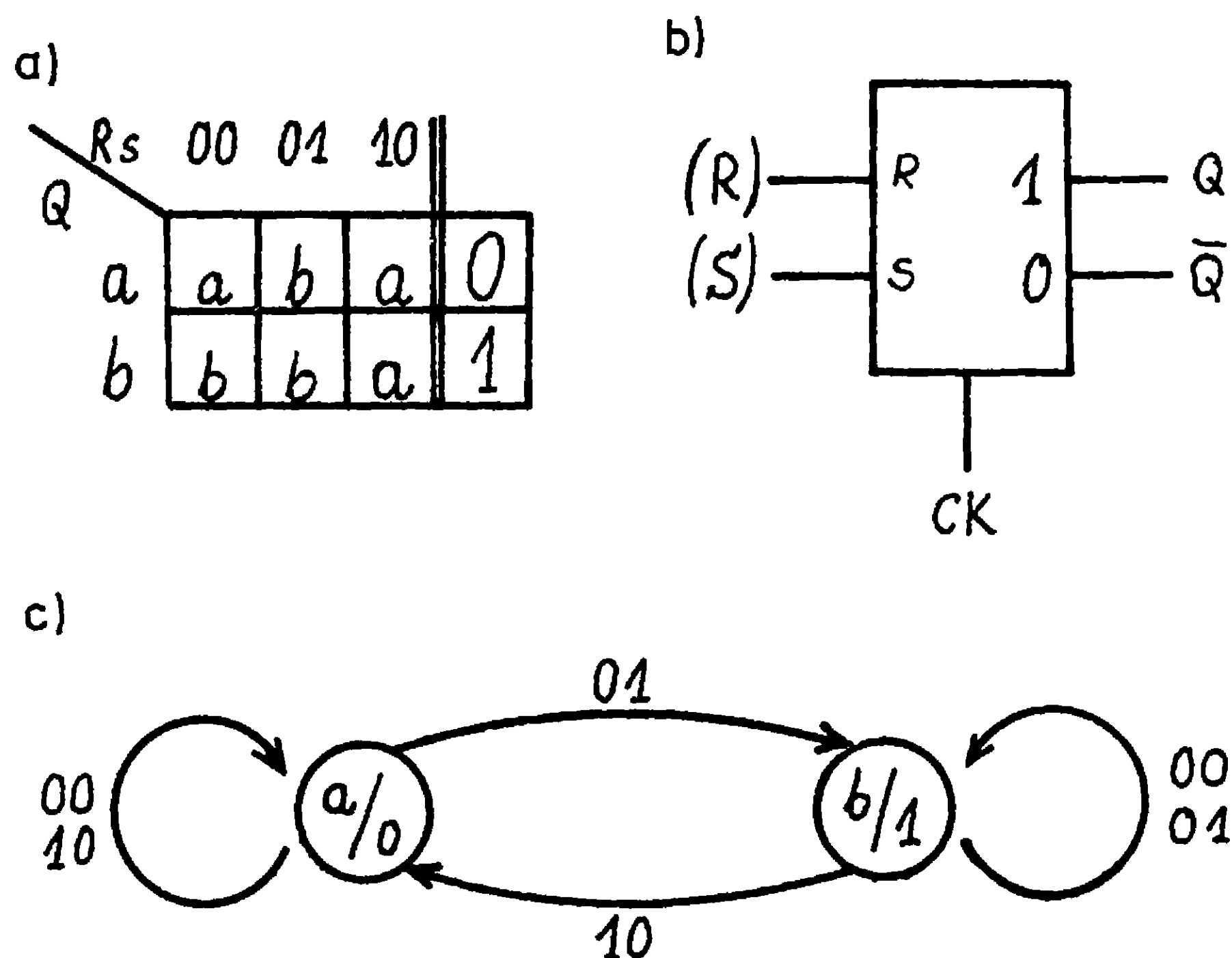
Układowe właściwości przerzutnika zostały przedstawione w poprzednim punkcie, obecnie zostanie podana jego bardziej ścisła definicja.

Definicja 5.1. Przerzutnikiem nazywamy elementarny pamięciowy układ logiczny będący automatem Moore'a o dwóch stanach wewnętrznych, przy czym litery wyjściowe związane z tymi stanami są najczęściej oznaczane symbolami 0 i 1. Alfabet wejściowy jest przedstawiony w postaci ciągu wartości wektora wejściowego o składowych binarnych, a funkcje przejść i wyjść decydują o nazwie przerzutnika.

Powyższa definicja nie precyzuje bliżej czy mamy do czynienia z realizacją przerzutnika synchroniczną czy też nie. W dalszej części punktu zostaną omówione podstawowe struktury przerzutników synchronicznych, mające zastosowanie w systemach cyfrowych. Omawiane układy nie są jedynymi możliwymi rozwiązaniami, gdyż zgodnie z definicją 5.1 liczba typów nie jest ograniczona. Są to więc rozwiązania bazowe, najczęściej spotykane. Nazwy poszczególnych typów pochodzą od nazw wejść i roli jaką odgrywa dany przerzutnik w układzie sekwencyjnym.

5.2.1. Przerzutnik typu RS

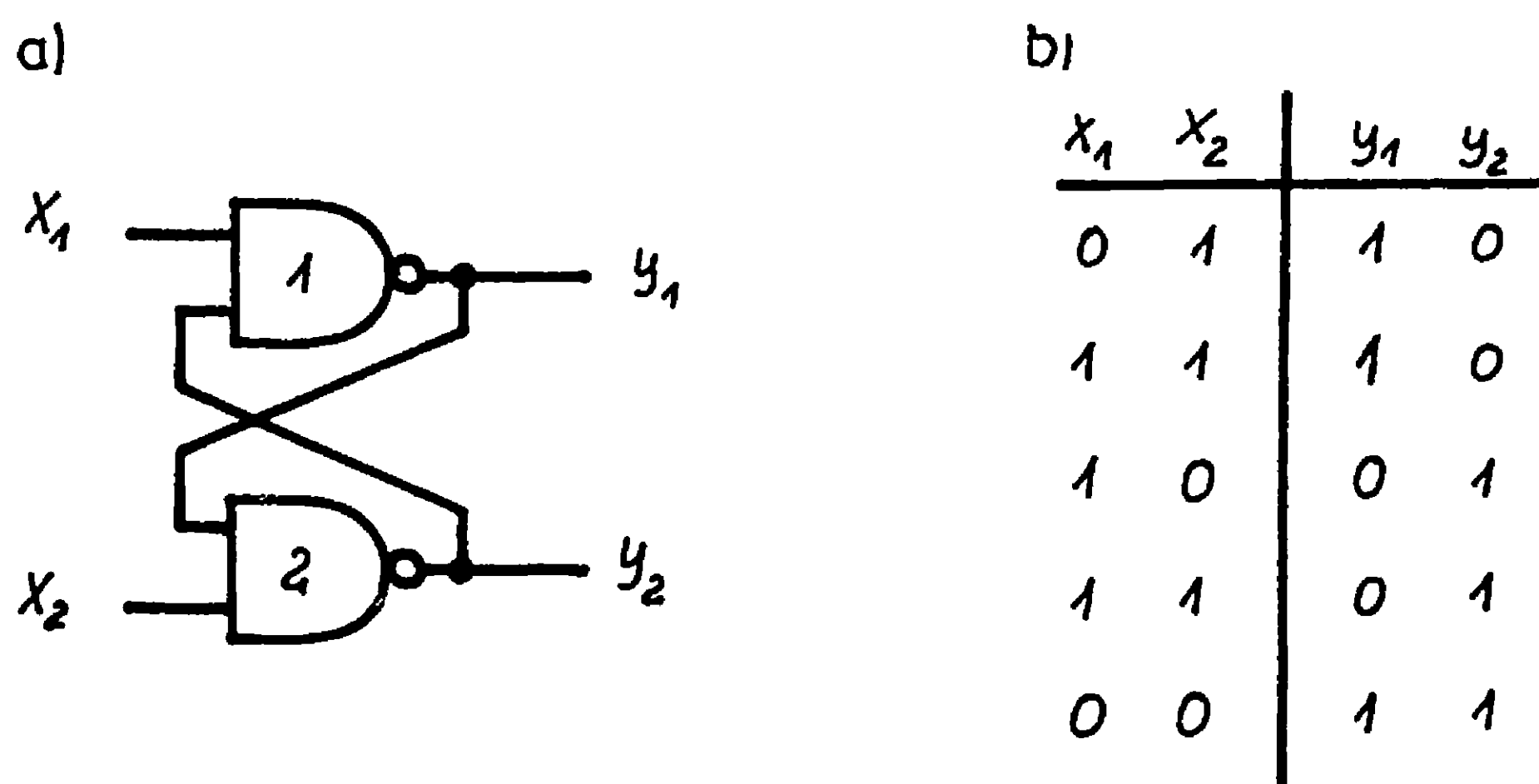
Tabele przejść - wyjść, symbol oraz graf przerzutnika RS przedstawiono na rysunku 5.2. Nazwy wejść pochodzą od reakcji wyjścia przerzutnika na sygnał logiczny 1 podany na to wejście. Wejście S - ustawia zawsze na wyjściu wartość 1 (ang. Set), natomiast wejście R zeruje wyjście (ang. Reset). Symbol graficzny przerzutnika jest przedstawiony najczęściej w postaci prostokąta z zaznaczonymi wejściami, wyjściem Q i jego dopełnieniem $\bar{Q}$ oznaczonym symbolami odpowiednio 1 i 0. Ponadto, dla podkreślenia że mamy do czynienia z przerzutnikiem synchronicznym zaznaczono wejście zegarowe CK (ang. Clock).



Rys. 5.2. Tablica przejść - wyjść (a), symbol (b) oraz graf (c) przerzutnika typu RS

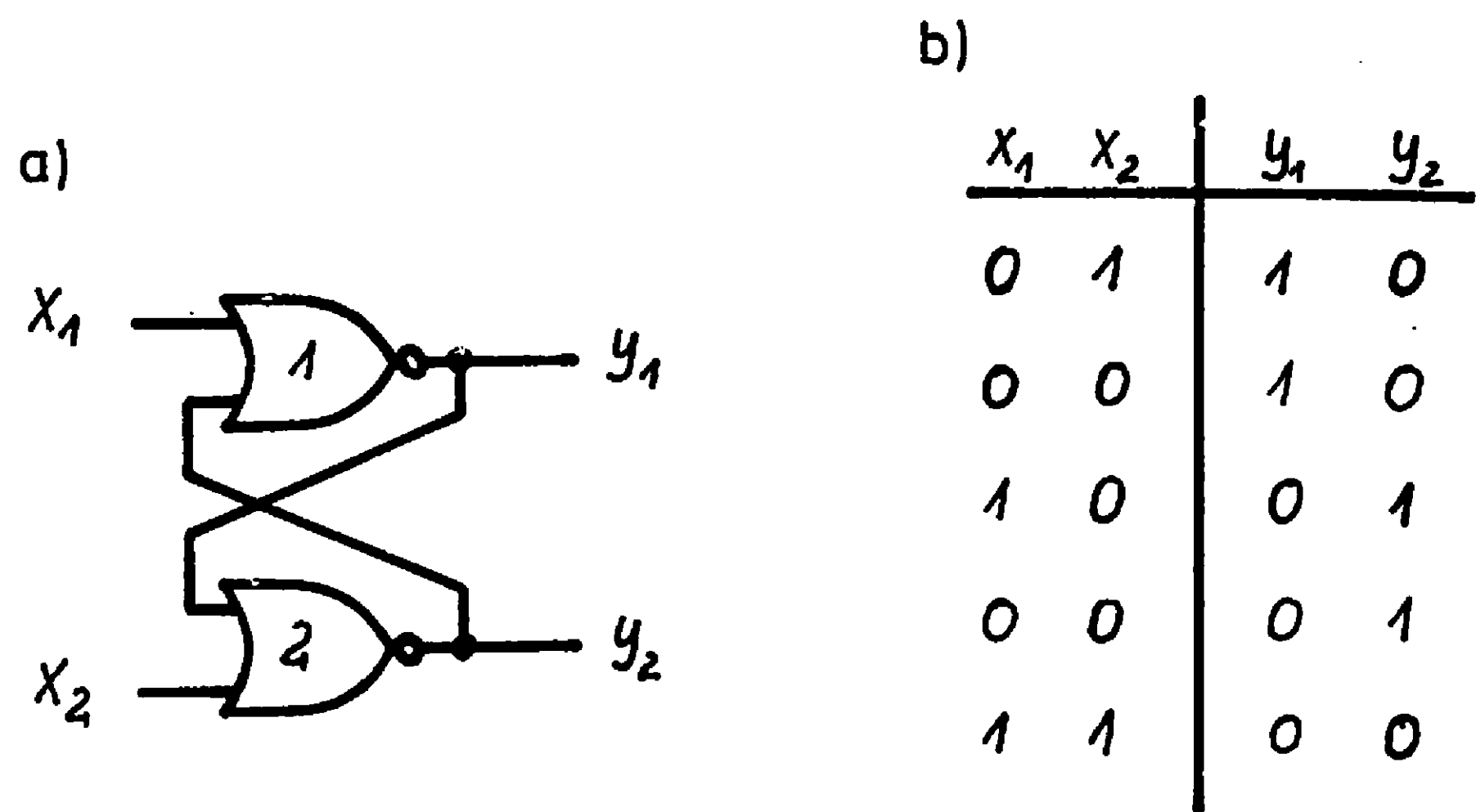
Realizacja układowa przerzutników bazuje najczęściej na bramkach logicznych ze sprzężeniami zwrotnymi, a więc połączeniami między wyjściami i wejściami bramek. Na rysunku 5.3 przedstawiono elementarny układ pamięciowy zrealizowany na bramkach NAND oraz związaną z tym układem tabelę wartości logicznych. Dla stanu wejść $x_1 = 0$, $x_2 = 1$ na wyjściu y_1 jest wymuszona 1. Wartość ta poprzez sprzężenie zwrotne jest podawana na wejście bramki nr 2. Ponieważ wejście $x_2 = 1$, więc na wyjściu bramki 2 pojawi się wartość $y_2 = 0$. Z kolei ta wartość jest

podawana na wejście bramki 1, podtrzymując na jej wyjściu stan $y_1 = 1$. W tej sytuacji zmiana sygnału x_1 z 0 na 1, przy nie zmienionej wartości x_2 ($x_2 = 1$), nie spowoduje zmian na wyjściach y_1 i y_2 . Jeśli teraz na wejście x_2 podać sygnał logiczny 0, tzn. $x_1 = 1$, $x_2 = 0$, to na wyjściu bramki drugiej zostanie wymuszona wartość $y_2 = 1$. Z uwagi na sprzężenie tego wyjścia z wejściem bramki 1, sygnał na wyjściu bramki 1 osiągnie wartość $y_1 = 0$, co również poprzez drugie sprzężenie podtrzymuje stan 1 na wyjściu bramki 2. Zmiana sygnału na wejściu x_2 z 0 na 1, tzn. $x_1 = 1$, $x_2 = 1$ nie spowoduje zmiany na wyjściach obu bramek z uwagi na samopodtrzymanie poprzez sprzężenie zwrotne. Obserwujemy więc sytuację niespotykaną dotychczas w układach kombinacyjnych: tej samej wartości wektora wejściowego $x_1 = 1$, $x_2 = 1$ odpowiadają dwie wartości wektora wyjściowego: $y_1 = 1$, $y_2 = 0$ oraz $y_1 = 0$, $y_2 = 1$, zależnie od tego czy uprzednio wektor wejściowy miał wartość 0 1 czy też 1 0. Dla $x_1 = 1$, $x_2 = 1$ układ pamięta stan odpowiadający poprzedniej wartości wektora. Jest on elementarnym dwubramkowym układem przerzutnikowym (patrz def. 5.1).

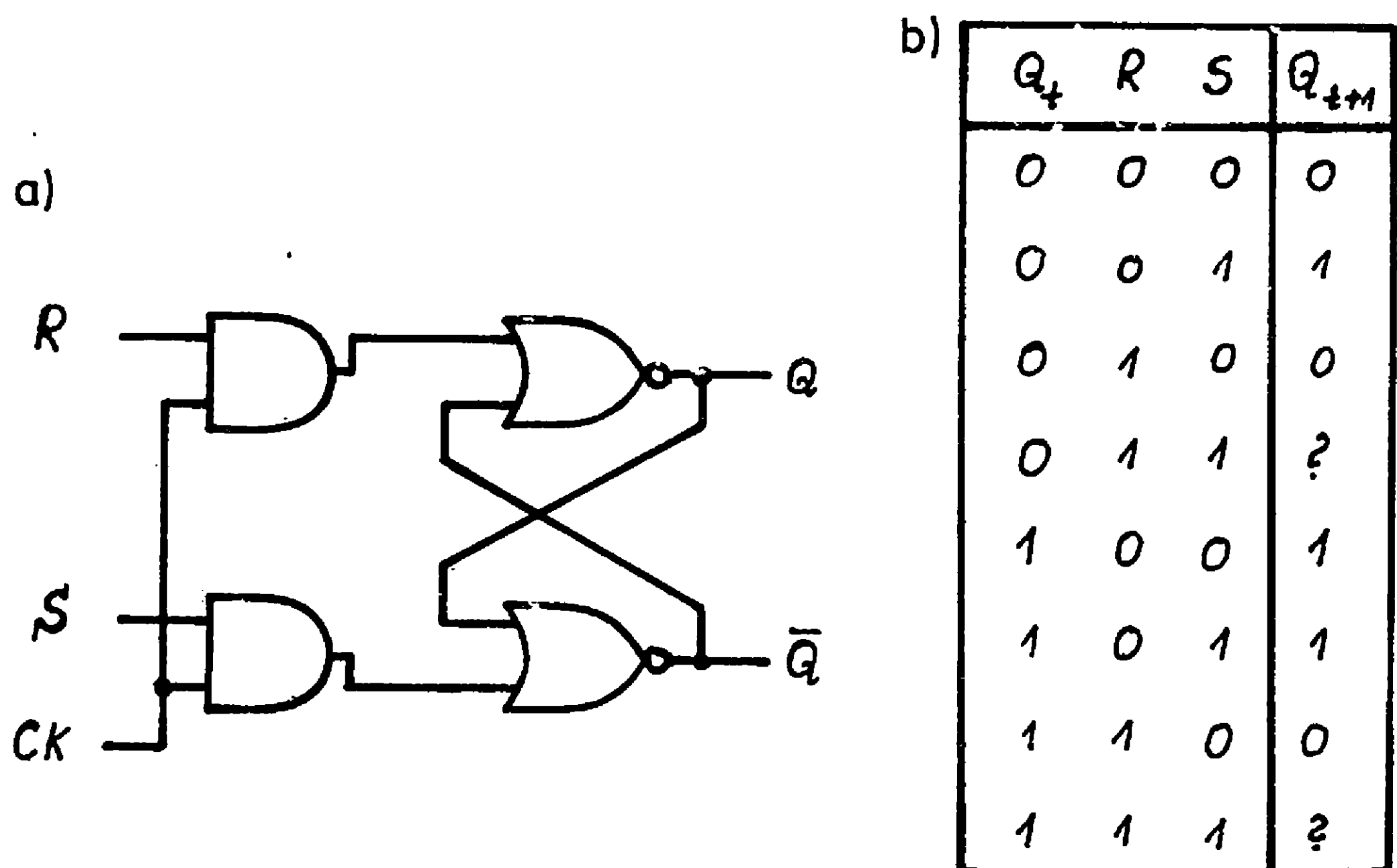


Rys. 5.3. Połączenie bramek tworzących elementarną komórkę pamięciową NAND (a,b)

W analogiczny sposób działa układ dwóch bramek NOR ze sprzężeniami zwrotnymi pokazany na rysunku 5.4. Tutaj sygnałem aktywnym jest sygnał 1. Zawsze wymusza on na wyjściu logiczne zero. Wartościom 00 wektora wejściowego odpowiadają dwie wartości wektora wyjściowego: 10, 01, zależnie od stanu poprzedniego. Efekt pamiętania dla tych bramek występuje więc dla wartości wektora wejściowego wynoszącej $x_1 = 0$, $x_2 = 0$.



Rys. 5.4. Połączenie bramek tworzących elementarną komórkę pamięciową NOR (a,b)



Rys. 5.5. Schemat synchronicznego przerzutnika RS z bramkami NOR (a,b)

Właściwości układów przedstawionych na rysunkach 5.3 i 5.4 można wykorzystać w różny sposób. Na rysunku 5.5 przedstawiono układ wykorzystujący jako blok pamięciowy bramki NOR. Właściwości przerzutnika w sposób kompleksowy opisuje tabela charakterystyczna pokazana na rysunku 5.5b. Bramki koniunkcji przekazują sygnały z wejść R i S na

wejścia bramek NOR tylko w momencie, gdy na wejściu zegarowym jest sygnał 1. Okres, dla którego $CK = 1$ jest okresem aktywizacji wejść R i S. Musi on być na tyle długi, by sygnał fizyczny na wejściach R i S wywołał reakcję na wyjściach przerzutnika. Jednocześnie nie powinien on być dłuższy, gdyż zmniejszyło by to odporność przerzutnika na ewentualne zakłócenia i przekreśliło zalety synchronizacji. W okresach, gdy $CK = 0$ przerzutnik jest niewrażliwy na zmiany sygnału wejść R i S. Czasy propagacji bramek są na tyle krótkie, że sygnał $CK = 1$ ma charakter krótkiego impulsu o czasie trwania zwykle pomijalnie małym w porównaniu z odstępem cyklu generacyjnego zegara. Oznaczając przez Q_t stan aktualny wyjścia Q przerzutnika, w oparciu o znajomość sygnałów wejściowych R i S i tabelę pokazaną na rysunku 5.4b, można przewidzieć stan następny Q_{t+1} , jaki ustali się na wyjściu Q po przejściu kolejnego impulsu zegarowego. Wynik przedstawia tabela pokazana na rysunku 5.5b. Jak już wspomniano nosi ona nazwę tabeli charakterystycznej przerzutnika RS. Dla wartości $R = 1$ i $S = 1$, stan następny przerzutnika jest nieokreślony. Wynika to bezpośrednio z tabeli pokazanej na rys. 5.4b, gdyż jeśli $x_1 = 1$ i $x_2 = 1$, to stany obu wyjść $y_1 = 0$ i $y_2 = 0$, a więc po impulsie $CK = 1$ wynik na wyjściu Q będzie zależał wyłącznie od różnicy w szybkości propagacji sygnału przez poszczególne bramki. Występowanie logicznej jedynki jednocześnie na obu wejściach przerzutnika RS jest więc niedopuszczalne z punktu widzenia poprawnej pracy układu.

RS		00	01	11	00
Q.	0	0	1	X	0
	1	1	1	X	0

Rys. 5.6. Tablica Karnaugh dla przerzutnika typu RS

W oparciu o tabelę charakterystyczną pokazaną na rysunku 5.5b, można określić równanie charakterystyczne przerzutnika RS, traktując zmienne Q_t , R, S jako zmienne wejściowe. Na rysunku 5.6 przedstawiono tabelę Karnaugh z zaznaczeniem stanów nieokreślonych dla $R = 1$ i $S = 1$. Tablica ta utożsamia się z tabelą przejść - wyjść pokazaną na rysunku 5.2a, jeśli przyjąć oznaczenia stanów wewnętrznych $a = 0$, $b = 1$ i do-

dać kolumnę stanów nieokreślonych dla wektora $R = 1$ i $S = 1$. Litery wyjściowe utożsamia się wtedy ze stanami wewnętrznymi, co nie wymaga odrębnego zaznaczania w grafie i w tablicy przejść - wyjść. Określone na podstawie tablicy równanie charakterystyczne będzie mieć postać

$$Q_{t+1} = S + Q_t \bar{R},$$

przy warunku dodatkowym $R \cdot S = 0$.

Obok tablicy charakterystycznej i równania charakterystycznego, wygodne jest określenie tzw. tablicy wzbudzeń przerzutnika pokazanej na rysunku 5.7a. Formuluje ona warunki jakie muszą spełnić sygnały na wejściach R , S aby nastąpiła określona zmiana wartości logicznej na wyjściu przerzutnika. Uzyskuje się ją bezpośrednio z tablicy charakterystycznej, którą w wersji skróconej przedstawiono na rysunku 5.7b.

a)

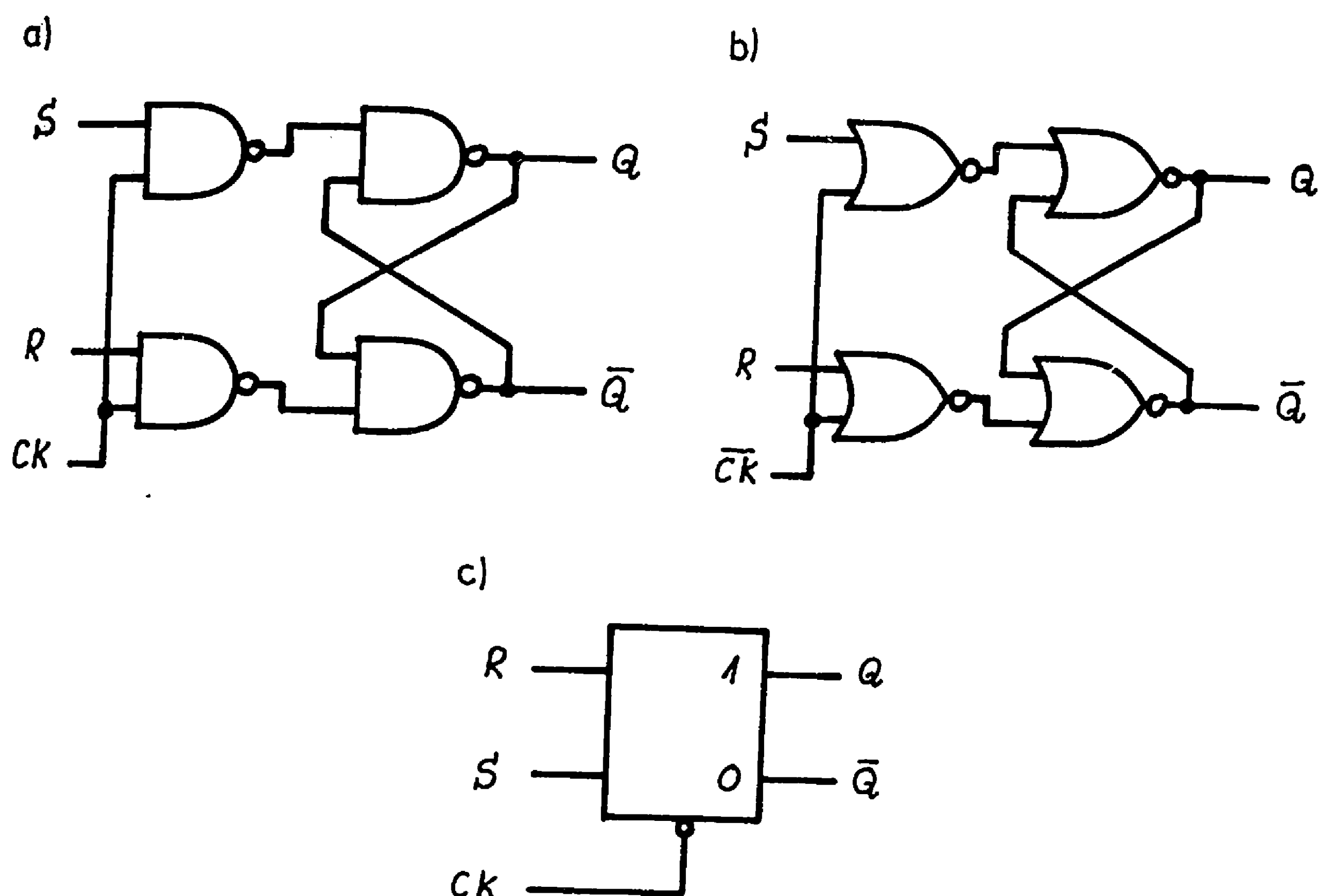
Q_t	Q_{t+1}	R	S
0	0	X	0
0	1	0	1
1	0	1	0
1	1	0	X

b)

R	S	Q_{t+1}
0	0	Q_t
0	1	1
1	0	0
1	1	?

Rys. 5.7. Tablica wzbudzeń (a) i skrócona tablica charakterystyczna przerzutnika RS (b)

Oprócz schematu synchronicznego przerzutnika RS pokazanego na rysunku 5.5 można podać jego inne realizacje bramkowe. Przykładowo na rysunku 5.8a podano układ zbudowany z bramek NAND. Łatwo sprawdzić pamiętając o tym, że sygnał 0 zawsze wymusza 1 na wyjściu bramki NAND, iż działanie tego układu jest takie same jak układu pokazanego na rysunku 5.5. Identyczne właściwości ma również układ dualny zbudowany z bramek NOR i pokazany na rysunku 5.8b. Jednak tutaj impuls zegarowy musi mieć postać "krótkiego zera", które w tym przypadku aktywizuje wejścia R i S . zaznaczono to symbolem dopełnienia nad nazwą wejścia zegarowego CK. Na schematach blokowych wejścia takie najczęściej oznacza się dodatkowym kółkiem (rys. 5.8b).



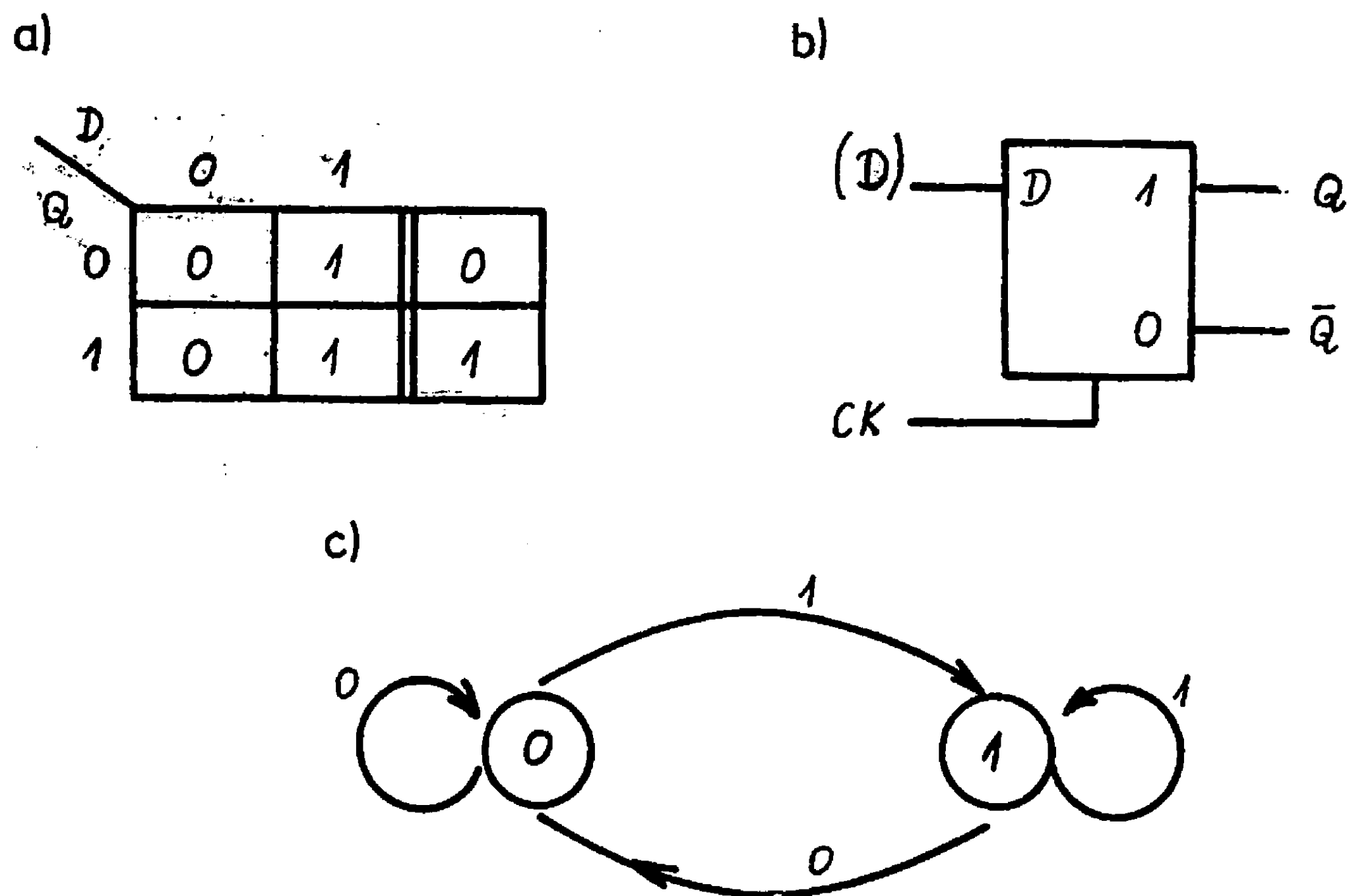
Rys. 5.8. Schematy synchronicznego przerzutnika RS zrealizowanego z bramek NAND oraz NOR (a,b)

5.2.2. Przerzutnik typu D

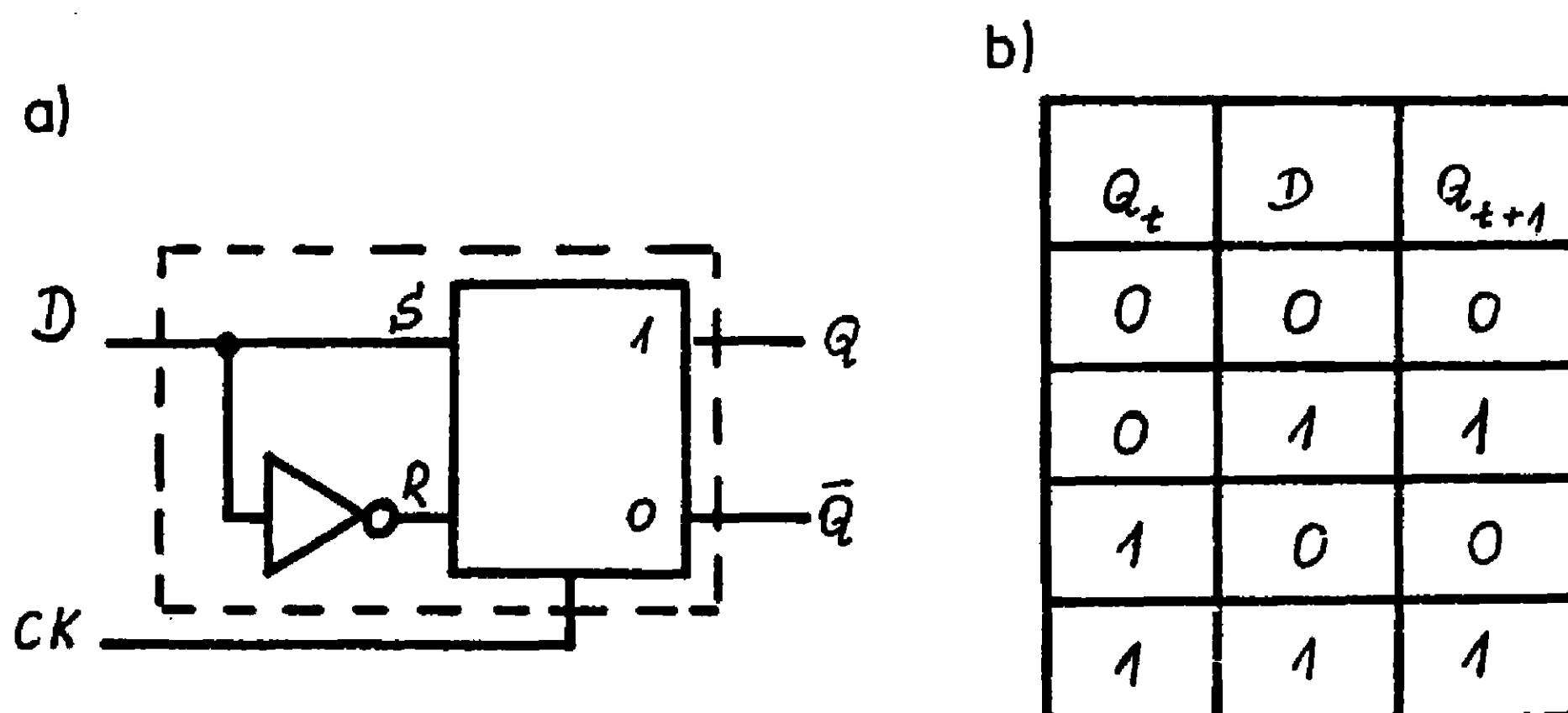
W systemach cyfrowych często zachodzi potrzeba przesyłania informacji binarnych na wyjścia danego bloku synchronicznego z impulsem zegarowym. Przed tym impulsem informacja jest podana na wejścia i po impulsie w niezmienionej postaci pojawi się na wyjściach. Wspomniany blok tworzą elementarne układy przerzutnikowe zwane przerzutnikami typu D (ang. Data). Na rysunku 5.9 przedstawiono tabelę przejść - wyjść, symbol oraz graf przerzutnika.

Z uwagi na identyczne symbole stanów wewnętrznych i liter wyjściowych nie ma potrzeby osobnego wpisywania do grafu liter wyjściowych związanych ze stanami. Przerzutnik ten można w prosty sposób otrzymać z przerzutnika RS, podając na wejście S sygnał D, natomiast na wejście R sygnał $\bar{D}$ (rys. 5.10). Z tablicy pokazanej na rysunku 5.5b wynika wówczas bezpośrednio tablica charakterystyczna pokazana na rysunku 5.10b. Jest ona zgodna z podanymi uprzednio zadaniami realizowanymi przez przerzutnik typu D. Równanie charakterystyczne jest tutaj bardzo proste

$$Q_{t+1} = D.$$



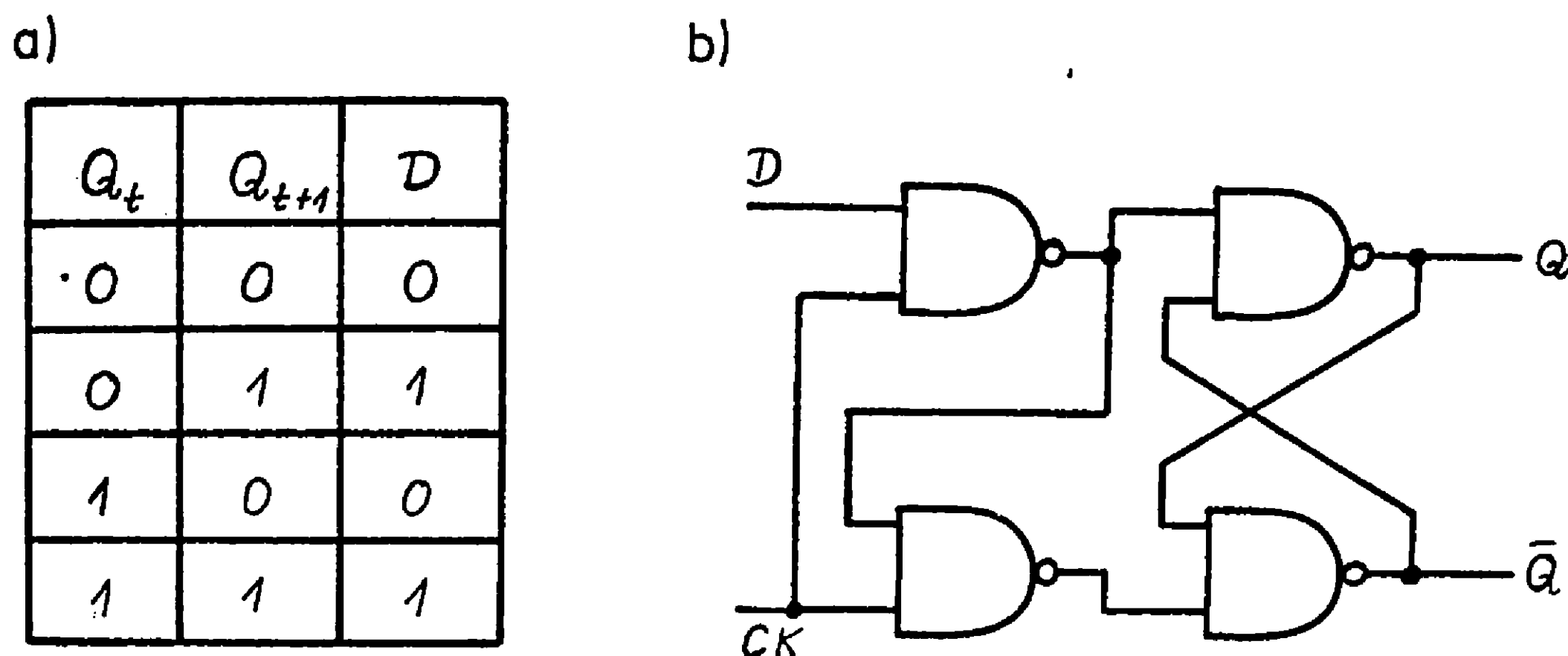
Rys. 5.9. Tablica przejść - wyjść (a), symbol (b) oraz graf (c) przerzutnika typu D



Rys. 5.10. Przerzutnik typu D otrzymany z przerzutnika RS (a) i jego tablica charakterystyczna (b)

Stan następny jest taki sam jak sygnał znajdujący się na wejściu przerzutnika przed pojawieniem się impulsu synchronizującego. Bloki pamięciowe złożone z przerzutników typu D, mogą więc być traktowane jako bloki synchronicznych opóźnień transmisji danych. Tablicę wzbu-
dzeń przedstawiono na rysunku 5.11a. Realizację bramkową przerzutnika
typu D otrzymuje się bezpośrednio ze schematów przerzutnika RS i ry-
sunku 5.10a. Dla połączeń przytoczonych w p. 5.2.1 daje to układy złoż-
żone z pięciu bramek. Uwzględniając jednak fakt, że na wyjściu bramki

NAND mamy negację iloczynu, co dla pierwszej bramki daje rezultat: $\overline{D \cdot (CK = 1)}$, można podać również realizację czterobramkową przerzutnika typu D, pokazaną na rysunku 5.11b. Układ dualny złożony z bramek NOR będzie miał tę samą postać, trzeba tylko stosować dopełnienie impulsu zegarowego.

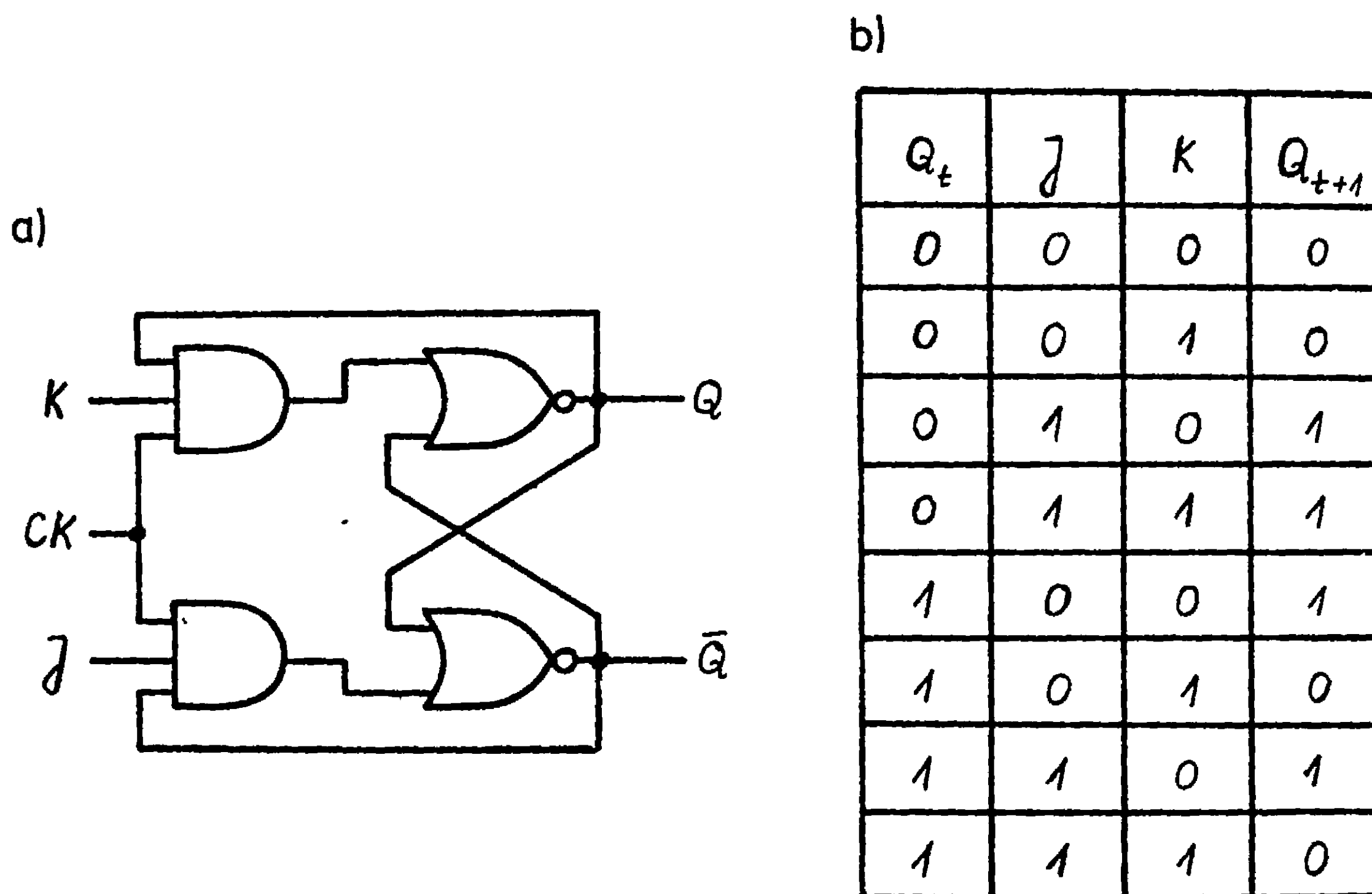


Rys. 5.11. Tablica wzbudzeń (a) oraz czterobramkowy schemat przerzutnika typu D(b)

5.2.3. Przerzutnik typu JK

Wadą przerzutnika RS jest to, że dla wartości wejść $R = 1$, $S = 1$ stan następny jest stanem nieokreślonym. Spośród czterech możliwych, wykorzystywane są więc tylko trzy wartości wektora wejściowego. Jeśli do bramek wejściowych przerzutnika RS pokazanego na rysunku 5.5a doprowadzić dodatkowo sygnał z wyjść (rys. 5.12a), to tak otrzymany nowy przerzutnik dla wartości wektora wejściowego 11 będzie realizował jako stan następny, stan przeciwny do aktualnego. Istotnie, jeśli założyć $Q_t = 0$ oraz na obu wejściach jedynki, to po pojawieniu się impulsu zegarowego $CK = 1$, dolna bramka wysterowana sygnałem $Q_t = 1$ zmieni stan bramek pamięciowych na przeciwny $Q_{t+1} = 1$, gdyż w tym samym czasie sygnał $Q_t = 0$ blokuje wyjście bramki wejściowej górnej. Z kolei, gdy przyjąć wartość sygnału $Q_t = 1$, to dla jedynek na wejściach zewnętrznych po pojawieniu się impulsu zegarowego bramka górna zmieni stan bramek pamięciowych, gdyż sygnał $Q_t = 0$ blokuje bramkę wejściową dolną. W pozostałych przypadkach praca przerzutnika jest taka sama jak przerzutnika RS pokazanego na rysunku 5.5a. Dla sygnału $Q_t = 0$ aktywna jest tylko bramka dolna i tylko jedynka na jej wejściu spowoduje zmianę stanu na $Q_{t+1} = 1$. Wejście to jest więc wejściem ustawiającym.

Przyjęto je określać symbolem "J". Dla sygnału $Q_t = 1$ aktywną jest tylko bramka górna. Jedynka na jej wejściu powoduje po przejściu impulsu zegarowego zmianę stanu na $Q_{t+1} = 0$. Wejście to pełni więc rolę wejścia kasującego. Oznacza się je symbolem "K". Sprzężenia zwrotne z wyjścia na wejścia, dla wektorów wejściowych 00, 01, 10, dublują jedynie efekt wysterowania sygnałami zewnętrznymi.



Rys. 5.12. Schemat bramkowy (a) i tablica charakterystyczna przerzutnika JK (b)

Analizując pracę układu należy zwrócić uwagę na fakt, że impuls zegarowy musi być krótszy niż czas propagacji sygnału z wyjścia na wejście. Inaczej zmiana stanu na wyjściu zostałaby podana na wejście przed końcem impulsu $CK = 1$, co doprowadzi do kolejnych zmian i w konsekwencji do drgań układu. Z tych względów schemat pokazany na rysunku 5.12a ma głównie charakter poglądowy, a realizacje użytkowe zostaną podane w czasie omawiania sposobów wyzwalania przerzutników.

Tablicę wzbudzeń przerzutnika JK pokazano na rysunku 5.13a, natomiast skróconą tablicę charakterystyczną na rysunku 5.13b. Porównując tablice wzbudzeń przerzutnika JK i przerzutnika RS (rys. 5.7a) można zauważyć, że dla przerzutnika JK stan następny jest określony jednoznacznie tylko przez jedno z wejść. Na ogół upraszcza to strukturę bloku kombinacyjnego, współpracującego z tymi przerzutnikami.

a)

Q_t	Q_{t+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

b)

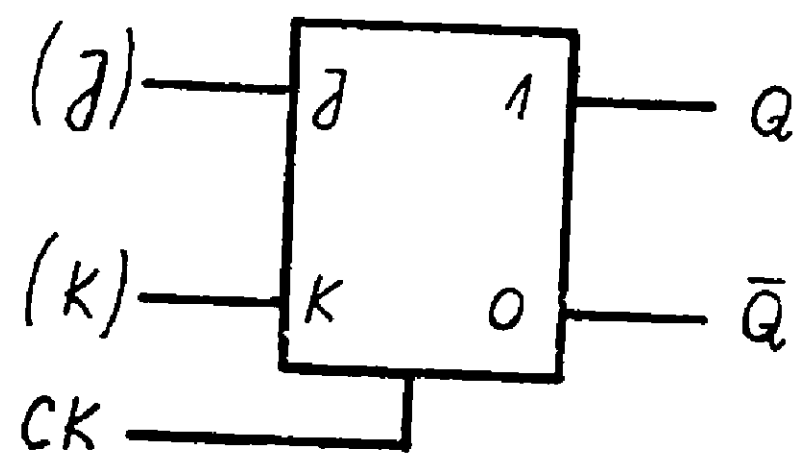
J	K	Q_{t+1}
0	0	Q_t
0	1	0
1	0	1
1	1	$\bar{Q}_t$

Rys. 5.13. Tablica wzbudzeń (a) i skrócona tablica charakterystyczna przerzutnika JK (b)

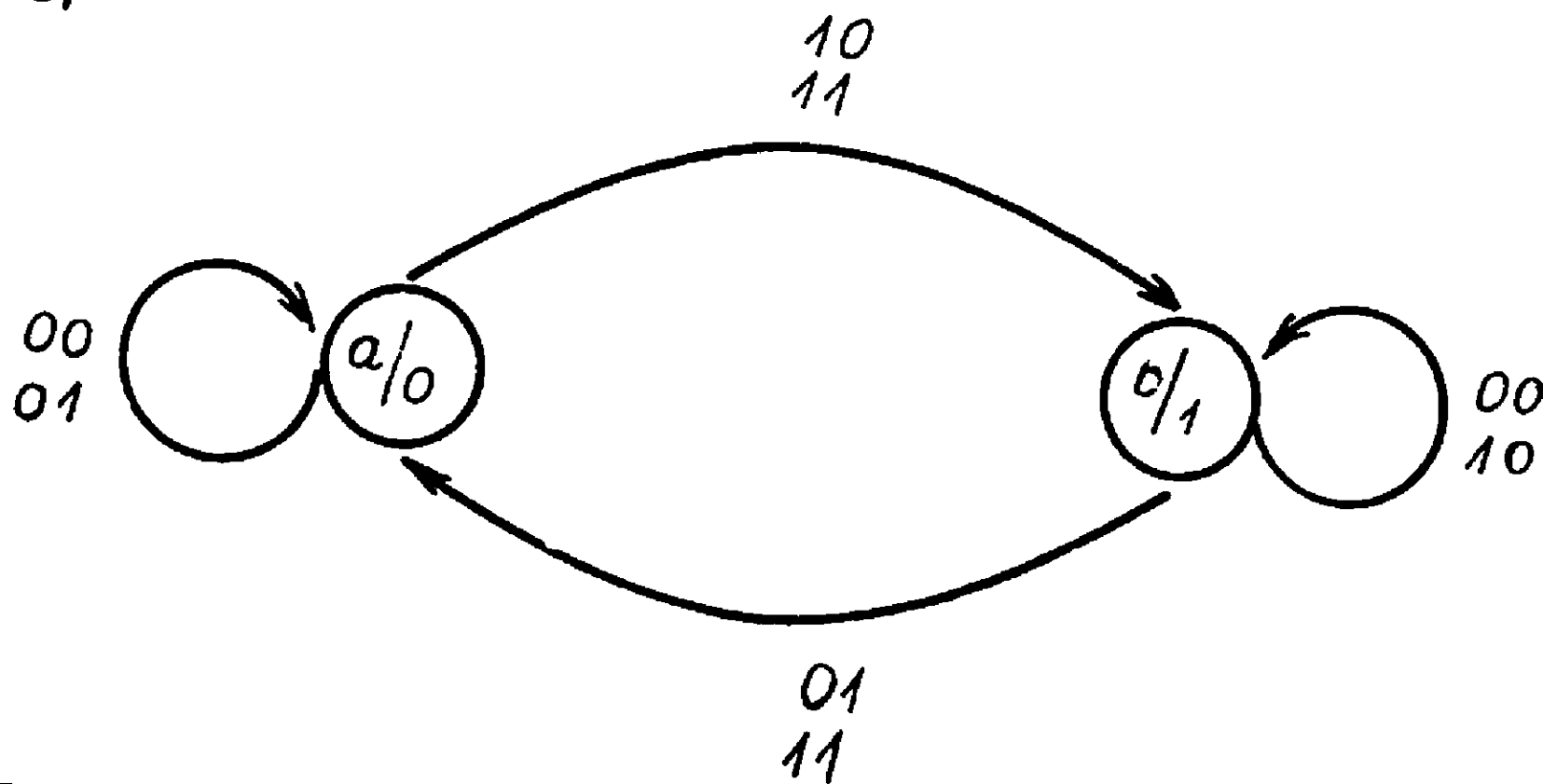
a)

J, K	00	01	10	11	
Q					
a	a	a	b	b	0
b	b	a	b	a	1

b)



c)



Rys. 5.14. Tablica przejść - wyjść (a), symbol (b) oraz graf (c) przerzutnika JK

W kategoriach wejścia i wyjścia, przerzutnik JK jest automatem typu Moore'a, który pokrywa automat będący przerzutnikiem RS. Tablicę przejść - wyjść, symbol oraz graf automatu JK pokazano na rysunku 5.14. W wersjach układowych stany wewnętrzne przyjmują te same wartości co i litery wyjściowe, tzn. $a = 0$, $b = 1$. Po zmianie kolejności kolumn i pominięciu kolu-

mny wyjściowej, tablica przejść - wyjść przekształci się wówczas w tablicę Karnaugh pokazaną na rysunku 5.15. Równanie charakterystyczne określone na podstawie tablicy ma postać

$$Q_{t+1} = Q_t \bar{K} + \bar{Q}_t J.$$

$J \backslash K$	00	01	11	10
0	0	0	1	1
1	1	0	0	1

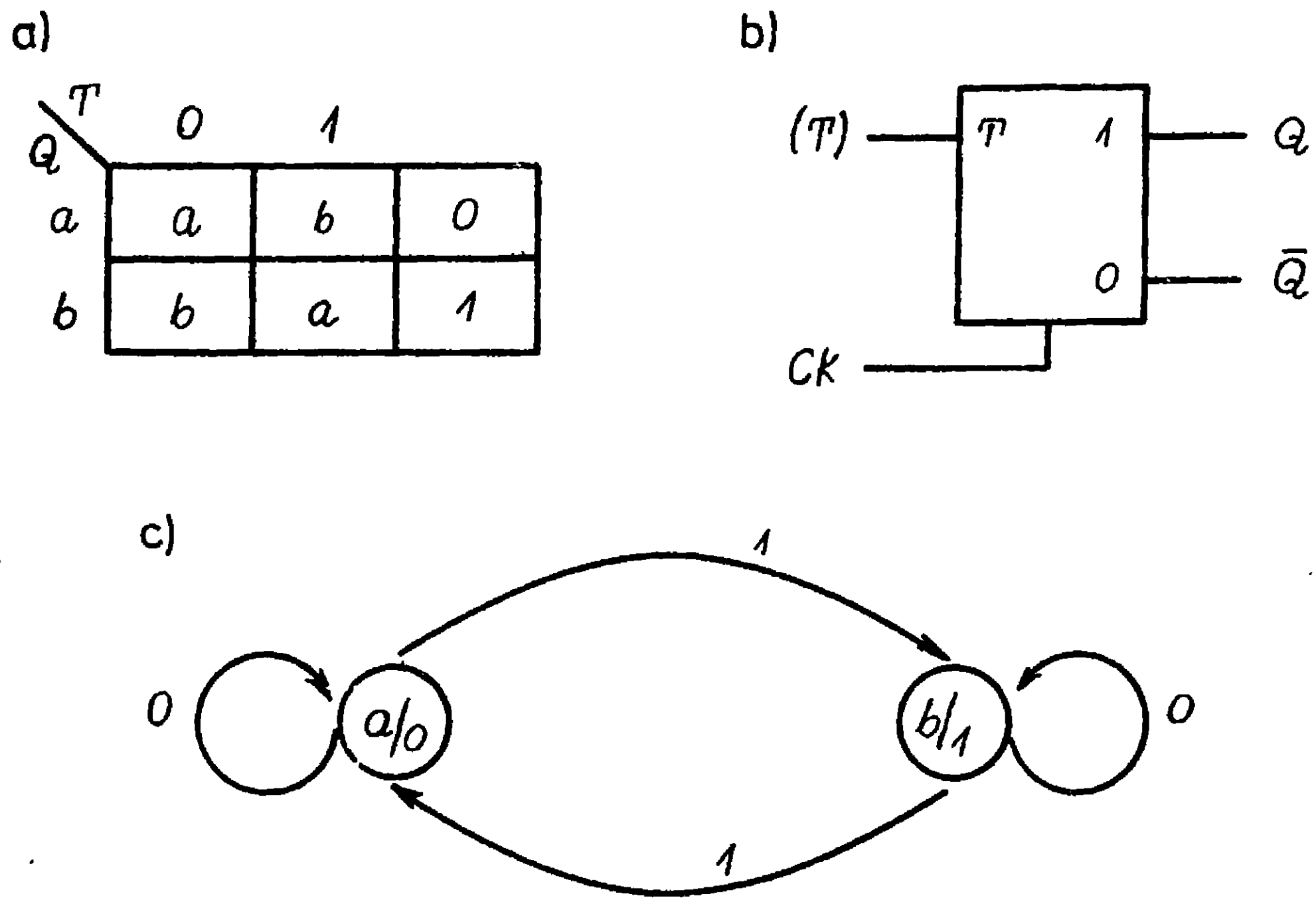
Rys. 5.15. Tablica Karnaugh przerzutnika JK

5.2.4. Przerzutnik typu T

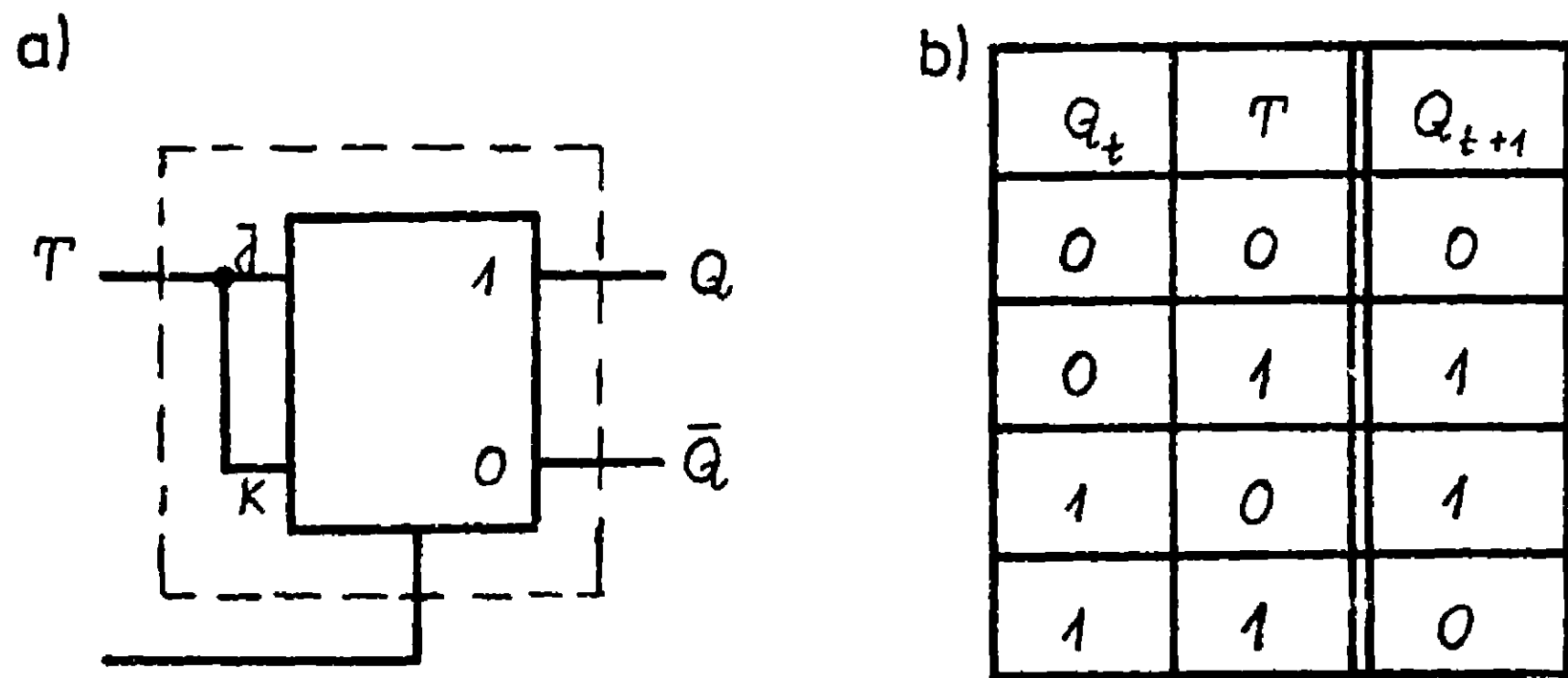
Istotną rolę w systemach cyfrowych, a w szczególności w tzw. układach licznikowych odgrywają przerzutniki typu T (ang. Toggle), które zmieniają stan po każdym impulsie zegarowym wówczas, gdy na ich wejściu jest sygnał 1, natomiast pozostają niezmiennione, gdy sygnał ten wynosi 0. Układ realizuje wówczas operację EXCLUSIVE-OR między stanem aktualnym i wejściem. Na rysunku 5.16 przedstawiono tablicę przejść - wyjść, symbol oraz graf stanów przerzutnika typu T. Realizację układową można uzyskać bardzo prosto z przerzutnika JK, łącząc równolegle oba jego wejścia (rys. 5.17a). Z tablicy pokazanej na rysunku 5.12b wynika wówczas bezpośrednio tablica charakterystyczna przerzutnika typu T pokazana na rysunku 5.17b. Tablicę wzbudzeń oraz skróconą tablicę charakterystyczną pokazano na rysunku 5.18. Podstawiając w tablicę przejść - wyjść (rys. 5.16a) oznaczenia stanów wewnętrznych takie same jak dla liter, wyjściowych otrzymujemy jej postać utożsamiającą się z tablicą Karnaugh (rys. 5.18c). Określone na podstawie tej tablicy równanie charakterystyczne ma postać

$$Q_{t+1} = \bar{Q}_t T + Q_t \bar{T} = Q_t \oplus T.$$

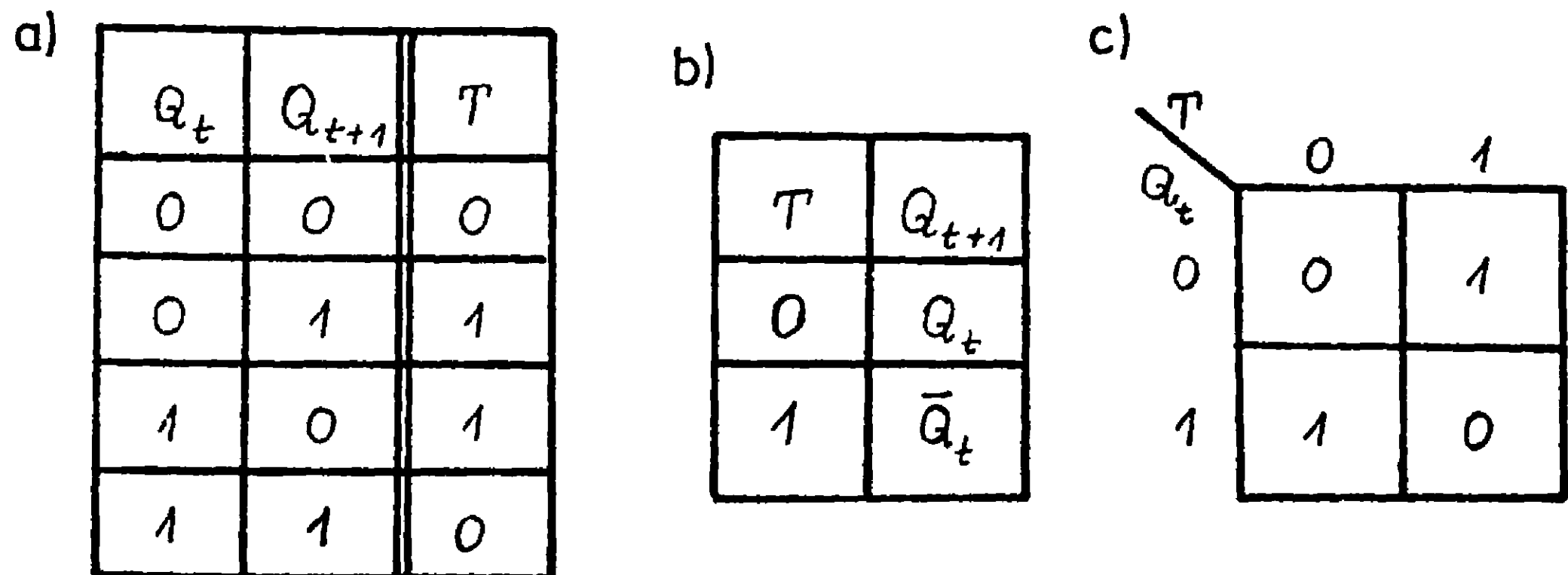
Jest to wspomniana już operacja EXCLUSIVE-OR między stanem aktualnym a wejściem. Rezultat tej operacji pojawia się dopiero po przejściu impulsu synchronizującego, nie ma więc ona charakteru takiego samego jak dla układów kombinacyjnych.



Rys. 5.16. Tablica przejść - wyjść (a), symbol (b) oraz graf (c) przerzutnika typu T



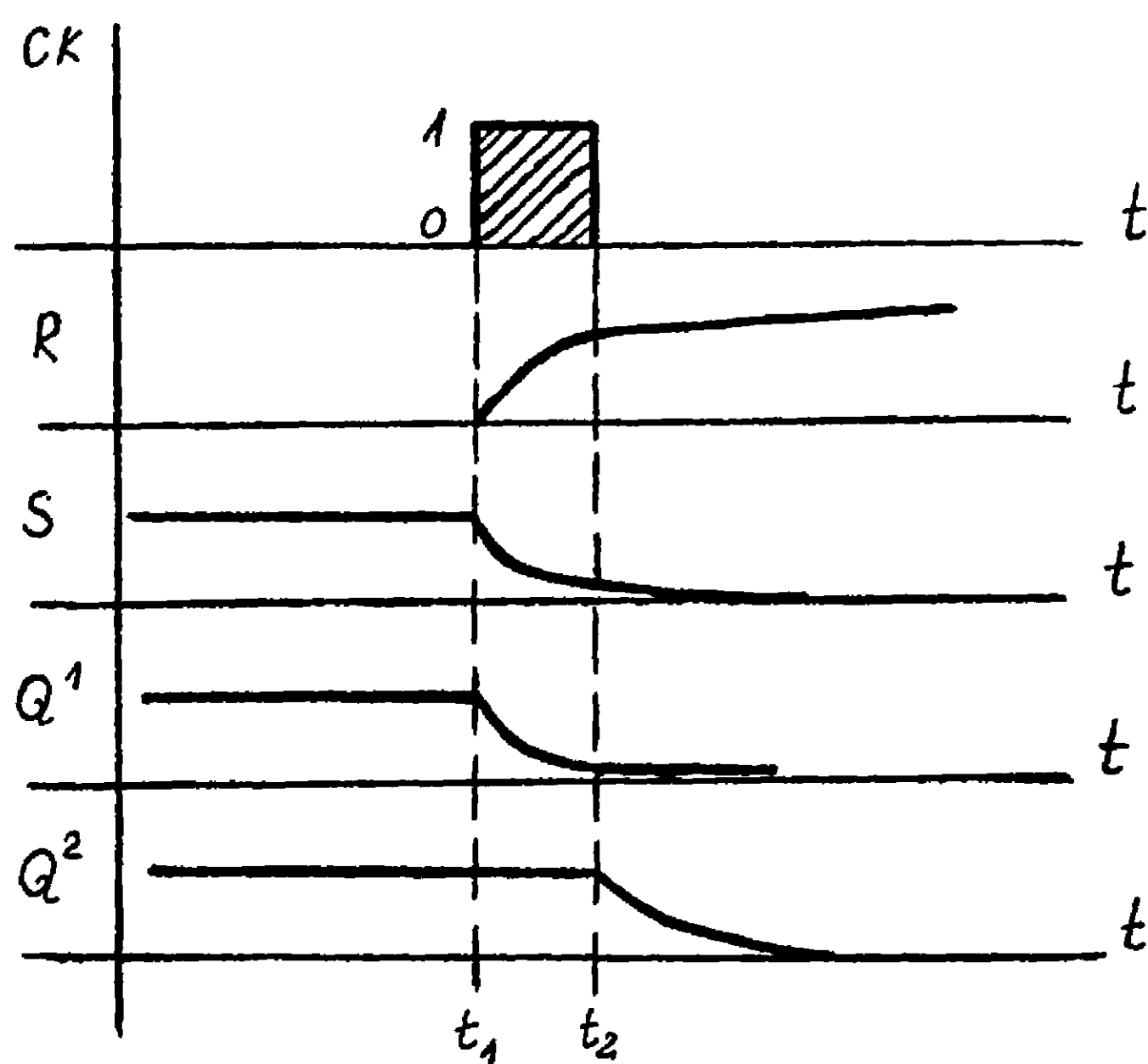
Rys. 5.17. Przerzutnik typu T otrzymany z przerzutnika JK (a) oraz jego tablica charakterystyczna (b)



Rys. 5.18. Tablica wzbudzeń (a), skrócona tablica charakterystyczna (b) oraz tablica Karnaugh przerzutnika typu T (c)

5.3. Wyzwalanie przerzutników na zboczach impulsów zegarowych

W momencie pojawienia się impulsu $CK = 1$ bramki wejściowe omawianych przerzutników stają się aktywne dla sygnałów zewnętrznych, tzn. do bramek pamięciowych mogą dochodzić sygnały wymuszające określony stan wyjścia. Ta aktywność trwa tak długo, jak długo sygnał $CK = 1$. W tym czasie przerzutnik może reagować na zmiany sygnałów wejściowych, również tych wywołanych zakłóceniami. Czas trwania impulsu synchronizującego nie powinien więc trwać dłużej niż wynika to z warunków propagacji sygnałów przez bramki. Ograniczenie to nie zawsze wystarcza, w związku z tym opracowano metody dalszej poprawy niezawodności pracy układu. Jedną z nich jest wymuszanie ewentualnych zmian na wyjściu dopiero na tylnym zboczu impulsu zegarowego, a więc w momencie kiedy on się kończy. Potem następuje stan blokady wejść i wszelkie zmiany sygnałów nie dochodzą do bramek pamięciowych. Przerzutnik zapamiętuje więc stan wynikający z wartości sygnałów tuż przed końcem impulsu zegarowego.

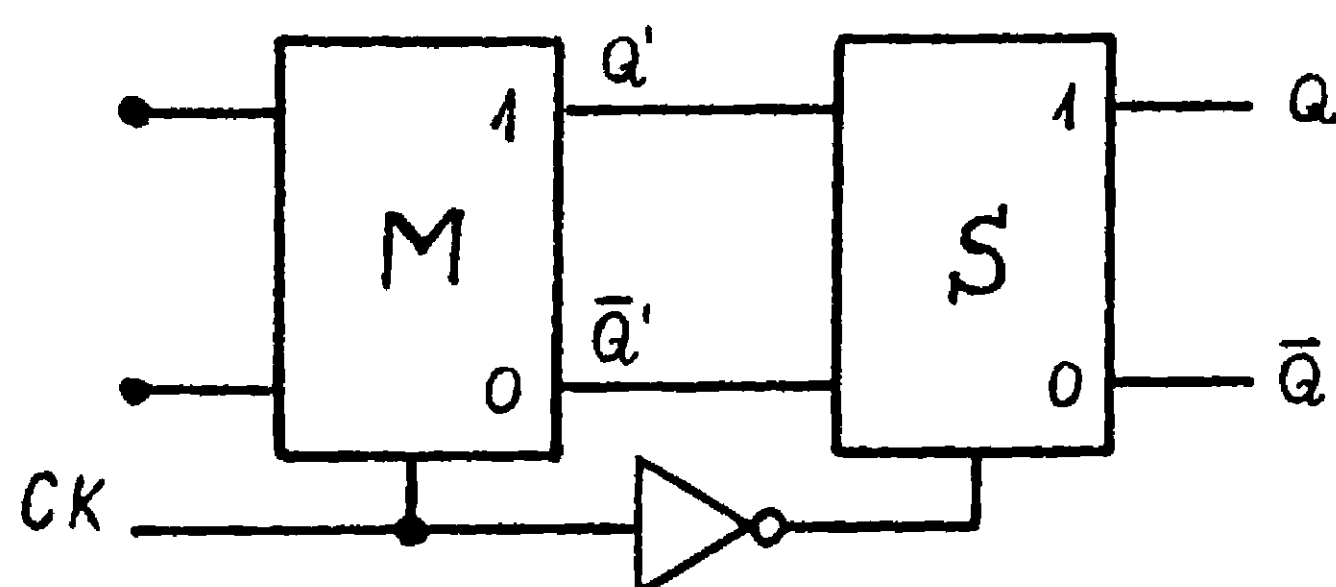


Rys. 5.19. Przebiegi czasowe zmian sygnałów na wejściach i wyjściach przerzutnika RS wywołane impulsem $CK = 1$

Aby ocenić dodatkowe korzyści wynikające z tego typu wyzwalania, rozpatrzmy wykresy przedstawione na rysunku 5.19. Przedstawiają one przebiegi czasowe sygnałów na wejściach R i S przerzutnika oraz na

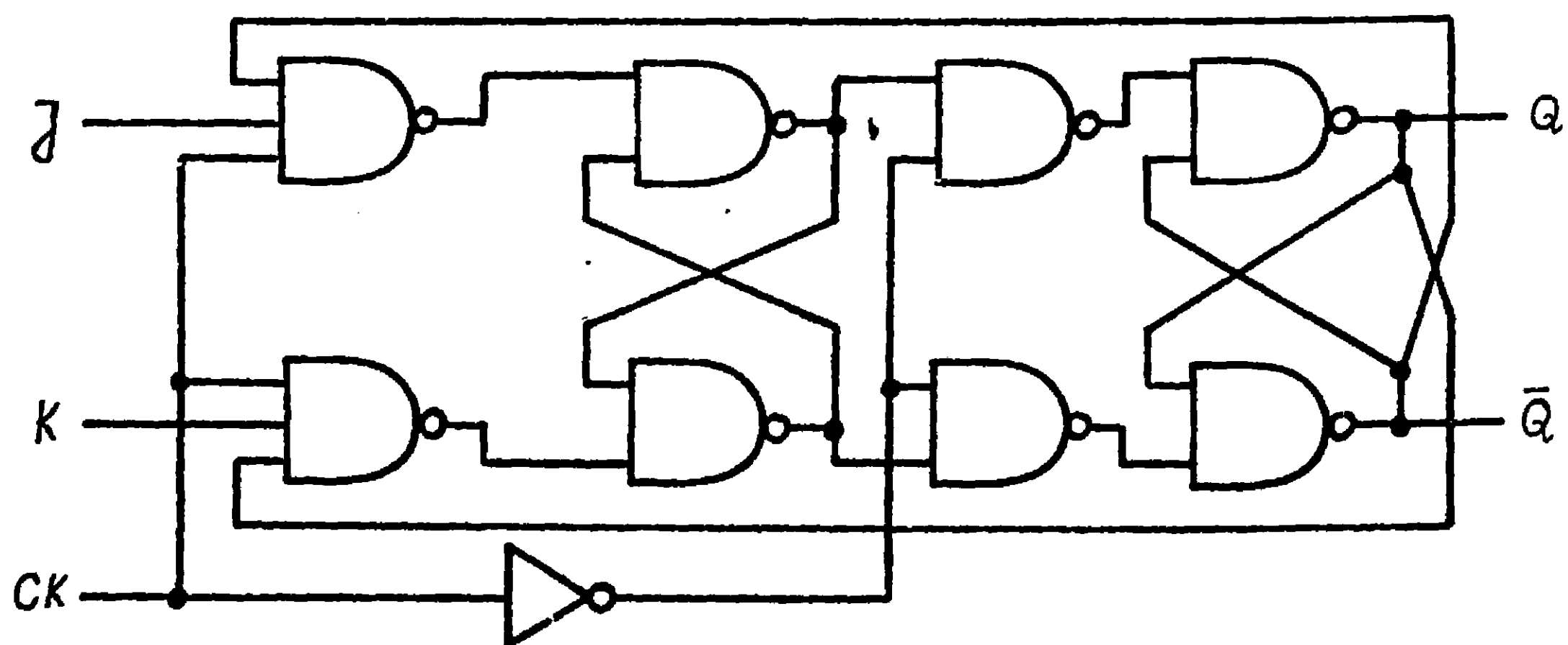
jego wyjściu, przy czym w pierwszym przypadku przerzutnik jest aktywny od momentu pojawienia się impulsu $CK = 1$ do momentu, gdy $CK = 0$. Obrazuje to przebieg określony symbolem Q^1 . Przedstawia on reakcję wyjścia na zmianę sygnału na wejściu S z 1 na 0, a na wejściu R z 0 na 1 z uwzględnieniem skończonych czasów narastania poszczególnych przebiegów. Założono przy tym dla uproszczenia, że szybkość narastania zboczy impulsów zegarowych jest znacznie większa od szybkości zmian pozostałych sygnałów. W przedziale czasu od t_1 do t_2 narasta sygnał na wejściu R i maleje do 0 sygnał na wejściu S . Przerzutnik, którego wyjście miało wartość 1 winien w związku z tym zmienić stan na logiczne 0. Jednak w przedziale czasu (t_1, t_2) sygnały R i S mają jakieś wartości pośrednie, które mogą być odczytane przez bramki przerzutnika bardzo różnie. Rezultat końcowy jest więc obarczony ryzykiem błędu, zależnym od szerokości strefy $t_2 - t_1$. Problem ten przestaje istnieć, jeśli zmiana sygnału na wyjściu będzie mieć miejsce dopiero w momencie przyjscia tylnego zbocza impulsu zegarowego, a więc gdy przebiegi na wejściach będą już ustalone. Obrazuje to przebieg Q^2 . Impuls zegarowy winien wówczas trwać tylko tak długo, jak tego wymagają fizyczne własności ustalania przebiegów.

Dla omawianego w p. 5.2.3 przerzutnika JK, czas trwania impulsu zegarowego musi być krótszy od czasu propagacji sygnału z wejścia na wyjście, gdyż inaczej praca układu okaże się niestabilna. Uaktywnianie wyjścia tylnym zboczem impulsu zegarowego rozwiązuje problem całkowicie, bowiem zmiany sygnałów wejściowych po impulsie zegarowym nie oddziałują już na wyjście, a czas trwania zbocza jest znacznie krótszy od czasu propagacji sygnału przez bramkę.

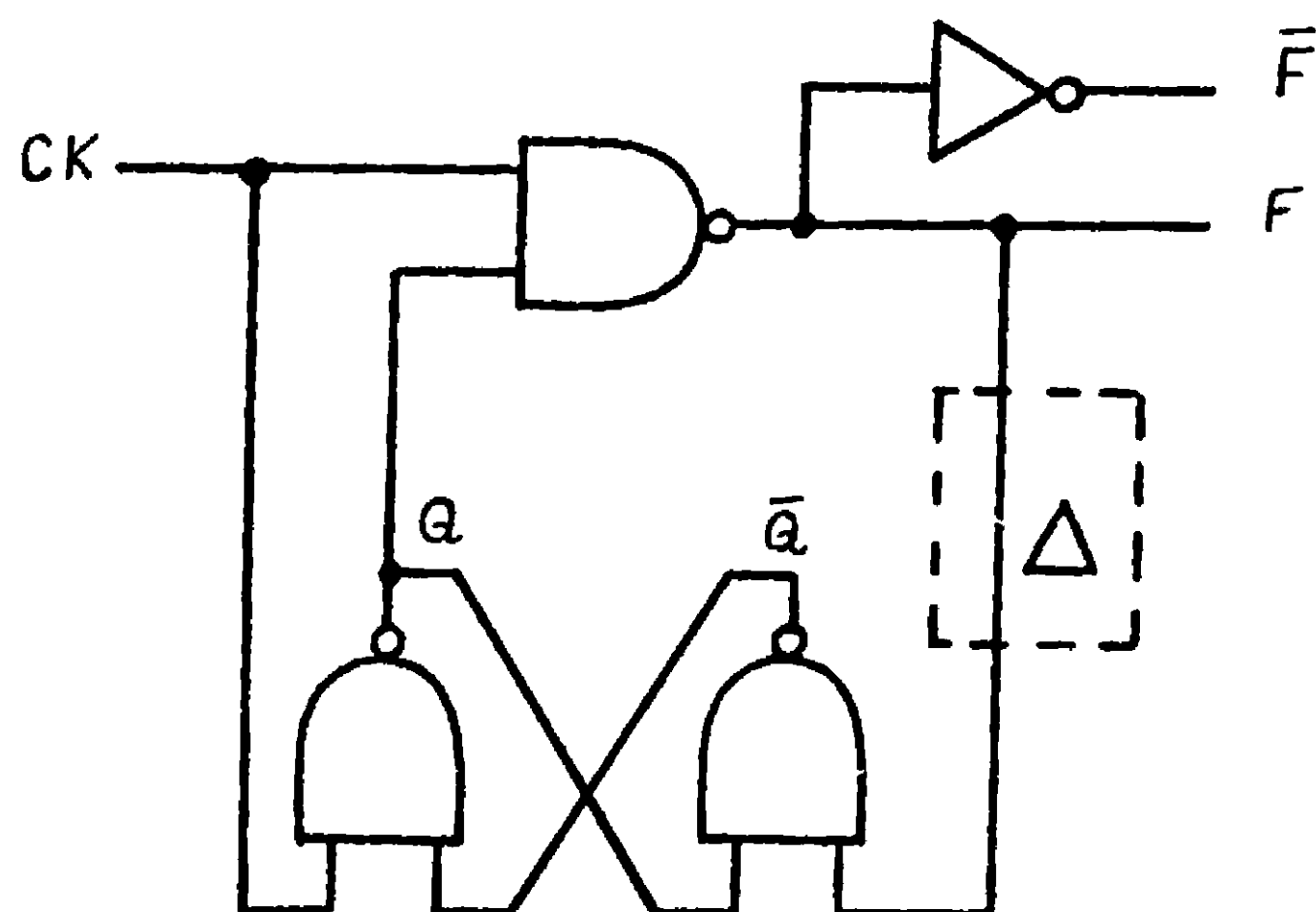


Rys. 5.20. System wyzwalania tylnym zboczem impulsu zegarowego typu Master-Slave (MS)

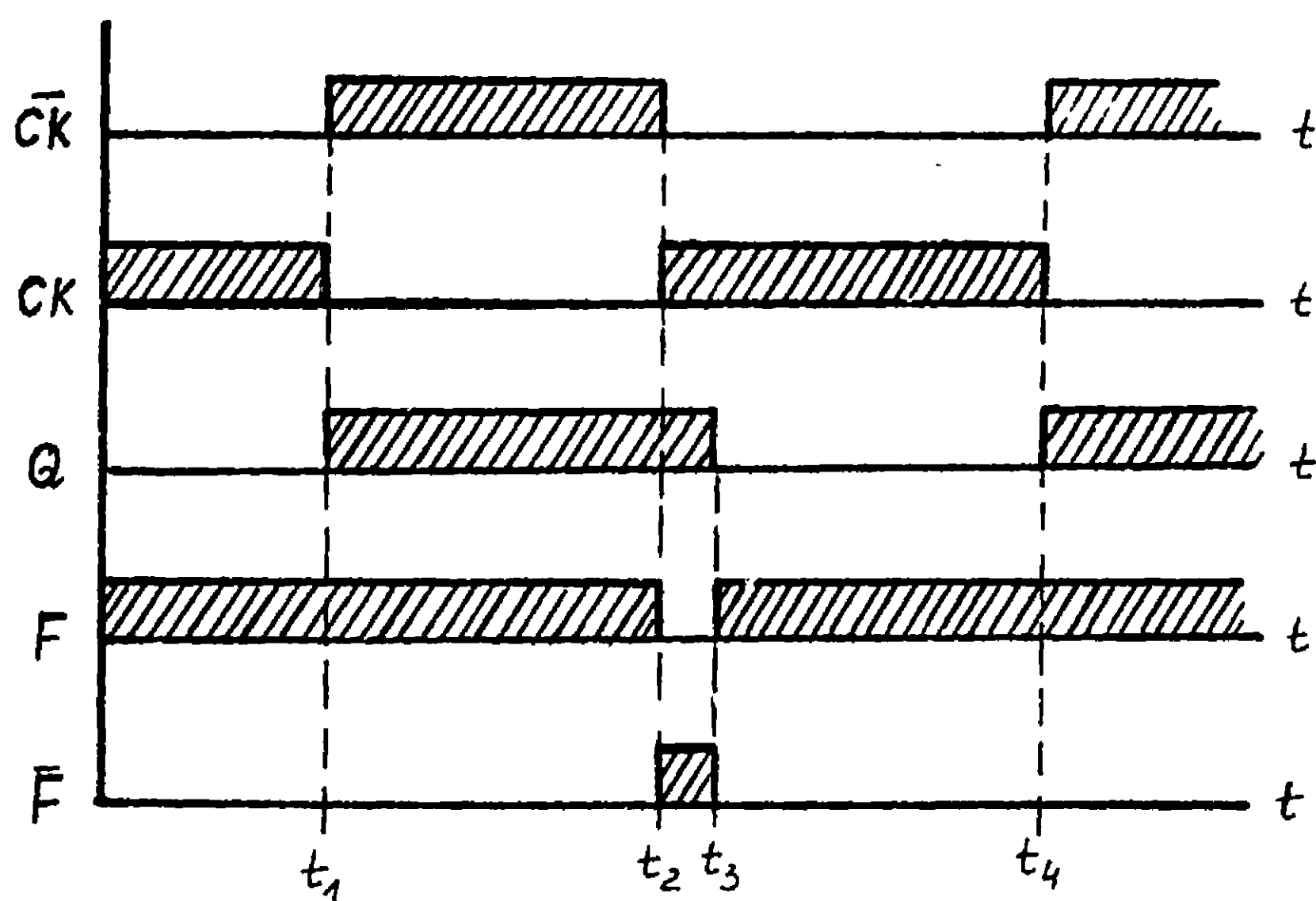
Istnieje kilka metod wyzwalania przerzutników tylnym zboczem. Na rysunku 5.20 przedstawiono system dwuprzerzutnikowy typu Master-Slave. Przerzutnik główny Master działa identycznie jak uprzednio omówione przerzutniki, a więc jego wejścia są aktywne przez cały czas, gdy $CK = 1$. Na wejściu zegarowym przerzutnika pomocniczego Slave jest w tym czasie sygnał $\overline{CK} = 0$, co skutecznie odcina jego wejścia, a więc wyjścia $Q, \overline{Q}$ pozostają niezmiennic. W momencie, gdy impuls zegarowy przyjmie wartość $CK = 0$, stan wyjść przerzutnika głównego Master zostaje przekazany do przerzutnika Slave, na którego wejściu zegarowym tym razem wystąpi sygnał $\overline{CK} = 1$. W tym czasie wejścia przerzutnika głównego są odcięte i wszelkie zmiany sygnałów zewnętrznych nie oddziałują na wyjścia. Przedstawiony system wyzwalania można zastosować dla każdego z omawianych uprzednio przerzutników. Najczęściej stosuje się go jednak do przerzutników typu JK z uwagi na wspomniane już niebezpieczeństwo występowania niestabilnej pracy wywołanej sprzężeniami zwrotnymi. Na rysunku 5.21 przedstawiono przykład realizacji przerzutnika JK z bramek NAND. Składa się on z dwóch przerzutników typu RS (rys. 5.8a), objętych pętlą sprzężenia zwrotnego eliminującego przypadek $R = 1, S = 1$ (rys. 5.12a). Bramki wejściowe przerzutnika głównego stają się aktywne wówczas, gdy $CK = 1$, natomiast bramki wejściowe przerzutnika pomocniczego, gdy $CK = 0$. Zmiana stanu wyjść Q i $\overline{Q}$ może wystąpić najwcześniej w momencie pojawienia się tylnego zbocza impulsu zegarowego, co skutecznie odcina wejścia i zapobiega drganiom. Milcząco zakładamy przy tym, że czas trwania zbocza jest znacznie krótszy od czasu propagacji sygnału przez bramki. Narzuca to określone wymagania na stromość zboczy impulsu zegarowego. W omawianych układach, spełnienie tych wymagań jest technicznie proste i nie wnosi istotnych ograniczeń.



Rys. 5.21. Schemat przerzutnika JK Master-Slave zbudowanego na bramkach typu NAND



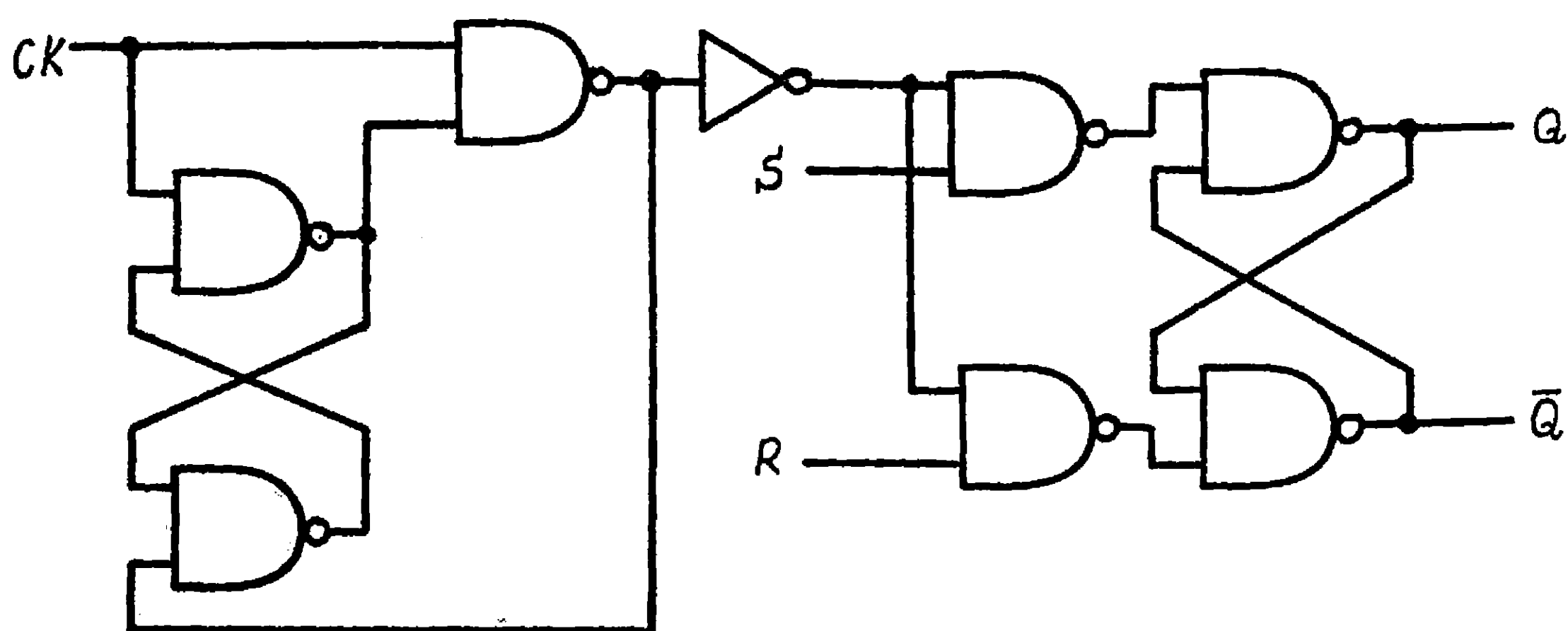
Rys. 5.22. Układ impulsowego różniczkowania zboczy



Rys. 5.23. Przebiegi czasowe sygnałów w układzie impulsowego różniczkowania zboczy

Inny sposób wyzwalania zboczem polega na wykorzystaniu opóźnień wywołanych czasem propagacji sygnału przez bramki logiczne. Na rysunku 5.22 przedstawiono układ impulsowego różniczkowania zboczy, a na rysunku 5.23 przebiegi czasowe sygnałów w układzie. Załóżmy, że w chwili t_1 sygnał zegarowy zmienia wartość z 1 na 0 i pominiemy opóźnienia wnoszone przez bramki przyjmując, że wnosi je jedynie człon opóźniający Δ . Wymusza to na wyjściu F oraz na wyjściu $\bar{Q}$ bramek

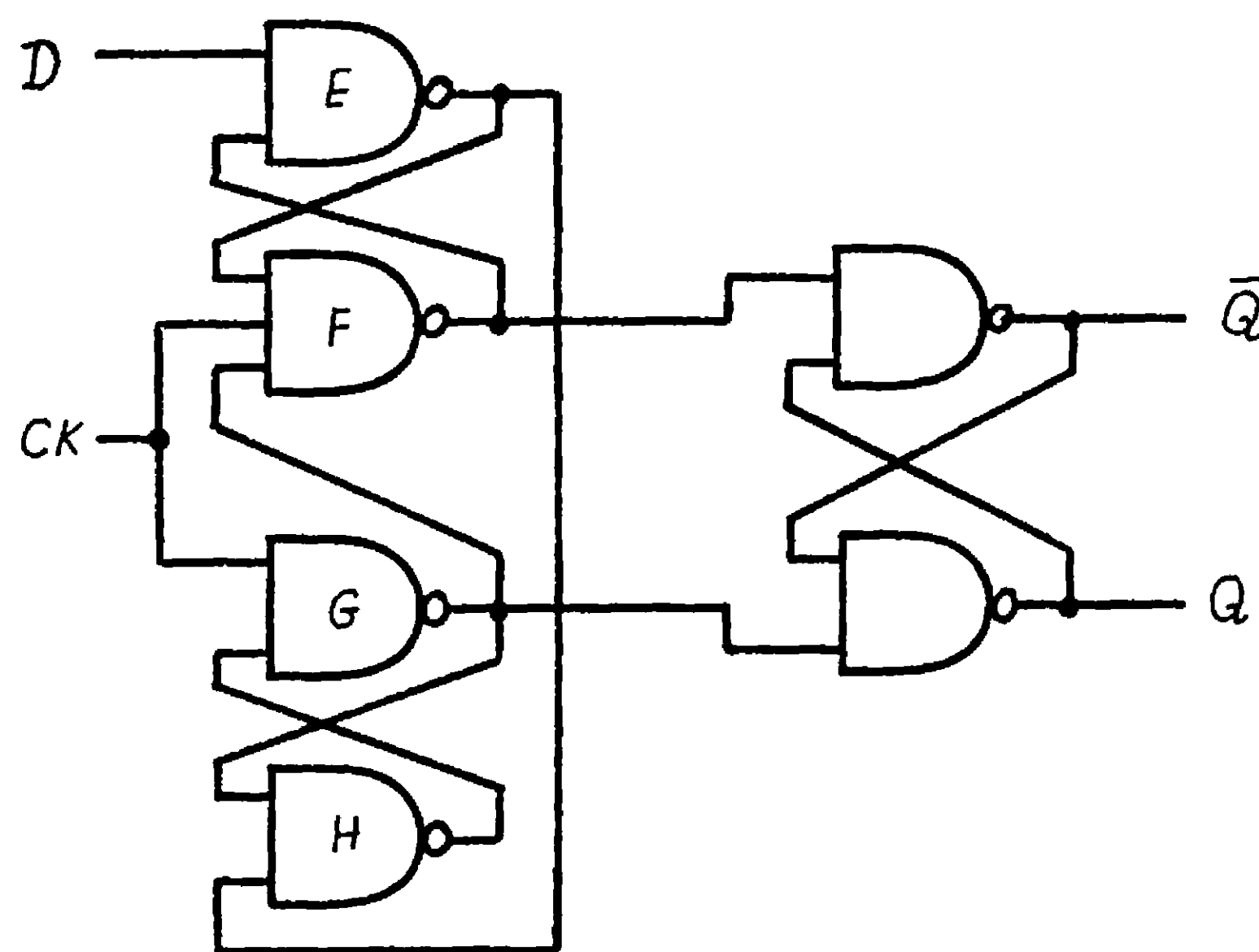
przerzutnikowych stan 1. Wyjście F wraz z wyjściem Q podtrzymuje stan 0 na wyjściu Q . Stan ten jest stabilny tak długo, dopóki impuls zegarowy nie zmieni wartości z 0 na 1 (chwila t_2). Z uwagi na to, że w układzie istnieje człon opóźniający Δ , wyjście F przyjmie wartość zero (jedynki na wejściach Q i CK) i trwać to będzie do momentu t_3 , po którym sygnał $F = 0$ dotrze do wejścia bramki przerzutnika zmieniając stan wyjścia Q na 1 oraz Q na 0. Jednocześnie na wyjściu $\bar{F}$ zostanie wygenerowana "krótka jedynka" będąca rezultatem różniczkowania przedniego zbocza impulsu zegarowego, bądź też tylnego zbocza impulsu dopełniającego $\bar{CK}$. Człon opóźniający Δ został tu wprowadzony sztucznie, dla ułatwienia analizy. W rzeczywistości opóźnienia bramek wystarczają do realizacji efektu różniczkowania. W chwili t_2 sygnał z opóźnieniem wnoszonym przez bramkę sterowaną z zegara, dotrze do pierwszej, a później drugiej bramki przerzutnika, co w zupełności wystarczy do wygenerowania impulsu różniczkowego o czasie trwania równym w przybliżeniu sumie czasów propagacji wszystkich bramek. Sygnał z wyjścia $\bar{F}$ może być podany na wejście zegarowe dowolnego z przerzutników omówionych w punkcie 5.2. Przerzutnik taki staje się wówczas przerzutnikiem wyzwalanym przednim zboczem impulsu CK , lub tylnym impulsu $\bar{CK}$. Na rysunku 5.24 przedstawiono przykładowo przerzutnik RS wyzwalany układem impulsowego różniczkowania zboczy.



Rys. 5.24. Schemat przerzutnika RS wyzwalanego przednim zboczem impulsu zegarowego

Należy zwrócić uwagę na względność pojęcia przednie i tylne zbocze impulsu zegarowego, jeśli operujemy również dopełnieniem tego impulsu. Układy wyzwalane przednim zboczem impulsu CK będą wyzwalane tylnym

zbochem impulsu $\overline{CK}$ i odwrotnie. Najbardziej ogólnym określeniem omówionych sposobów wyzwalania będzie więc nazwanie ich wyzwalaniem na zboczach impulsów zegarowych. W porównaniu z omówionym w punkcie 5.2 wyzwalaniem poziomym impulsu CK, wyzwalanie zboczem osiąga się kosztem dość znacznej rozbudowy układu. Rozwiązania te znalazły więc dopiero powszechne zastosowanie w zintegrowanych strukturach układów logicznych.



Rys. 5.25. Schemat przerzutnika typu D wyzwalanego przednim zboczem impulsu zegarowego

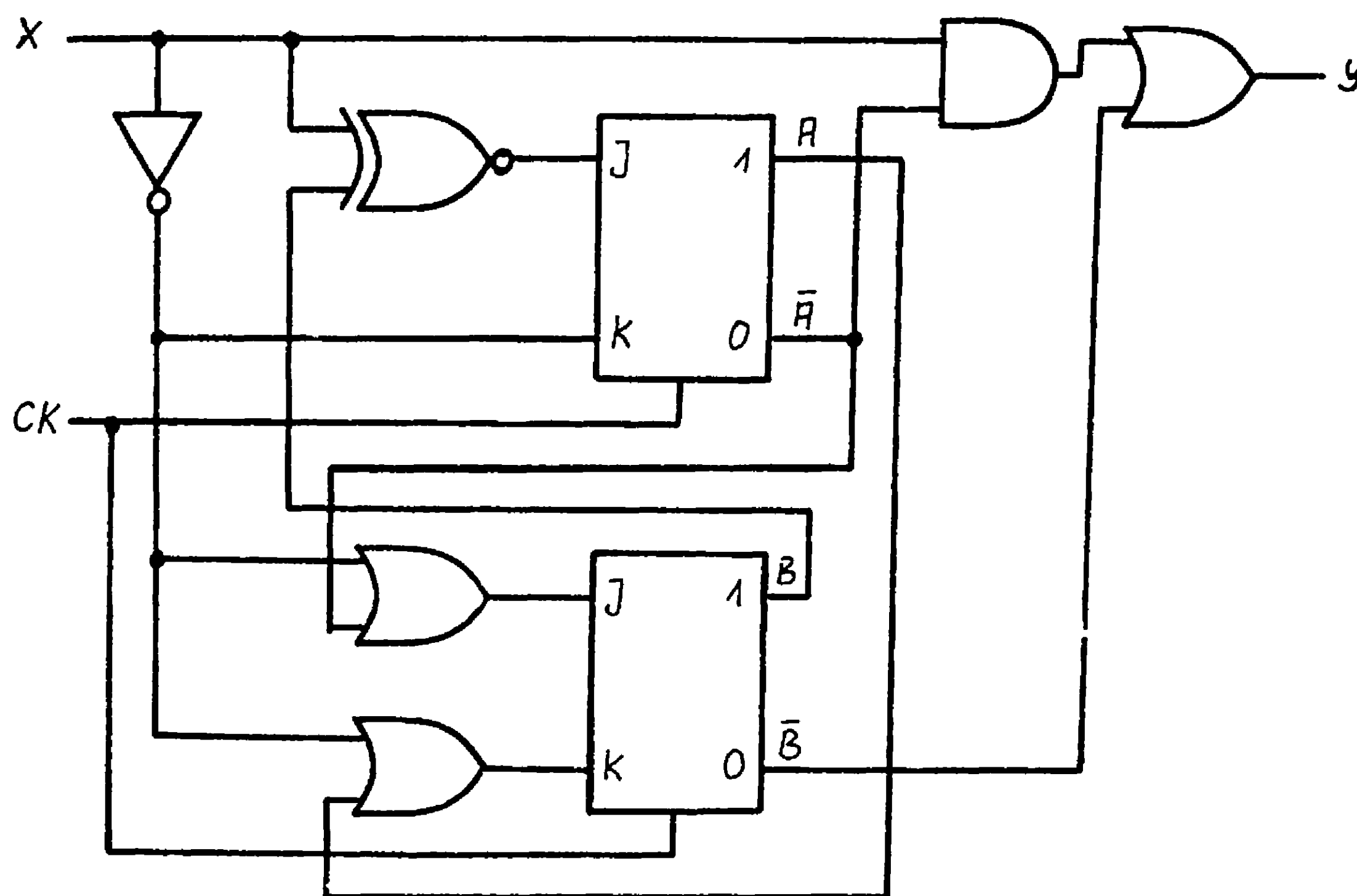
Na zakończenie przedstawionych w tym punkcie rozważań zostanie omówiony interesujący system wyzwalania przednim zboczem impulsu zegarowego przerzutnika typu D, pokazany na rysunku 5.25. Układ składa się z przerzutnika głównego utworzonego z dwóch bramek NAND o wyjściach Q i $\overline{Q}$ oraz z przerzutników pomocniczych utworzonych z bramek E, F i G, H. W okresie, gdy sygnały $CK = 0$ wyjścia bramek F i G mają wartość 1, a więc zgodnie z tabelą pokazaną na rysunku 5.3b przerzutnik główny zachowuje stan wynikający z poprzednich sekwencji. Załóżmy, że wejście D ma wartość $D = 0$. Wymusza to na wyjściu bramki E sygnał 1, podawany dalej na wejście bramki H. Ponieważ drugie wejście tej bramki jest również wysterowane sygnałem 1 z bramki G, więc na wyjściu H otrzymujemy sygnał 0, który dodatkowo wymusza 1 na wyjściu bramki G. W momencie pojawienia się sygnału $CK = 1$, logiczne 0 pojawi się tylko na wyjściu bramki F, gdyż wszystkie jej wejścia są wysterowane sygna-

łem 1. To 0 wymusi sygnał 1 na wyjściu $\bar{Q}$ przerzutnika głównego. W konsekwencji wyjście Q przyjmie wartość 0, gdyż druga bramka jest wystawiana sygnałem 1 (rys. 5.3b). Zero na wyjściu bramki F podtrzymuje 1 na wyjściu E, w konsekwencji ewentualna zmiana sygnału D w czasie, gdy $CK = 1$ nie jest już w stanie zmienić stanu głównego przerzutnika. Układ został więc wyzwolony przednim zboczem, osiągając dla $D = 0$ stan $Q = 0$, niezależnie od stanu poprzedniego, co jest charakterystyczne dla przerzutnika typu D.

Założmy teraz, że wejście $D = 1$. Przed przyjściem impulsu zegarowego wyjścia bramek F i G mają wartość 1, w związku z czym na wyjściu bramki E jest sygnał 0. Wymusza on sygnał 1 na wyjściu bramki H. Jeśli teraz przyjdzie impuls $CK = 1$, to na wyjściu bramki G pojawi się 0, a więc przerzutnik główny osiągnie stan $Q = 1$, gdyż wyjście F ma wartość 1, wymuszona zarówno przez wyjście bramki G, jak i E. Jeśli teraz wystąpiłaby ewentualna zmiana sygnału D na 0, to nie wywołała to reakcji na wyjściu przerzutnika głównego, gdyż zero na wyjściu G podtrzymuje 1 niezależnie na wyjściach bramek H i F, odcinając wpływ bramki E. Układ został więc wyzwolony wyłącznie przednim zboczem impulsu zegarowego osiągając dla wejścia $D = 1$ stan $Q = 1$. Reasumując można stwierdzić, że o szybkości wyzwalań przerzutnika decyduje stan ustalania się poziomu 0 na bramkach F lub G. Po ustaleniu się tego poziomu, dopóki $CK = 1$, przerzutnik jest niewrażliwy na zmiany sygnału zewnętrznego na wejściu D.

5.4. Analiza pracy synchronicznego układu sekwencyjnego

Kompleksową informację o układzie sekwencyjnym zawiera graf tego układu. Nie jest to oczywiście jedyna forma opisu jego pracy, ale odzwierciedla ona najlepiej mechanizm przejść między stanami. Dysponując schematem konkretnego układu, aby określić graf jego pracy wygodnie jest określić tablicę przejść - wyjść dostosowaną do specyfiki układu synchronicznego. Tablicę taką nazywamy często tablicą charakterystyczną układu albo tablicą stanów. Na rysunku 5.26 przedstawiono schemat konkretnego układu sekwencyjnego. Blok pamięciowy tworzą dwa przerzutniki typu JK, których wyjścia oznaczono literami A i B. Wejścia przerzutników oznaczamy najczęściej dwiema literami, z których pierwsza określa typ wejścia (np. J, K, T, D, R, S) a druga nazwę przerzutnika. Dla omawianego układu będą to oznaczenia: JA, KA,



Rys. 5.26. Przykładowy schemat układu sekwencyjnego

JB, KB. Oznaczeń tych nie należy mylić z iloczynem logicznym zmiennych; poza tym występują one w zależnościach boolowskich na ogół po lewej stronie, a więc jako nazwy. Blok kombinacyjny składa się z trzech bramek sumy, jednej bramki równoważności, jednej bramki iloczynu i z jednego inwertera dla zmiennej wejściowej x . Posiada on jedno wyjście zewnętrzne y oraz cztery wyjścia sterujące wejściami przerzutników: JA, KA, JB, KB. Na rysunku 5.27 przedstawiono tablicę przejść - wyjść oraz graf układu pokazanego na rysunku 5.26. Podobnie jak podczas omawiania przerzutników, stan aktualny oznaczamy indeksem t , natomiast stan następny indeksem $t + 1$. Aby tę tabelę wypełnić, należy w oparciu o dany schemat określić funkcje boolowskie opisujące blok kombinacyjny. Z rysunku 5.26 wynikają dla poszczególnych wyjść następujące zależności:

- dla wyjścia zewnętrznego

$$y = \bar{A}x + \bar{B},$$

- dla wyjść sterujących wejścia przerzutników:

$$JA = x \odot B = Bx + \bar{B}\bar{x},$$

$$KA = \bar{x},$$

$$JB = \bar{A} + \bar{x},$$

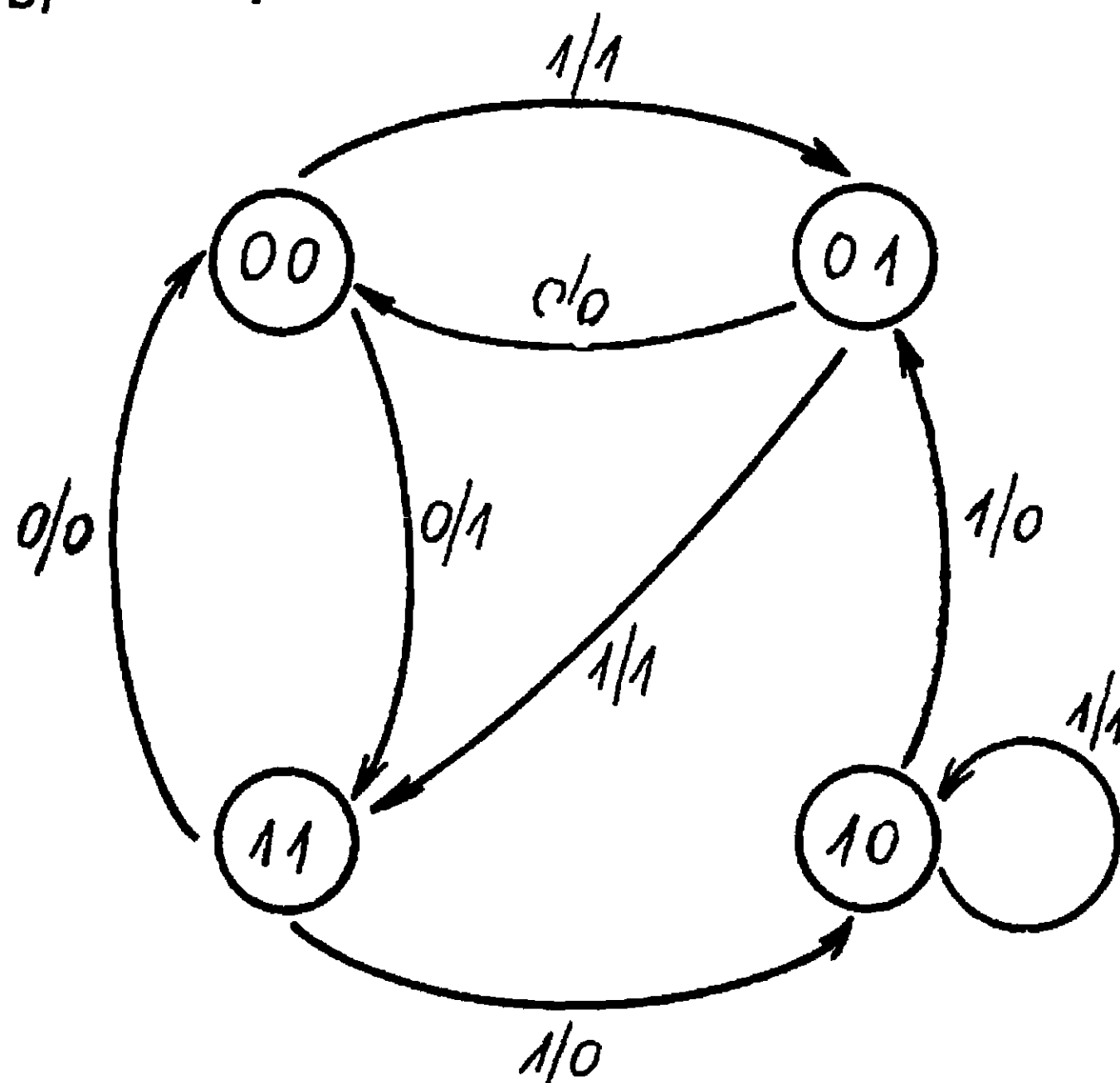
$$KB = A + \bar{x}.$$

Z uwagi na to, że mamy do czynienia z układem kombinacyjnym, nie ma potrzeby wprowadzania indeksów do nazw przerzutników. Zwróćmy uwagę na fakt, że podany wyżej zbiór formuł boolowskich całkowicie wystarcza do pełnego odtworzenia schematu logicznego układu sekwencyjnego.

a)

A_t	B_t	X	A_{t+1}	B_{t+1}	Y
0	0	0	1	1	1
0	0	1	0	1	1
0	1	0	0	0	0
0	1	1	1	1	1
1	0	0	0	1	1
1	0	1	1	0	1
1	1	0	0	0	0
1	1	1	1	0	0

b)



Rys. 5.27. Tablica przejść - wyjść (a) oraz graf (b) układu pokazanego na rys. 5.26

W oparciu o podane zależności oraz tablicę wzbudzeń przerzutnika JK (rys. 5.13a) można wypełnić tablicę przejść - wyjść i narysować graf układu. Zarówno tablica jak i graf nie zawierają informacji o rodzaju zastosowanych przerzutników, konieczne jest więc podanie dodatkowego komentarza do tablicy grafu. Z tablicy charakterystycznej można określić równania stanów następnego układu, przy czym formuły minimalne uzyskamy wpisując funkcje do tablic Karnaugh (rys. 5.28):

$$A_{t+1} = B_t x + A_t x + \bar{A}_t \bar{B}_t \bar{x},$$

$$B_{t+1} = \bar{B}_t \bar{x} + \bar{A}_t x.$$

a)

x $A_t B_t$		0	1
00	1	0	
01	0	1	
11	0	1	
10	0	1	

b)

x $A_t B_t$		0	1
00	1	1	
01	0	1	
11	0	0	
10	1	0	

Rys. 5.28. Tablice Karnaugh funkcji stanów następujących układu pokazanego na rys. 5.26 (a,b)

Równania powyższe wraz z równaniem wyjścia y , zawierają te same informacje co graf i tablica przejść - wyjść układu. Nie zawierają one informacji o rodzaju zastosowanych przerzutników - można je dopiero dostosować do równań charakterystycznych przerzutników założonego typu. Przekształcając równanie stanu dla przerzutnika A, otrzymamy

$$\begin{aligned}
 A_{t+1} &= B_t x + A_t x + \bar{A}_t \bar{B}_t \bar{x} = (\bar{A}_t B_t + A_t) x + \bar{A}_t \bar{B}_t \bar{x} = \\
 &= \bar{A}_t B_t x + \bar{A}_t \bar{B}_t \bar{x} + x A_t = (B_t x + \bar{B}_t \bar{x}) \bar{A}_t + x A_t = \\
 &= (B_t \odot x) \bar{A}_t + x A_t = (JA) \bar{A}_t + (\overline{KA}) A_t.
 \end{aligned}$$

Ponieważ $JA = B_t \odot x$ oraz $KA = x$, więc otrzymaliśmy w efekcie równanie charakterystyczne przerzutnika JK. W analogiczny sposób można przekształcić równanie stanu dla przerzutnika B

$$\begin{aligned}
 B_{t+1} &= \bar{B}_t \bar{x} + \bar{A}_t x = \bar{B}_t \bar{x} + \bar{A}_t x + \bar{A}_t \bar{B}_t = \\
 &= \bar{B}_t \bar{x} + \bar{A}_t (\bar{B}_t + x B_t) = \bar{B}_t \bar{x} + \bar{A}_t \bar{B}_t + x \bar{A}_t B_t = \\
 &= (\bar{A}_t + \bar{x}) \bar{B}_t + (\overline{A_t + \bar{x}}) B_t = (JB) \bar{B}_t + (\overline{KB}) B_t;
 \end{aligned}$$

tutaj $JB = \bar{A}_t + \bar{x}$ oraz $KB = A_t + \bar{x}$ (rys. 5.26).

Przekształcone w powyższy sposób równania stanów są już dostosowane do realizacji na przerzutnikach typu J K. Zbiór tych równań w połączeniu z równaniem dla wyjścia y zawiera pełną informację o analizowanym układzie, tzn. stanowi on podstawę do narysowania schematu logicznego synchronicznego układu sekwencyjnego.

Reasumując można stwierdzić, że obok schematu logicznego, komp'eksową informację o układzie synchronicznym zawierają: tablica przejść - wyjść, graf, równania stanów oraz równania wyjść bloku kombinacyjnego. W ostatnim przypadku jest zawarta explicite również informacja o zastosowanych przerzutnikach. W pozostałych przypadkach trzeba to dodatkowo dopowiedzieć. Każdy z wymienionych sposobów można stosować zamiennie zależnie od zadań analizy.

5.5. Liczniki i rejestry

Liczba synchronicznych układów sekwencyjnych nie jest oczywiście ograniczona, wszystkie jednak dadzą się sprowadzić do schematu przedstawionego na rysunku 5.1. Podobnie jak dla układów kombinacyjnych, pewne typowe podzespoły sekwencyjne systemów cyfrowych doczekały się nazw. Należą do nich głównie układy zwane licznikami i zbiory przerzutników przechowujących informację określone mianem rejestrów.

5.5.1. Liczniki

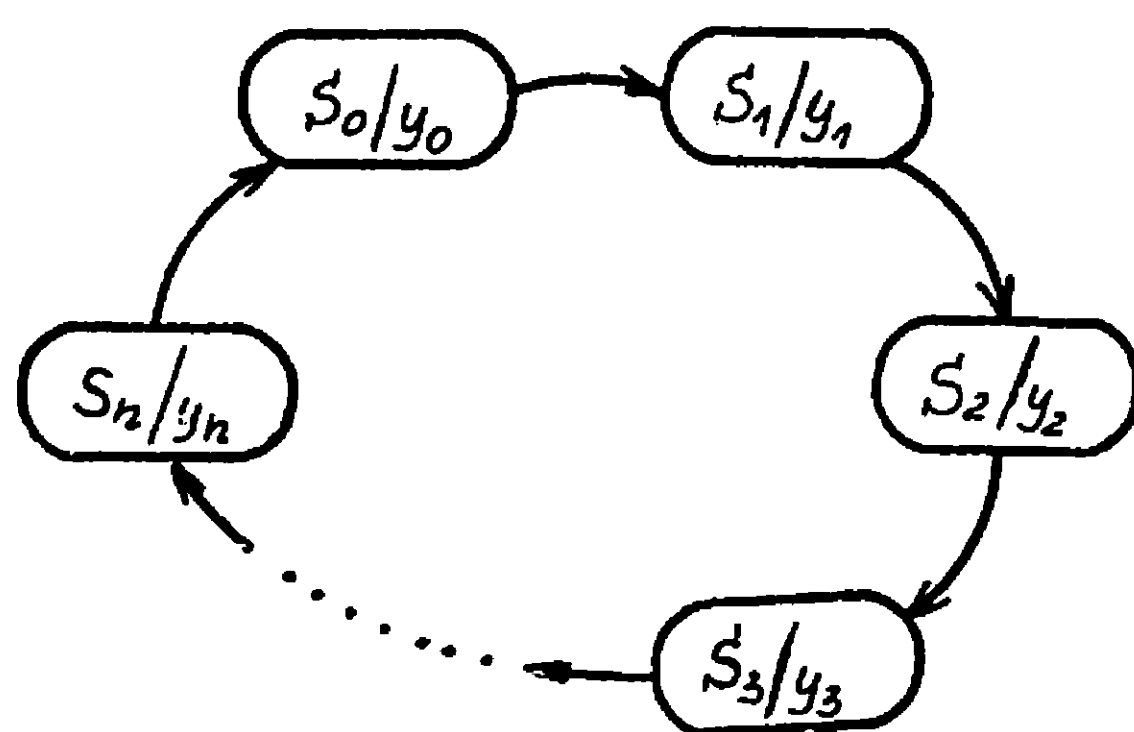
Licznikiem nazywamy układ sekwencyjny typu Moore'a, który zmienia stan po każdym impulsie zegarowym. Graf licznika tworzy zamknięty pierścień stanów. Na rysunku 5.29 przedstawiono graf automatu będącego licznikiem o $n+1$ stanach wewnętrznych, a na rysunku 5.30 przykład realizacji licznika o dziesięciu stanach wewnętrznych, tzw. synchronicznej dekady liczącej. Jako przerzutników użyto układów JK. Tablicę charakterystyczną i graf stanów przedstawiono na rysunku 5.31. Litery wyjściowe utożsamiono ze stanami. Zwraca uwagę brak zmiennych wejściowych zewnętrznych. Zmiany stanów wewnętrznych występują kolejno po przejściu impulsu zegarowego, przy czym po dziesięciu impulsach cykl się powtarza. Stąd nazwa licznika: dekada licząca. Równania wejściowe układu są następujące:

$$JA = A B C, \quad KA = D,$$

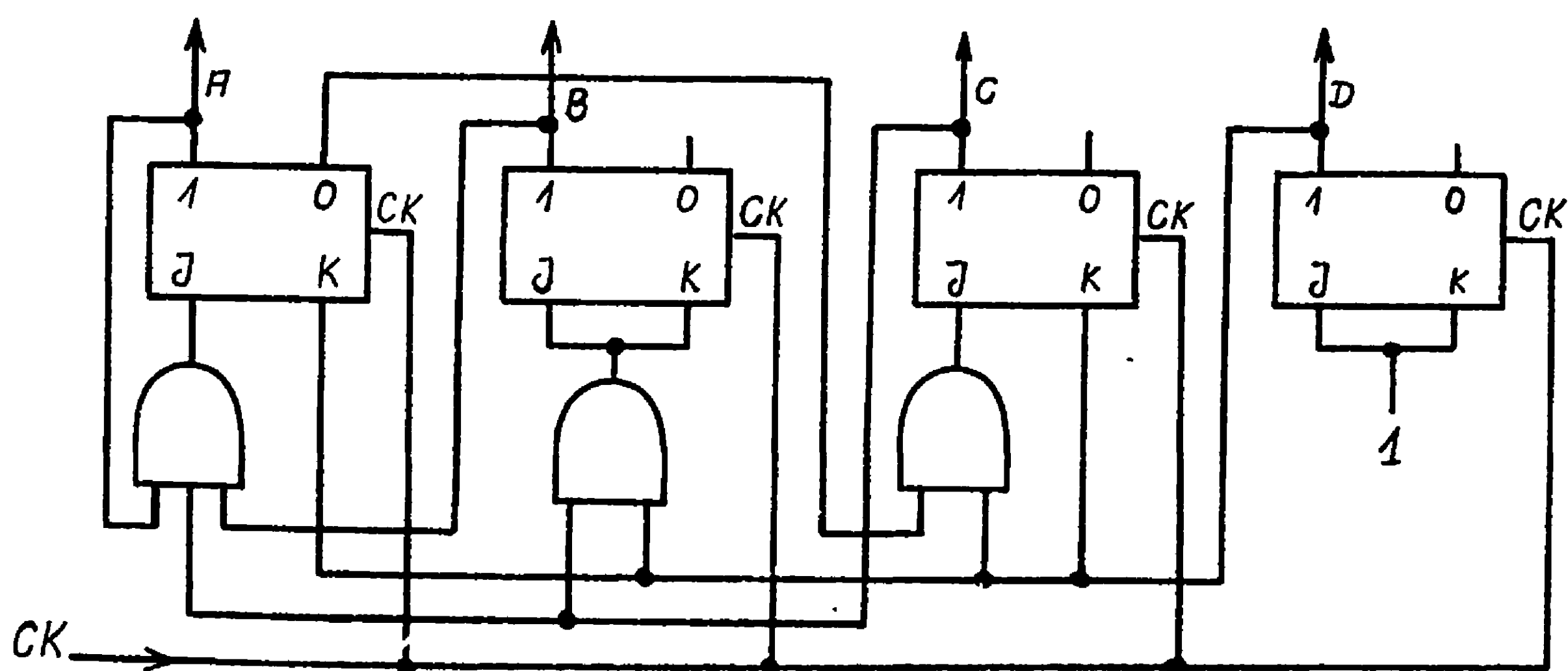
$$JB = KB = CD,$$

$$JC = \bar{A}D, \quad KC = D,$$

$$JD = KD = 1.$$



Rys. 5.29. Graf automatu Moore'a będącego licznikiem o $n+1$ stanach wewnętrznych

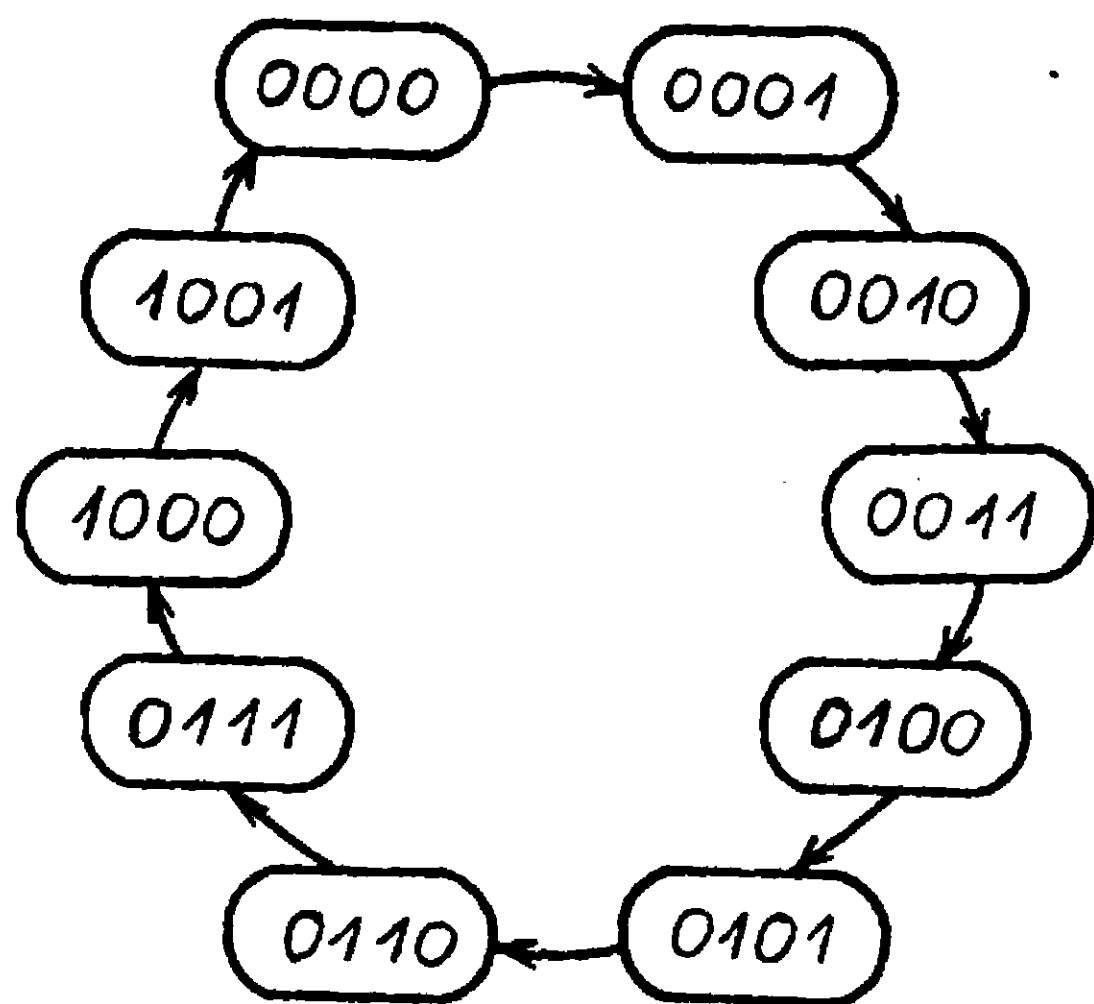


Rys. 5.30. Schemat synchronicznej dekady liczącej zbudowanej na przerzutnikach JK

Opisują one w sposób jednoznaczny schemat pokazany na rysunku 5.30. Łącząc wyjścia dekady z dekodерem czterowejściowym (p. 3.5) uzyskamy informację explicite o liczbie impulsów zegarowych zliczonych przez dekadę, pod warunkiem, że startujemy od stanu 0000. Przerzutniki scalone posiadają specjalne wejścia umożliwiające ustawianie stanu 0 lub 1 niezależnie od impulsów synchronizujących. Za pomocą tych wejść wymuszamy stan startowy licznika.

Wypada jeszcze wspomnieć o niekonwencjonalnym wykorzystaniu wejść zegarowych przerzutników synchronicznych, celem utworzenia podzielnika liczby impulsów zegarowych przez 2^n , gdzie n - liczba przerzutników. Na rysunku 5.32a przedstawiono układ złożony z trzech przerzutników typu T - MS (Master - Slave), a więc wyzwalanych tylnym zboczem. Graf układu przedstawiono na rysunku 5.32b, a przebiegi czasowe na rysun-

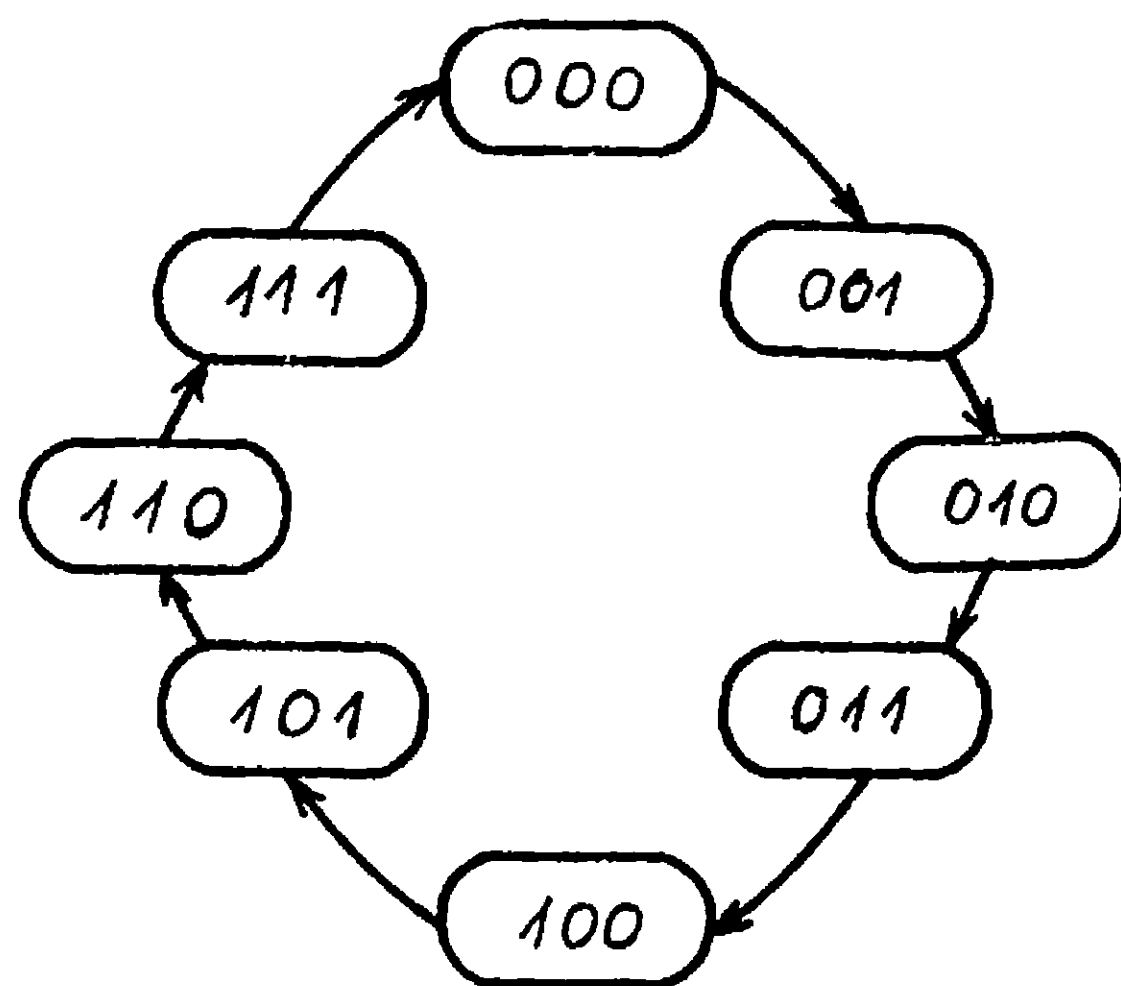
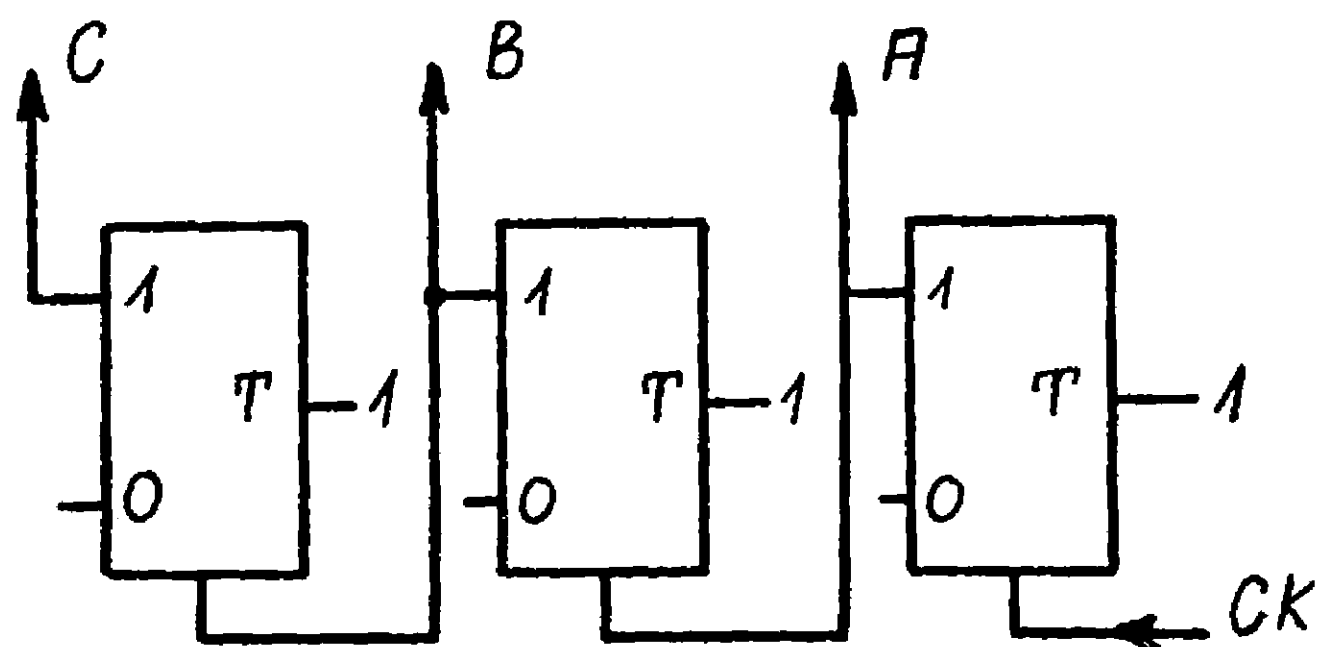
a)



b)

A_t	B_t	C_t	D_t	A_{t+1}	B_{t+1}	C_{t+1}	D_{t+1}
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	0	0	0	0

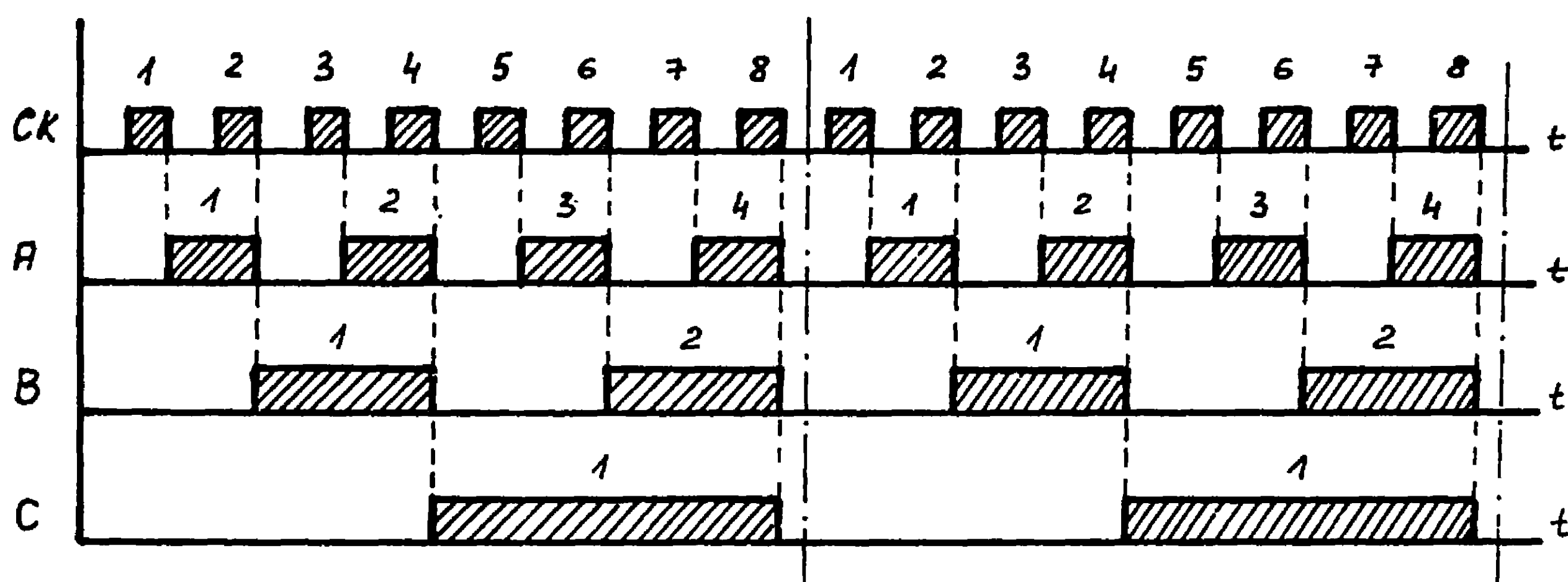
Rys. 5.31. Graf (a) oraz tablica charakterystyczna dekady liczącej pokazanej na rys. 5.30 (b)



Rys. 5.32. Schemat i graf układu licznika będącego podzielnikiem przez 8 liczby impulsów zegarowych

ku 5.33. Cykl pracy licznika wynosi 8, przy czym po przejściu 8 impulsów CK na wyjściu przerzutnika C jest wygenerowany tylko jeden impuls. Licznik można więc traktować jako podzielnik przez 8 liczby impulsów wejściowych. Wejścia T wszystkich przerzutników sąysterowane

sygnałem 1, w związku z czym każdorazowo na tylnym zboczu impulsu podanego na wejście zegarowe następuje zmiana stanu wyjścia odpowiedniego przerzutnika. Przerzutnik A generuje dwukrotnie, przerzutnik B czterokrotnie, a przerzutnik C - ośmiokrotnie mniejszą liczbę impulsów niż wynosi liczba impulsów podanych na wejście licznika.



Rys. 5.33. Przebiegi czasowe na wyjściach układu licznika pokazanego na rys. 5.32

Wykorzystując np. układ impulsowego różniczkowania zboczy oraz różnicę w czasach propagacji sygnałów, można tworzyć podzielniki o dowolnym cyklu pracy. Liczników tych - podobnie jak omówionego już układu pokazanego na rys. 5.32a - nie można jednak nazwać licznikami synchronicznymi, mimo iż są złożone z przerzutników synchronicznych, gdyż zmiany stanu na ich wyjściach nie są bezpośrednio synchronizowane i występują z opóźnieniami wynikającymi z różnych czasów propagacji. Z uwagi na wspomniane zastosowanie przerzutników synchronicznych, możliwość bazującego głównie na intuicji konstruowania tych układów, jest tutaj tylko sygnalizowana.

5.5.2. Rejestry

Rejestrem nazywamy zbiór przerzutników służących do przechowywania informacji. Liczbę przerzutników nazywamy długością rejestru. Rejestr jako automat jest rozpatrywany głównie w kategoriach układu logicznego, tzn. sygnały i stany są oznaczane ciągami zero-jedynkowymi.

Informację do rejestru można wprowadzać w różny sposób. Do najbardziej typowych należy wprowadzanie szeregowo z przesuwaniem w lewo lub w prawo oraz wprowadzanie równoległe. Rejestry z wprowadzaniem szeregowym są określane jako rejestry szeregowo, natomiast z wprowadzaniem równoległym - jako rejestry równoległe. Rejestry łączące obie te metody zapisu informacji są określane jako rejestry szeregowo-równoległe. Proces równoległego wpisywania informacji jest wtedy często określany jako tzw. ładowanie rejestru (ang. load). Termin ten stosuje się również podczas wpisywania informacji do rejestru szeregowego, jednak z dodaniem przymiotnika "szeregowo", dla zaakcentowania, że proces jest realizowany dla tylu impulsów zegarowych, ile bitów liczy informacja. Rejestr jest podstawowym układem programowalnych systemów cyfrowych. Struktura procesora maszyn cyfrowych jest strukturą rejestrową. Zbiory rejestrów tworzą pamięci operacyjne lub notatnikowe. Rejestry są także podstawowymi podzespołami urządzeń wejścia i wyjścia komputerów i terminali. Wielka rola rejestru w zastosowaniach do sprzętu informatycznego, jest niewspółmierna do jego skromnej struktury układowej. Z punktu widzenia układów logicznych, rejestr jest kaskadą prostych komórek pamięciowych z warunkowym sterowaniem wejść. Na rysunku 5.34a pokazano taką komórkę, a na rysunku 5.34b przedstawiono kaskadę tworzącą rejestr czterobitowy. Równanie wejściowe prostej komórki pamięciowej będzie mieć postać

$$D y_i = R I_{R_i} + L I_{L_i} + P x_i,$$

gdzie: R - sygnał warunku przesuwania w prawo,
 L - sygnał warunku przesuwania w lewo,
 I_{R_i} - wejście szeregowego przesuwania w prawo,
 I_{L_i} - wejście szeregowego przesuwania w lewo,
 P - sygnał warunku wpisywania równoległego (ładowania),
 x_i - wejście wpisywania równoległego.

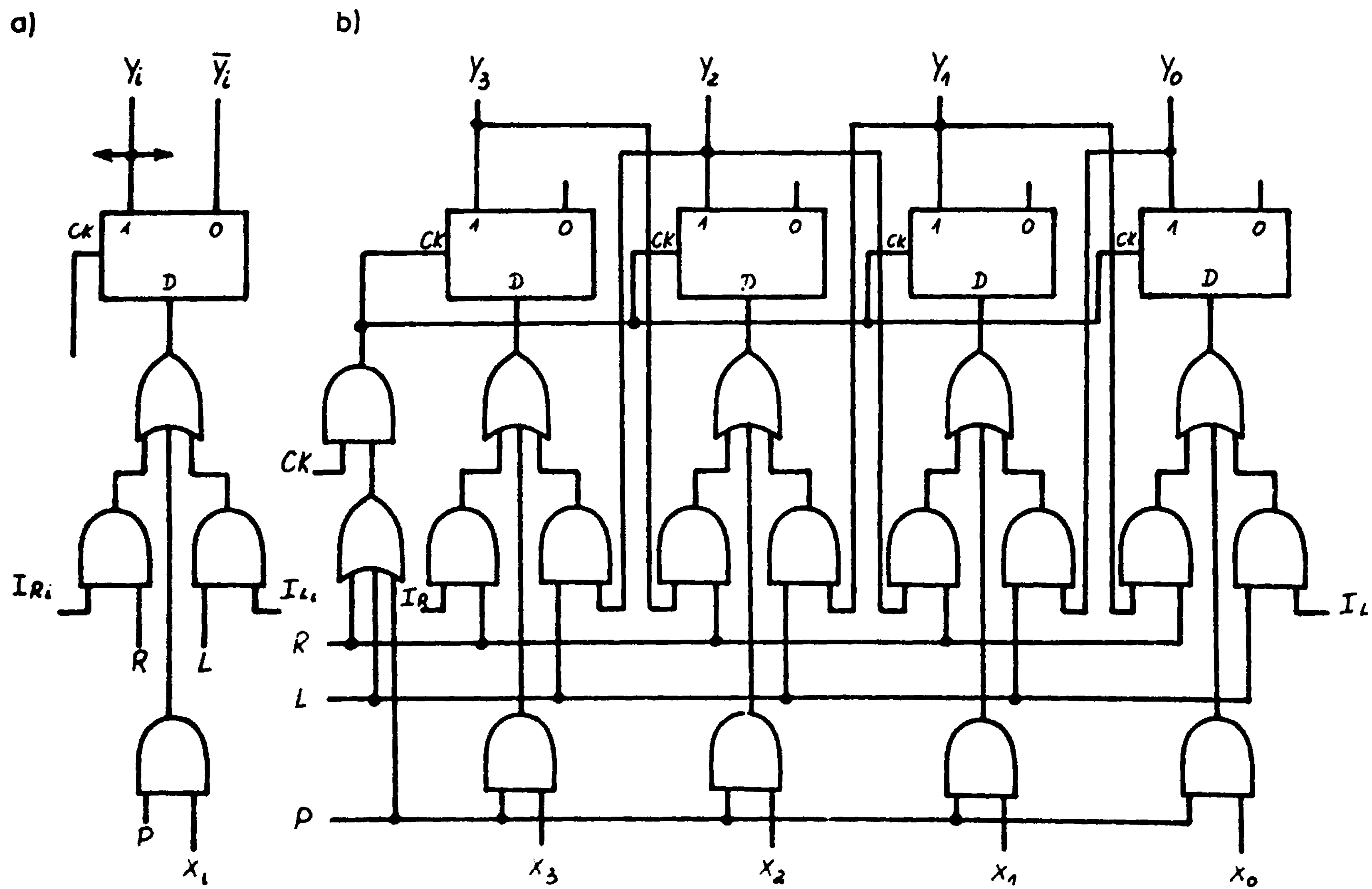
Wektor stanów następnych rejestru czterobitowego będzie opisany zależnościami:

$$y_3^{t+1} = R I_R + L y_2^t + P x_3,$$

$$y_2^{t+1} = R y_3^t + L y_1^t + P x_2,$$

$$y_1^{t+1} = R y_2^t + L y_0^t + P x_1,$$

$$y_0^{t+1} = R y_1^t + L I_L + P x_0.$$



Rys. 5.34. elementarna komórka pamięciowa i czterobitowy rejestr szeregowo - równoległy złożony z tych komórek

Traktując rejestr jako blok iteracyjny złożony z komórek elementarnych układów pamięciowych, równania powyższe można zapisać w postaci bardziej ogólnej. Dla i -tej komórki otrzymamy

$$y_i^{t+1} = R y_{i+1}^t + L y_{i-1}^t + P x_i \quad (i = 1, \dots, n-1),$$

natomiast na brzegach, formuły boolowskie przyjmą postać:

$$y_n^{t+1} = R I_R + L y_{n-1}^t + P x_n,$$

$$y_0^{t+1} = R y_1^t + L I_L + P x_0.$$

Równania stanów następnych są niezależne od rodzaju zastosowanego przerzutnika, jednak właściwości przerzutnika typu D predysponują go szczególnie do budowy rejestrów.

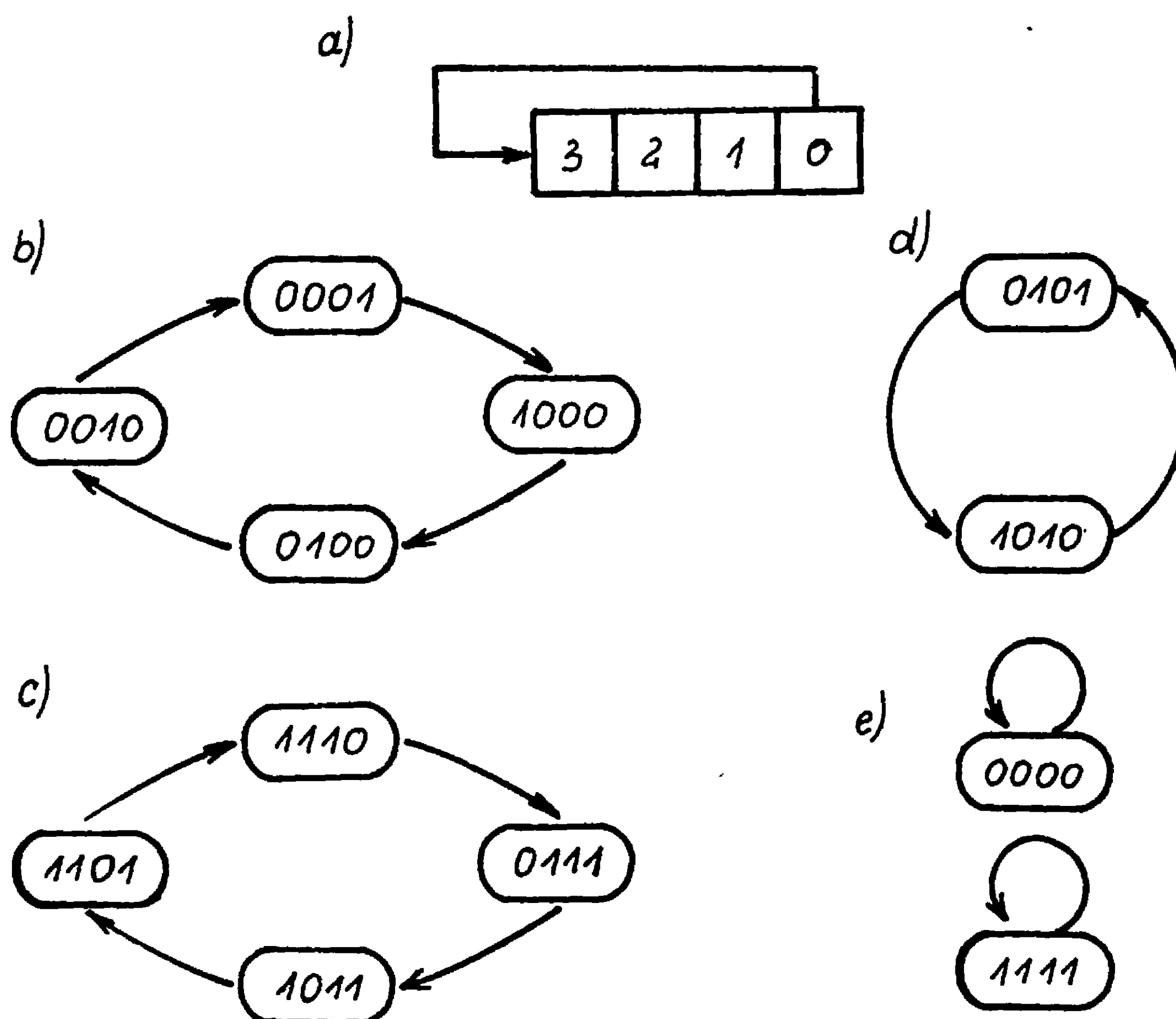
Warunkiem prawidłowej pracy rejestru jest nieprzyjmowanie przez wejścia L, P, R jednocześnie sygnału 1. W praktyce najczęściej do sterowania warunkami stosuje się dwa sygnały a, b dekodowane na wejściu rejestru zgodnie z zależnościami:

$$L = a \bar{b}, R = \bar{a} b, P = a b.$$

Wektor $a = 0, b = 0$ blokuje wejścia zegarowe rejestru, co umożliwia przechowywanie informacji.

Łącząc wyjście ostatniego przerzutnika z wejściem szeregowym przerzutnika pierwszego (licząc - zależnie od sygnału warunku - z lewej do prawej bądź odwrotnie) umożliwiamy cykliczne przesuwanie się zapisywanej informacji. Graf układu będzie wówczas tworzył zamknięty pierścień stanów charakterystyczny dla licznika. Dlatego też rejestr z połączonym wyjściem i wejściem szeregowym przyjęto nazywać licznikiem pierścieniowym. Ciągi zerojedynkowe reprezentujące poszczególne stany wewnętrzne zależą od załadowanej informacji początkowej. Maksymalny cykl pracy licznika jest równy liczbie przerzutników w rejestrze i zależy od stanu startowego. Na rysunku 5.35 przedstawiono licznik złożony z rejestru czterobitowego oraz grafy określone dla różnych stanów startowych. Zawsze po czterech impulsach zegarowych licznik powraca do stanu startowego, z tym że dla niektórych stanów startowych, stany pośrednie są nieodróżnialne i obserwowany cykl grafu może być mniejszy (rys. 5.35d). Stany 1111 oraz 0000 dają nieodróżnialne cykle

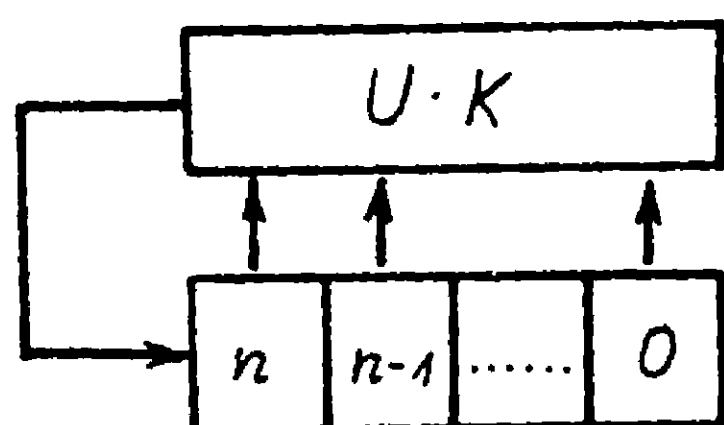
(rys. 5.35e) w związku z tym nie tworzą pierścienia stanów i są z punktu widzenia licznika niedozwolone. Łącznie, dla n -bitowego rejestru można otrzymać wiele różnych grafów bez zmiany konfiguracji układu, co niejednokrotnie jest bardzo użyteczne w zastosowaniach.



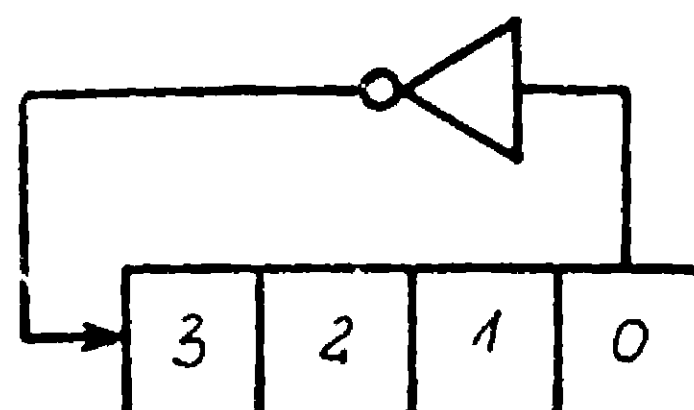
Rys. 5.35. Rejestr czterobitowy jako licznik pierścieniowy i przykłady realizowanych przez licznik grafów (a,b,c,d,e)

Wydłużenie maksymalnego cyklu pracy licznika zbudowanego z rejestru można osiągnąć wprowadzając w pętlę sprzęgającą wejście z wyjściem układ kombinacyjny realizujący formułę boolowską, której argumentami są sygnały na wyjściach określonych przerzutników rejestru (rys. 5.36a). Na rysunku 5.36b,c,d przedstawiono licznik uzyskany przez zastosowanie inwertera sprzęgającego wyjście z wejściem rejestru czterobitowego. Realizuje on tylko dwa różne grafy o ośmiu stanach wewnętrznych i nie ma stanów niedozwolonych. Gdyby stany te umieścić w rogach ośmiokąta foremnego, to liczby binarne reprezentujące ciągi zero-jedynkowe stanów, umieszczone na najdłuższych przekątnych, dadzą w sumie zawsze liczbę dziesiętną 15 (1111). Są więc one wobec siebie wzajemnie uzupełnieniami jedynekowymi. Liczniki tak zbudowane noszą nazwę liczników pseudopierścieniowych.

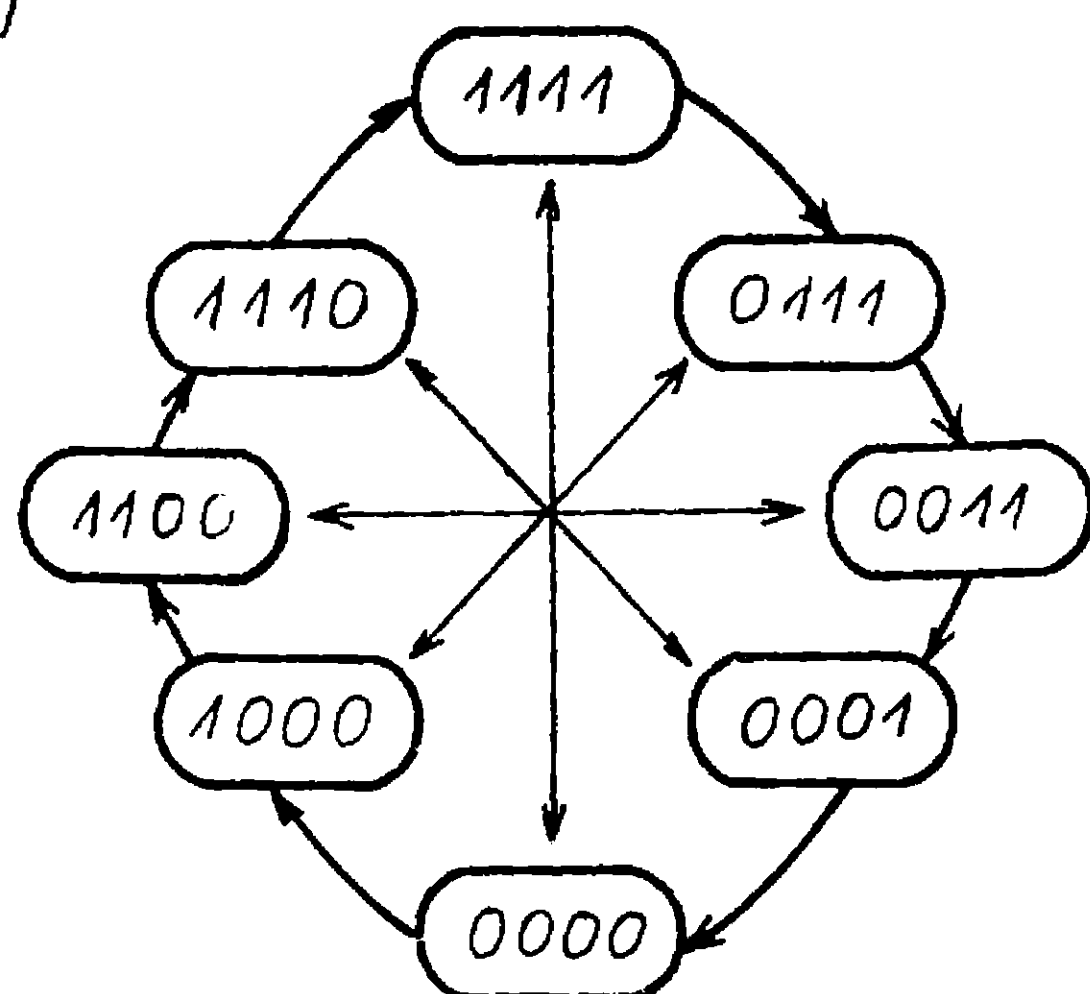
a)



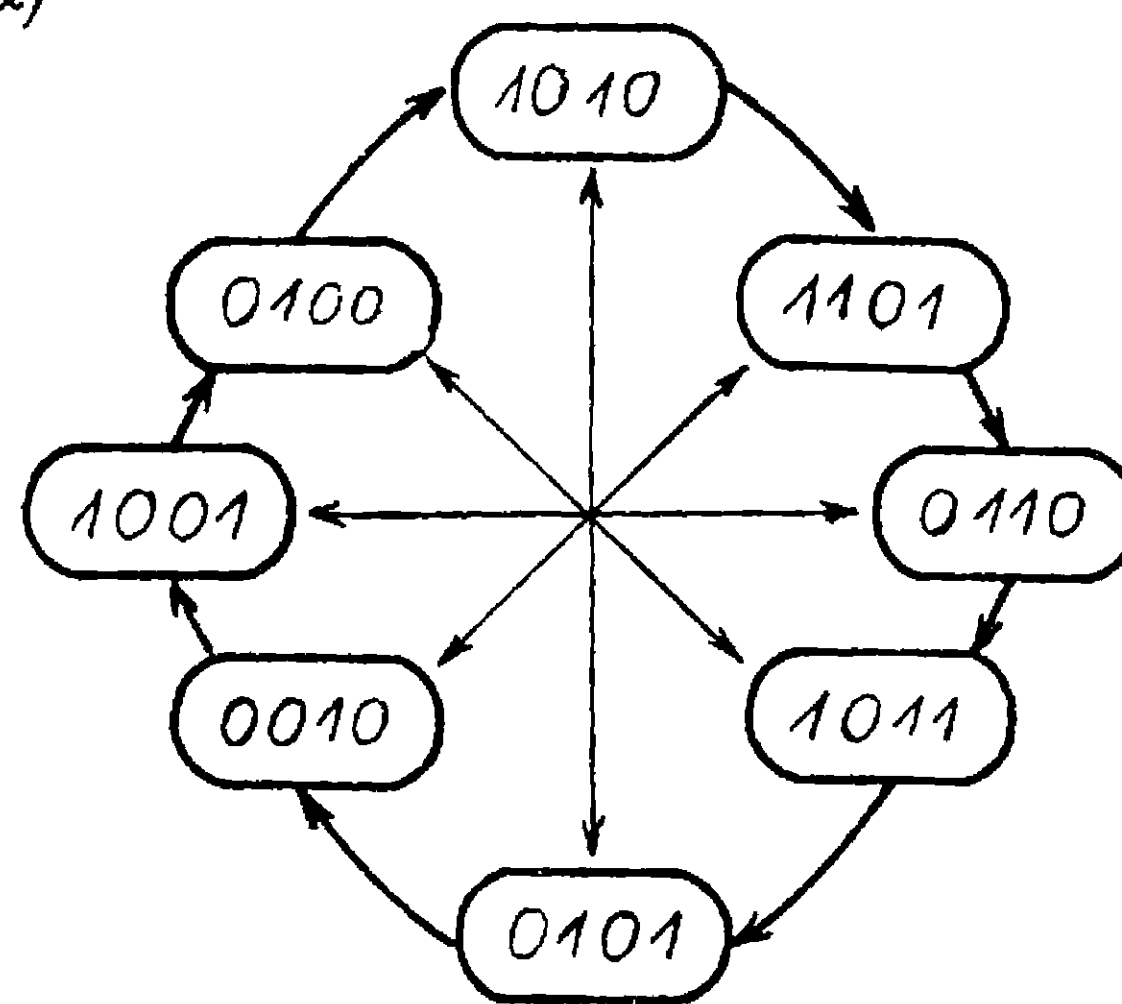
b)



c)

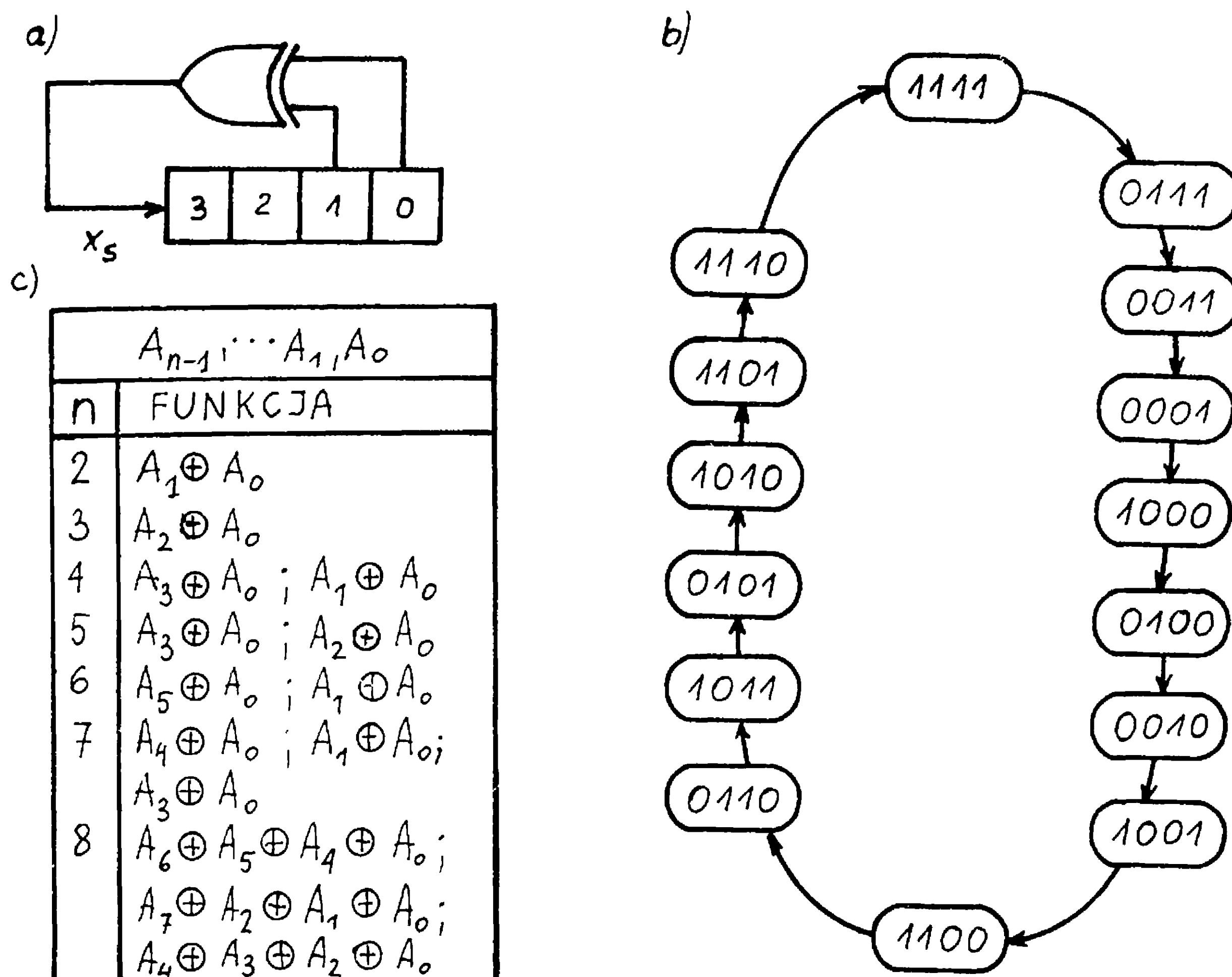


d)



Rys. 5.36. Ogólny schemat licznika z funkcyjnym sprzężeniem wyjścia z wejściem oraz przykład realizacji z zastosowaniem inwertera (a,b,c,d)

Najdłuższe cykle pracy licznikowej rejestru uzyskuje się stosując w sprzężeniu formułę EXCLUSIVE-OR (nierównoważności). Na rysunku 5.37c przedstawiono tabelę argumentów operacji XOR dla różnych długości rejestru, gwarantującą maksymalny cykl pracy licznikowej [6]. Rejestr n -bitowy, wygeneruje graf zawierający $2^n - 1$ różnych stanów. Stanem niedozwolonym jest stan $00...0$, z którego nie ma wyjścia z uwagi na zerową wartość formuły EXCLUSIVE-OR dla zerowych argumentów. Na rysunku 5.37a,b przedstawiono układ dla rejestru czterobitowego oraz graf tego układu. Rejestry o większych długościach ze sprzężeniami nierównoważnościowymi są wykorzystywane jako generatory liczb pseudolosowych o rozkładzie zbliżonym do równomiernego. Dystrybucja takich rozkładów jest linią prostą i dlatego liczniki te nazywa się często rejestrami liniowymi. Zagadnienia te są szerzej omówione w literaturze poświęconej konstruowaniu generatorów liczb pseudolosowych.



Rys. 5.37. Rejestr ze sprzężeniem nierównoważnościowym (a), jego graf (b) oraz tabela argumentów operacji EXCLUSIVE-OR dla różnych długości rejestru (c)

5.6. Projektowanie synchronicznych układów sekwencyjnych

Projektowanie układu synchronicznego sprowadza się do dobrania liczby i rodzaju przerzutników, sformułowania warunków wejściowych i wyjściowych dla bloku kombinacyjnego i optymalnego zaprojektowania go metodami omówionymi w rozdziale 3. Jak już powiedziano w punkcie 5.1, wejściami bloku kombinacyjnego są wejścia zewnętrzne reprezentowane przez wektor X o m składowych oraz wyjścia przerzutników reprezentowane przez wektor stanów aktualnych S o k składowych. Warunki wyjściowe są sformułowane dla wyjść zewnętrznych reprezentowanych przez wektor Y o n składowych oraz dla wyjść realizujących formuły sterujące

wejścia przerzutników i reprezentowane przez wektor Z o l składowych. Liczby składowych poszczególnych wektorów k, l, m, n , należą do zbioru liczb naturalnych powiększonego o 0 dla uwzględnienia przypadków braku wejść lub wyjść określonego rodzaju (zerowa wymiarowość wektora). Projektowanie wygodnie jest przeprowadzać w oparciu o tablicę pokazaną na rysunku 5.38. Zawiera ona te same informacje co tablica przejść - wyjść, powiększone o kolumnę wektora Z , który jest często określony jako wektor wzbudzeń przerzutników. Dla odróżnienia tej tablicy od tablicy przejść - wyjść (tablicy charakterystycznej) nazywa się ją często tablicą wzbudzeń układu synchronicznego. Wartości poszczególnych składowych wektora Z wyznacza się w oparciu o tablice wzbudzeń zastosowanych przerzutników. Można w tym celu posłużyć się zbiorczą tabelą pokazaną na rysunku 5.39. Układ uważany za zaprojektowany, jeśli są określone formuły wektorowe dla wyjść bloku kombinacyjnego:

$$Z = f_Z (X, S^t),$$

oraz

$$Y = f_Y (X, S^t).$$

(1)

Wejścia U.K.	Stany następne	Wyjścia U.K.	
X, S^t	S^{t+1}	Z	Y
.	.	.	.
.	.	.	.
.	.	.	.

Rys. 5.38. Tablica wzbudzeń układu synchronicznego

Na ich podstawie można jednoznacznie określić schemat logiczny układu. Po dobraniu przerzutników tworzących blok pamięci (rys. 5.1) projektowanie sprowadza się więc do zaprojektowania najczęściej wielowyjściowego układu kombinacyjnego (rozdz. 3).

Ważną rolę w procesie projektowania odgrywa formalizacja zagadnienia projektowego (patrz p. 3.1). W zasadzie nie ma ścisłych reguł algorytmizujących ten proces z uwagi na szeroką różnorodność rozwiązywanych zagadnień. Dla jednych przypadków proces syntezy projektowej będzie przebiegał poprzez poszukiwanie elementów minimalnych, w in-

a)

S^t	X		S^{t+1}	Z		Y	
A_t	x_1	x_2	A_{t+1}	JA	KA	y_1, y_2	
0	0	0	0	0	X	1	1
0	0	1	0	0	X	0	1
0	1	0	1	1	X	1	0
0	1	1	0	0	X	1	1
1	0	0	1	X	0	0	1
1	0	1	0	X	1	0	1
1	1	0	1	X	0	0	0
1	1	1	1	X	0	1	0

b)

$x_1 x_2$		00	01	11	10
A_t	0	0	0	0	1
	1	X	X	X	X

JA

$x_1 x_2$		00	01	11	10
A_t	0	X	X	X	X
	1	0	1	0	0

KA

$x_1 x_2$		00	01	11	10
A_t	0	1	0	1	1
	1	0	0	1	0

y_1

$x_1 x_2$		00	01	11	10
A_t	0	1	1	1	0
	1	1	1	0	0

y_2

Rys. 5.41. Tablica wzbudzeń (a) i tablice Karnaugh układu pokazanego na rys. 5.40 (b)

Układ posiada dwa wejścia zewnętrzne, dwa wyjścia i dwa stany wewnętrzne. Do realizacji grafu wystarczy jeden przerzutnik. Wybierzmy przerzutnik typu JK i oznaczmy zmienne wejściowe przez x_1, x_2 , a wyjścia przez y_1, y_2 . Wektor X będzie więc miał dwie składowe, wektor Y również dwie. Wektor wzbudzeń musiysterować dwa wejścia JK. Tablica wzbudzeń będzie mieć postać pokazaną na rysunku 5.41. Kolumnę wzbudzeń wypełniamy zgodnie z tabelą pokazaną na rysunku 5.39. Na podstawie tablic Karnaugh dla wszystkich czterech wyjść (rys. 5.40b) otrzymamy następujące formuły minimalne:

$$JA = x_1 \bar{x}_2, \quad KA = \bar{x}_1 x_2,$$

$$y_1 = x_1 x_2 + \bar{A} \bar{x}_2,$$

$$y_2 = \bar{x}_1 + \bar{A} x_2.$$

Schemat logiczny układu przedstawiono na rysunku 5.40a.

Przykład 5.2. Zaprojektować układ opisany następującymi równaniami stanów:

$$\begin{aligned}A_{t+1} &= A_t + x, \\B_{t+1} &= \bar{B}_t + \bar{A}_t \bar{x}, \\y &= (\bar{A} + B)x.\end{aligned}$$

Przekształćmy te równania w ten sposób, by można było zastosować przerzutniki typu T:

$$\begin{aligned}A_{t+1} &= A_t + x = A_t + \bar{A}_t x = A_t (A_t + \bar{x}) + \bar{A}_t \bar{A}_t x = \\&= A_t \overline{(\bar{A}_t x)} + \bar{A}_t (\bar{A}_t x) = A_t \oplus (\bar{A}_t x),\end{aligned}$$

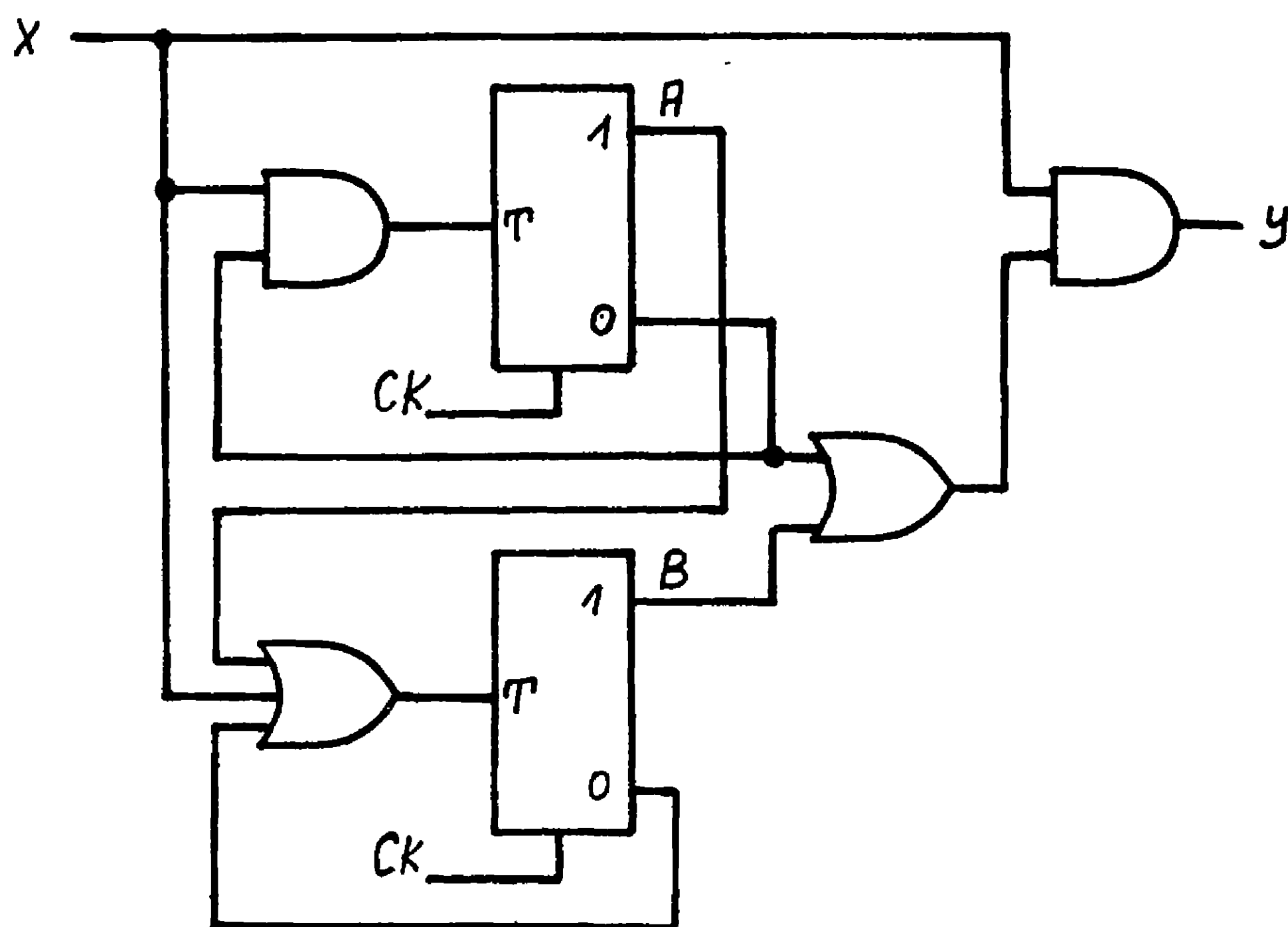
a więc $TA = \bar{A} x$,

$$\begin{aligned}B_{t+1} &= \bar{B}_t + \bar{A}_t \bar{x} = \bar{B}_t + \bar{A}_t B_t \bar{x} = \bar{A}_t B_t \bar{x} + A_t \bar{B}_t + \bar{B}_t = \\&= B_t \bar{A}_t B_t \bar{x} + A_t \bar{B}_t + \bar{B}_t x + \bar{B}_t = \\&= B_t (A_t + \bar{B}_t + x) + \bar{B}_t (A_t + \bar{B}_t + x) = B_t \oplus (A_t + \bar{B}_t + x),\end{aligned}$$

a więc $TB = A + \bar{B} + x$.

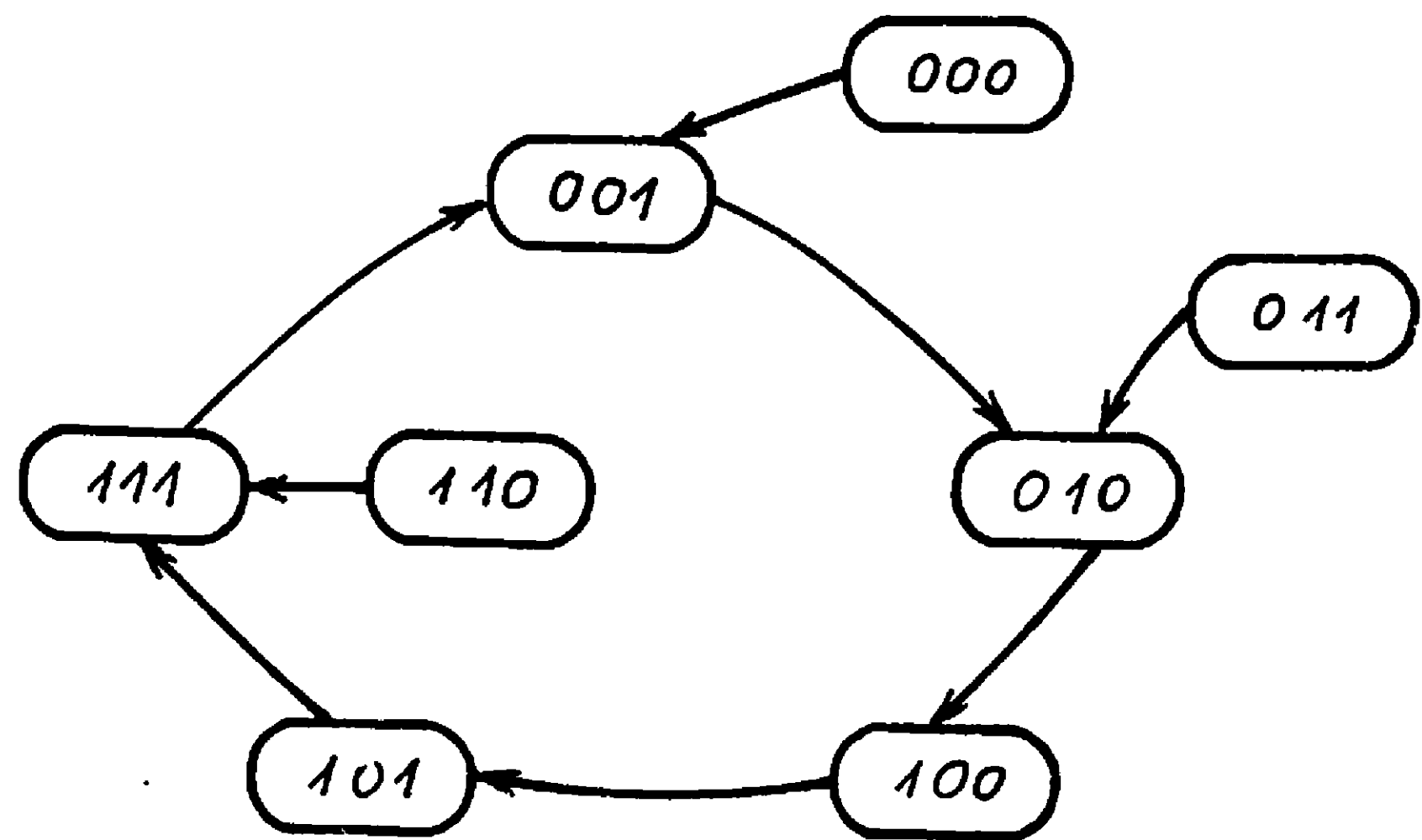
Otrzymane równania pozwalają zbudować układ, którego schemat pokazano na rysunku 5.42. Był to przykład typowo analitycznego podejścia. Inna metoda polega na zbudowaniu w oparciu o równania stanów tablicy charakterystycznej i następnie tablicy wzbudzeń. Dalej tok projektowania przebiegałby identycznie jak w przykładzie 5.1. Wybór innego typu przerzutnika wymaga dokonania innych przekształceń. Wyjątkiem jest przerzutnik typu D, dla którego jak łatwo zauważyć, równania stanów są jednocześnie równaniami wejść, a więc można przystąpić bezpośrednio do rysowania schematu logicznego z równań stanów.

Przykład 5.3. Zaprojektować licznik, o następującej sekwencji zmian stanów: 001, 010, 100, 101, 111, 001, ...



Rys. 5.42. Schemat układu z przykładu 5.2

Na wstępie należy zauważyć, że trójwymiarowy wektor stanów wewnętrznych ma 8 wartości, z których w zadaniu wykorzystano tylko 5. Pozostałe trzy: 000, 011, 110 są niewykorzystane. W tablicy charakterystycznej i tablicy wzbudzeń stanom tym można przypisać wartości nieokreślone i nie zmieni to funkcji układu. Postępowanie takie w odniesieniu do przerzutników jest jednak błędem konstrukcyjnym, bowiem zakłócenia mogą zawsze wprowadzić przerzutnik w stan nieprzewidziany i projektant musi podać mechanizm ponownego wejścia układu w cykl normalnej pracy i to w możliwie jak najkrótszym czasie. W omawianym przypadku najprościej można tego dokonać w sposób przedstawiony na rysunku 5.43. Przypadkowe wejście w stan niedozwolony powoduje przejście regeneracyjne już w następnym cyklu zegarowym. Wybierając do realizacji przerzutniki typu T, otrzymamy tabelę wzbudzeń pokazaną na rys. 5.44a. Schemat połączeń otrzymamy na podstawie tabeli pokazano na rysunku 5.44b. Funkcje wejściowe przerzutników uzyskano minimalizując zespół funkcji T_A , T_B , T_C metodą omówioną w punkcie 3.7. Dla przejrzystości rysunku, podano jedynie nazwy sygnałów sterujących odpowiednimi wejściami bramek drugiego poziomu.

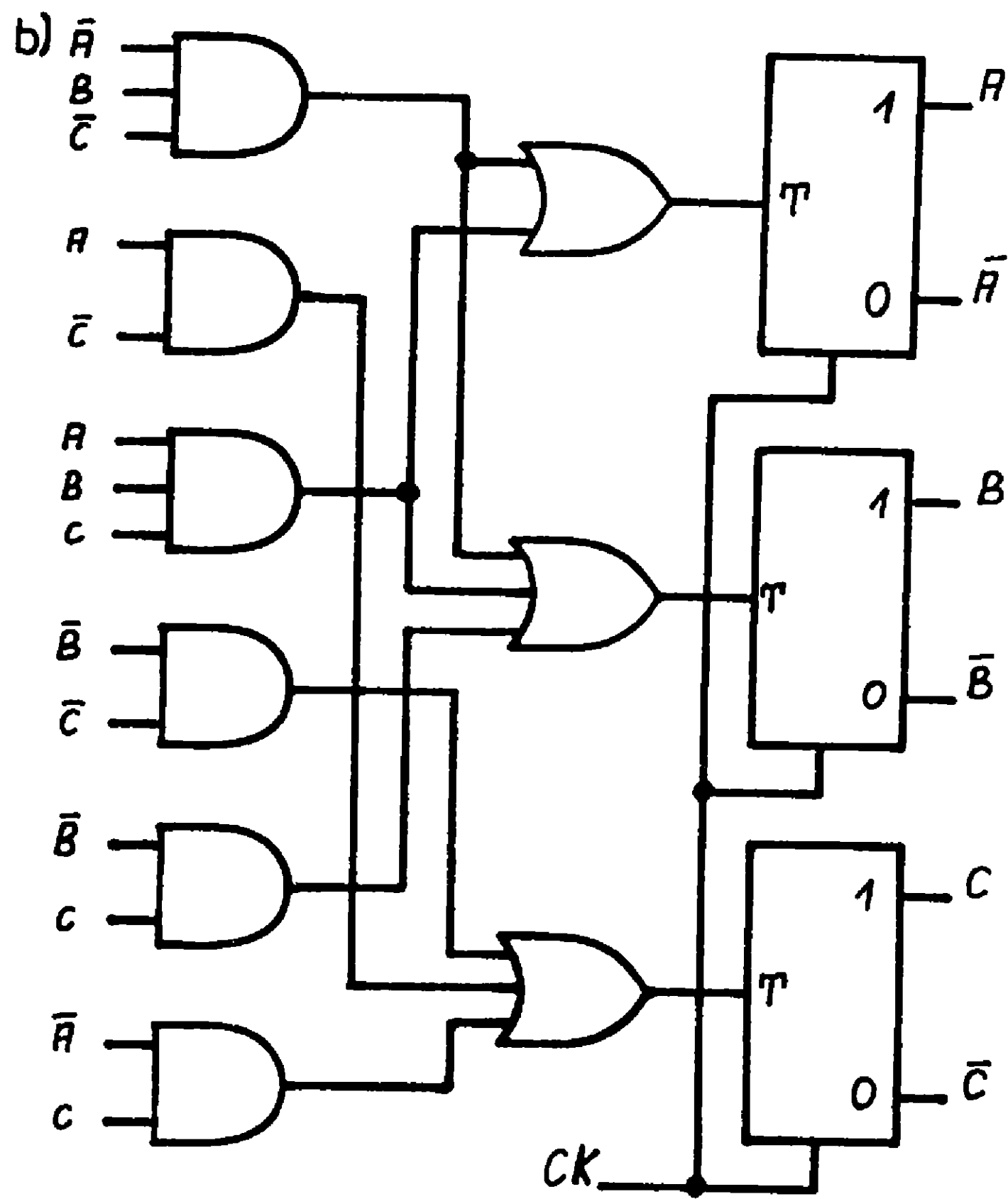


Rys. 5.43. Graf układu do zadania z przykładu 5.3

a)

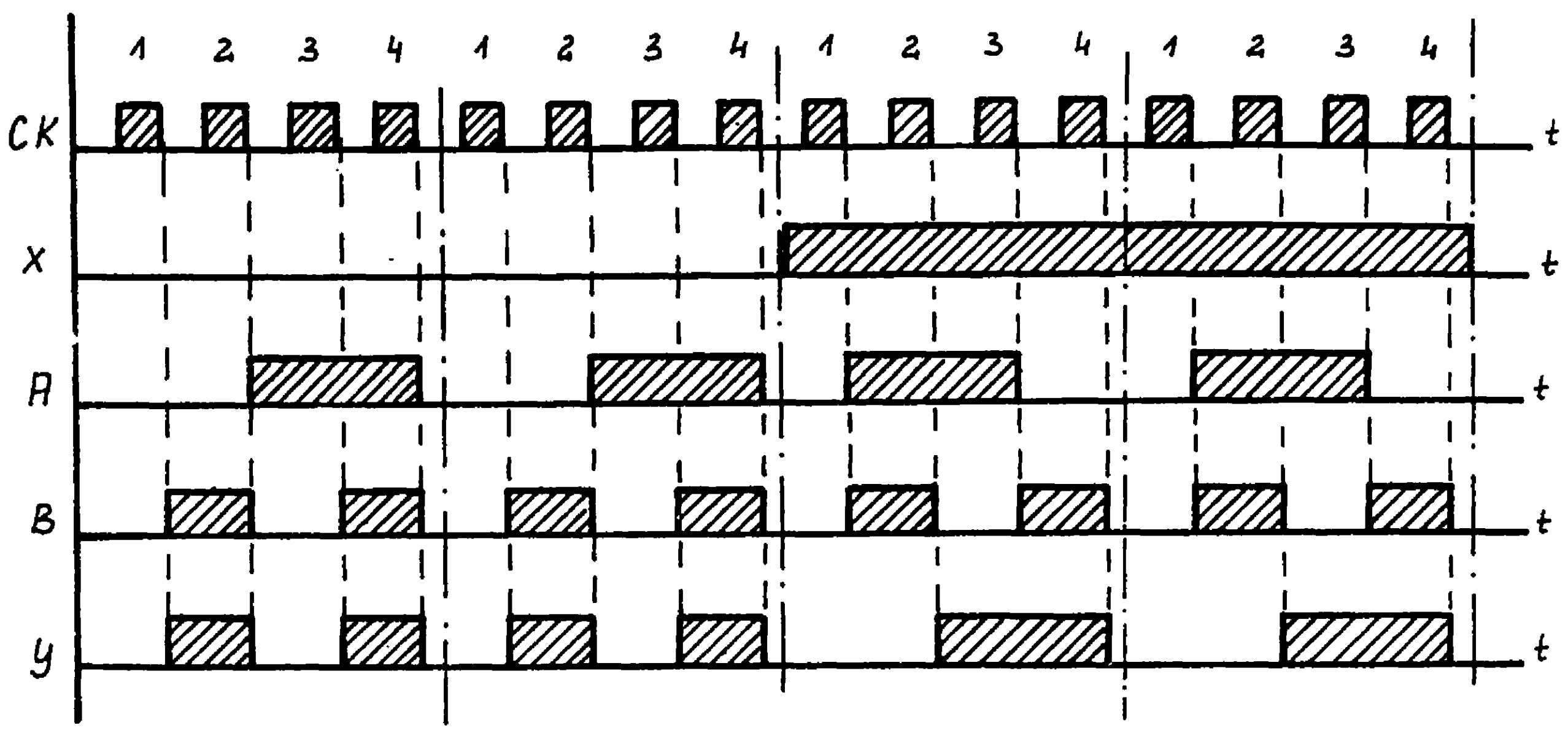
A_t	B_t	C_t	A_{t+1}	B_{t+1}	C_{t+1}	TA	TB	TC
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	1	0	0	1	1	0
0	1	1	0	1	0	0	0	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	1	0	1	0
1	1	0	1	1	1	0	0	1
1	1	1	0	0	1	1	1	0

b)



Rys. 5.44. Tablica wzbudzeń (a) i schemat układu z przykładu 5.3 (b)

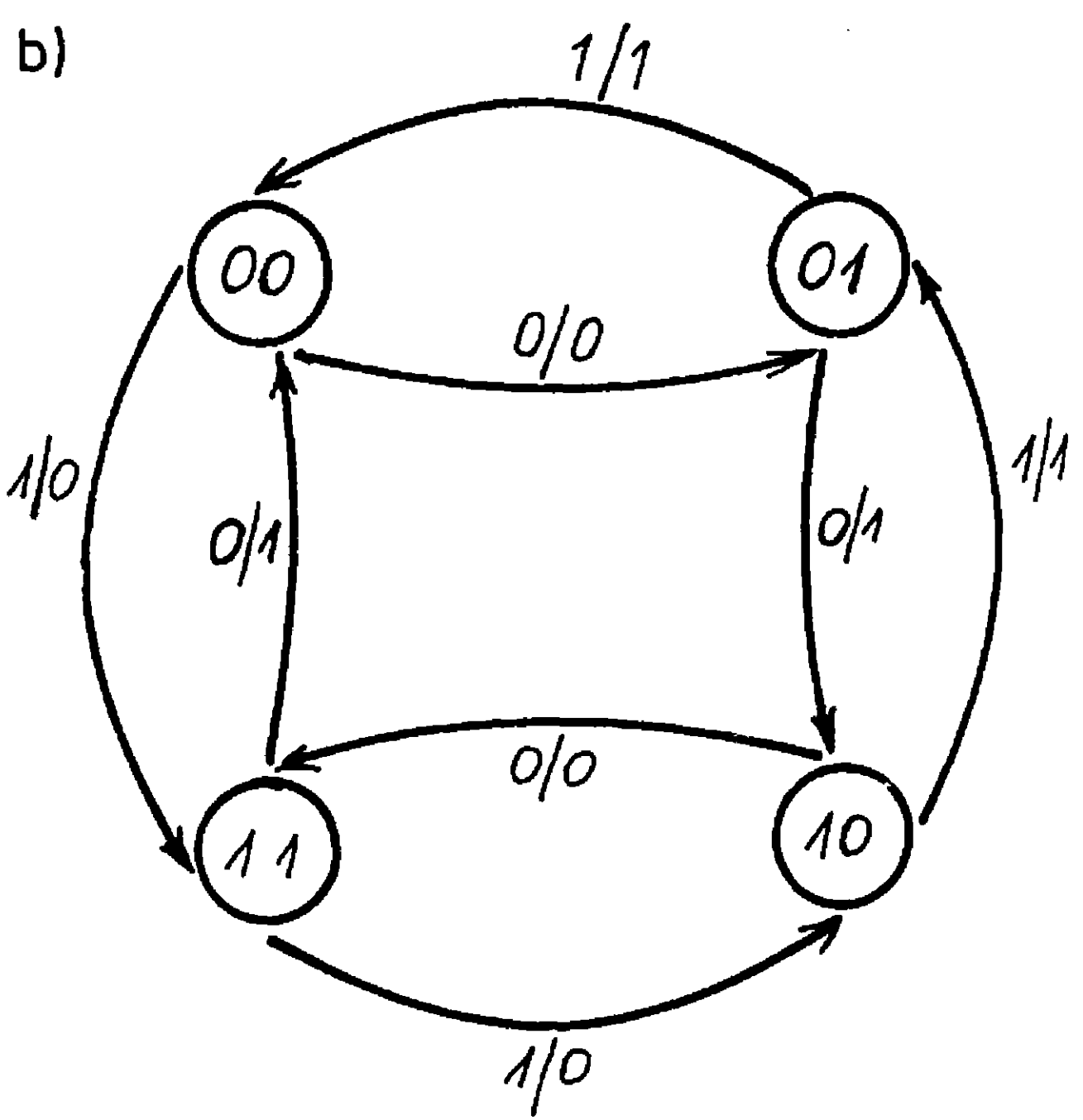
Przykład 5.4. Zaprojektować układ pracujący zgodnie z wykresem czasowym pokazanym na rysunku 5.45.



Rys. 5.45. Wykresy czasowe do przykładu 5.4

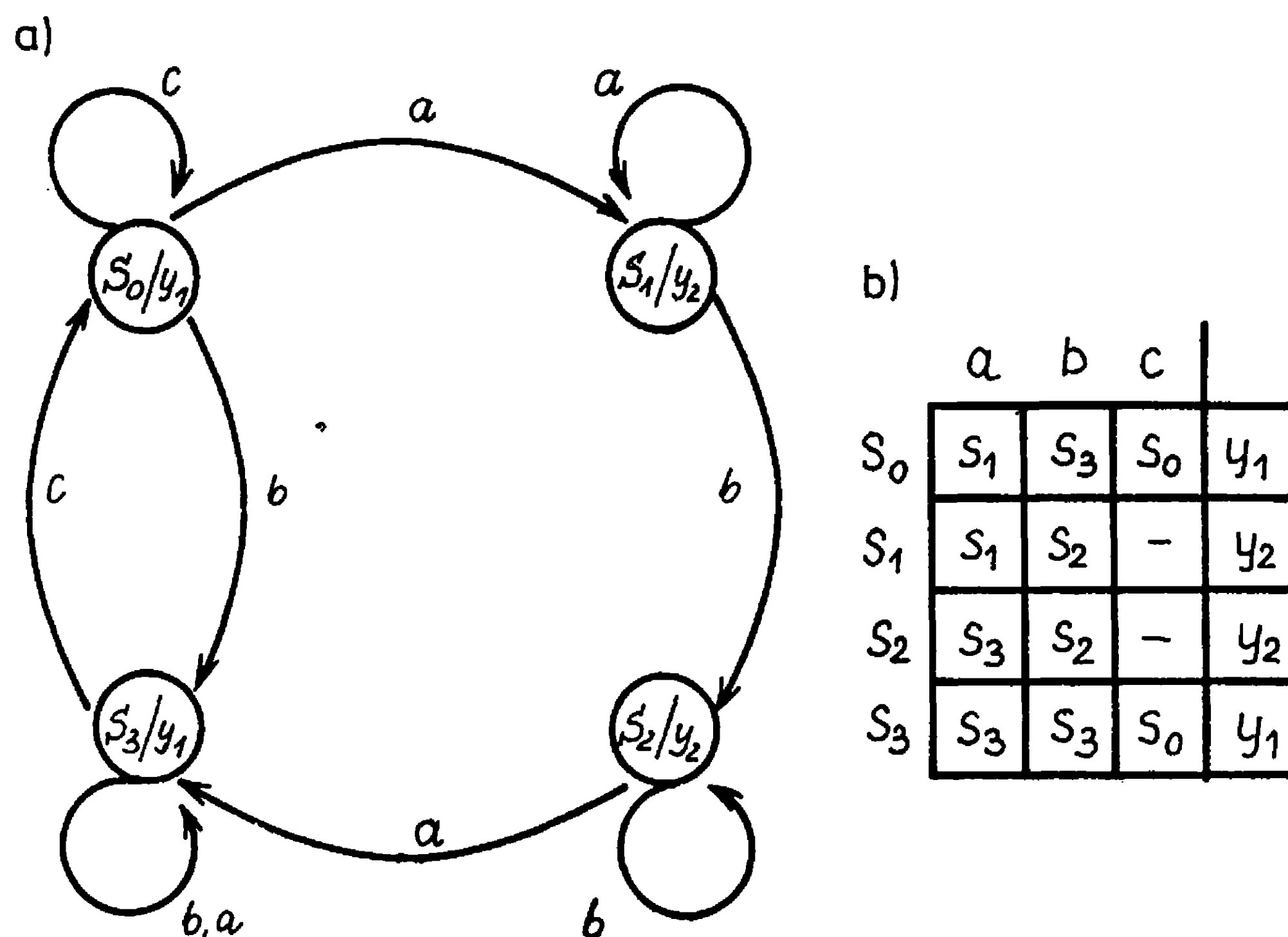
a)

A_t	B_t	x	A_{t+1}	B_{t+1}	TA	TB	y
0	0	0	0	1	0	1	0
0	0	1	1	1	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	0	1	1
1	0	0	1	1	0	1	0
1	0	1	0	1	1	1	1
1	1	0	0	0	1	1	1
1	1	1	1	0	0	1	0



Rys. 5.46. Tablica wzbudzeń (a) i graf układu z przykładu 5.4 (b)

Analizując wykres można zauważyć, że układ pracuje cyklicznie z okresem wynoszącym 4 impulsy zegara, przy czym inny cykl występuje dla



Rys. 5.48. Graf (a) i tablica przejść - wyjść automatu z przykładu 5.5 (b)

literach jest najmniejsza, zbiór staje się ograniczony. W omawianym przykładzie, aby zakodować alfabet wewnętrzny $S = \{s_0, s_1, s_2, s_3\}$ wystarczą dwa bity wektora stanu 00, 01, 11, 10, a więc w konsekwencji dwa przerzutniki. Alfabet wyjściowy liczy dwie litery: $Y = \{y_1, y_2\}$; do ich rozróżnienia wystarczy więc tylko jedno wyjście binarne przyjmujące wartość 0 i 1. Alfabet wejściowy zawiera trzy litery $X = \{a, b, c\}$, wektor wejściowy musi więc mieć dwie składowe, a więc jedna jego wartość nie będzie wykorzystana. W ramach klasy układów o minimalnych reprezentacjach liter automatu wektorami binarnymi, wybór wartości określonego wektora dla zakodowania litery, prowadzi na ogół do realizacji różnych bloków kombinacyjnych układu sekwencyjnego. Wykazano, że proces kodowania można przeprowadzić w sposób taki, by otrzymany blok kombinacyjny miał możliwie mały wskaźnik kosztu. Zagadnieniami tymi zajmowali się Hartmanis i Stearns [2,6]. Tematyka ta była szczególnie użyteczna wówczas, gdy bramki logiczne wykonywano techniką dyskretną. Obecnie, gdy układy scalone zawierają zestawy bramek, korzyści z optymalnego podziału są niewielkie, ale sama metoda jest interesującą z punktu widzenia poznawczego. Jest ona jednocześnie dość złożona, dlatego tutaj problem optymalnego kodowania jest tylko sygnalizowany.

a)

	00	01	10	11	
00	01	11	00	—	0
01	01	10	—	—	1
10	11	10	—	—	1
11	11	11	00	—	0

b)

A_t	B_t	x_1	x_2	A_{t+1}	B_{t+1}	RA	SA	RB	SB	y
0	0	0	0	0	1	X	0	0	1	0
0	0	0	1	1	1	0	1	0	1	0
0	0	1	0	0	0	X	0	X	0	0
0	0	—	—	—	—	X	X	X	X	0
0	1	0	0	0	1	X	0	0	X	1
0	1	0	1	1	0	0	1	1	0	1
0	1	1	0	—	—	X	X	X	X	1
0	1	—	—	—	—	X	X	X	X	1
1	0	0	0	1	1	0	X	0	1	1
1	0	0	1	1	0	0	X	X	0	1
1	0	1	0	—	—	X	X	X	X	1
1	0	—	—	—	—	X	X	X	X	1
1	1	0	0	1	1	0	X	0	X	0
1	1	0	1	1	1	0	X	0	X	0
1	1	1	0	0	0	1	0	1	0	0
1	1	—	—	—	—	X	X	X	X	0

Rys. 5.49. Tablica przejść - wyjść (a) oraz tablica wzbudzeń z przykładu 5.5 (b)

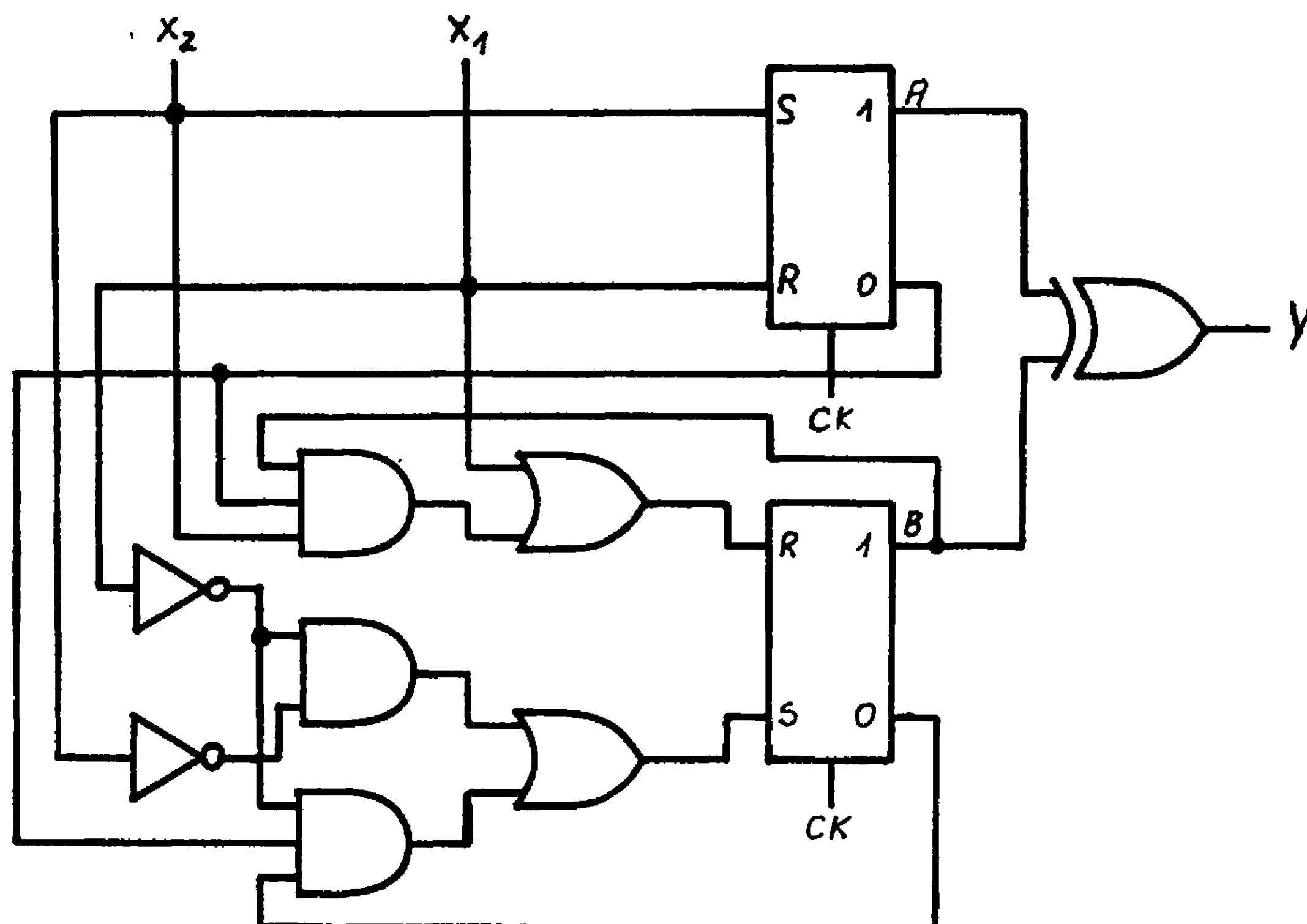
Na rysunku 5.49a przedstawiono tablicę przejść - wyjść rozpatrywanego automatu zakodowaną arbitralnie wg schematu: s_0 - 00, s_1 - 01, s_2 - 10, s_3 - 11, a - 00, b - 01, c - 10, y_1 - 0, y_2 - 1. Dodatkowo umieszczono kolumnę odpowiadającą niewykorzystanej wartości wektora wejściowego. Oznaczając literami A, B wyjścia przerzutników i literami x_1 , x_2 składowe wektora wejściowego oraz przyjmując do realizacji przerzutniki typu R S, otrzymamy tabelę wzbudzeń pokazaną na rysunku 5.49b. Część wierszy w kolumnie stanów następnych jest nieokreślona. Fakt ten można potraktować różnie, gdyż za pomocą zmiennych zewnętrznych układ daje się zawsze sprowadzić do pracy nominalnej, a więc zakłócenia mogą być neutralizowane. Najwygodniej więc jest stany te potraktować jako nieokreślone, co upraszcza blok kombinacyjny. Otrzymamy wówczas następujące równania wejść przerzutników i wyjścia y:

$$RA = x_1, \quad SA = x_2,$$

$$RB = x_1 + \bar{A} B x_2, \quad SB = \bar{x}_1 \bar{x}_2 + \bar{A} \bar{B} \bar{x}_1,$$

$$y = \bar{A} B + A \bar{B} = A \oplus B.$$

Schemat układu pokazano na rysunku 5.50. Należy zwrócić uwagę na to, że zrealizowany układ jest automatem typu Moore'a, gdyż wyjście y zależy bezpośrednio tylko od wektora stanów wewnętrznych.



Rys. 5.50. Schemat układu z przykładu 5.5

Przykład 5.6. Proces technologiczny podlega ośmiu cyklom operacyjnym i jest sterowany sygnałem zewnętrznym o wartościach a, b, c . Przejścia między cyklami są uwarunkowane zgodnie z tabelą pokazaną na rysunku 5.51a. Dla wybranych cykli i określonych wartości sygnału wejściowego, wyjście przyjmuje trzy poziomy sygnałizacji zgodnie z tabelą pokazaną na rysunku 5.51b. Zaprojektować układ sygnalizacji.

Tabele wymienione w zadaniu mogą być potraktowane łącznie jako tabela przejść - wyjść automatu. Bezpośrednie kodowanie ciągami zero-jedynkowymi liter tego automatu jest niewskazane, gdyż najpierw należy zastanowić się nad tym, czy nie istnieje automat minimalny realizujący te same funkcje wyjściowe, a więc pokrywający automat zdefiniowany poprzez treść zadania. Rodzina maksymalnych zbiorów stanów niesprzecznych wyznaczona na podstawie tablicy przejść - wyjść będzie następująca (patrz p. 4.3):

$$\{Q_{\max}\} = \left\{ \{1,3,4\}, \{2,6\}, \{5,8\}, \{1,2\}, \{1,5\}, \{1,7\} \right\}.$$

Do pełnego pokrycia wszystkich stanów automatu pierwotnego nie są potrzebne wszystkie zbiory. Odrzucając elementy $\{1,2\}, \{1,5\}$ otrzymamy następującą rodzinę zbiorów stanów niesprzecznych

$$\{P_j\} = \left\{ \{1,3,4\}, \{2,6\}, \{5,8\}, \{1,7\} \right\}.$$

a)	a	b	c	b)	a	b	c
1	—	2	—	1	—	y ₂	y ₁
2	5	1	6	2	y ₃	—	y ₁
3	4	6	1	3	y ₁	—	—
4	4	—	7	4	y ₁	—	y ₁
5	1	2	8	5	y ₂	y ₂	—
6	8	3	2	6	y ₃	y ₁	y ₁
7	5	6	8	7	y ₃	y ₂	y ₁
8	7	6	5	8	y ₂	—	y ₃

Rys. 5.51. Tablice do przykładu 5.6 (a,b)

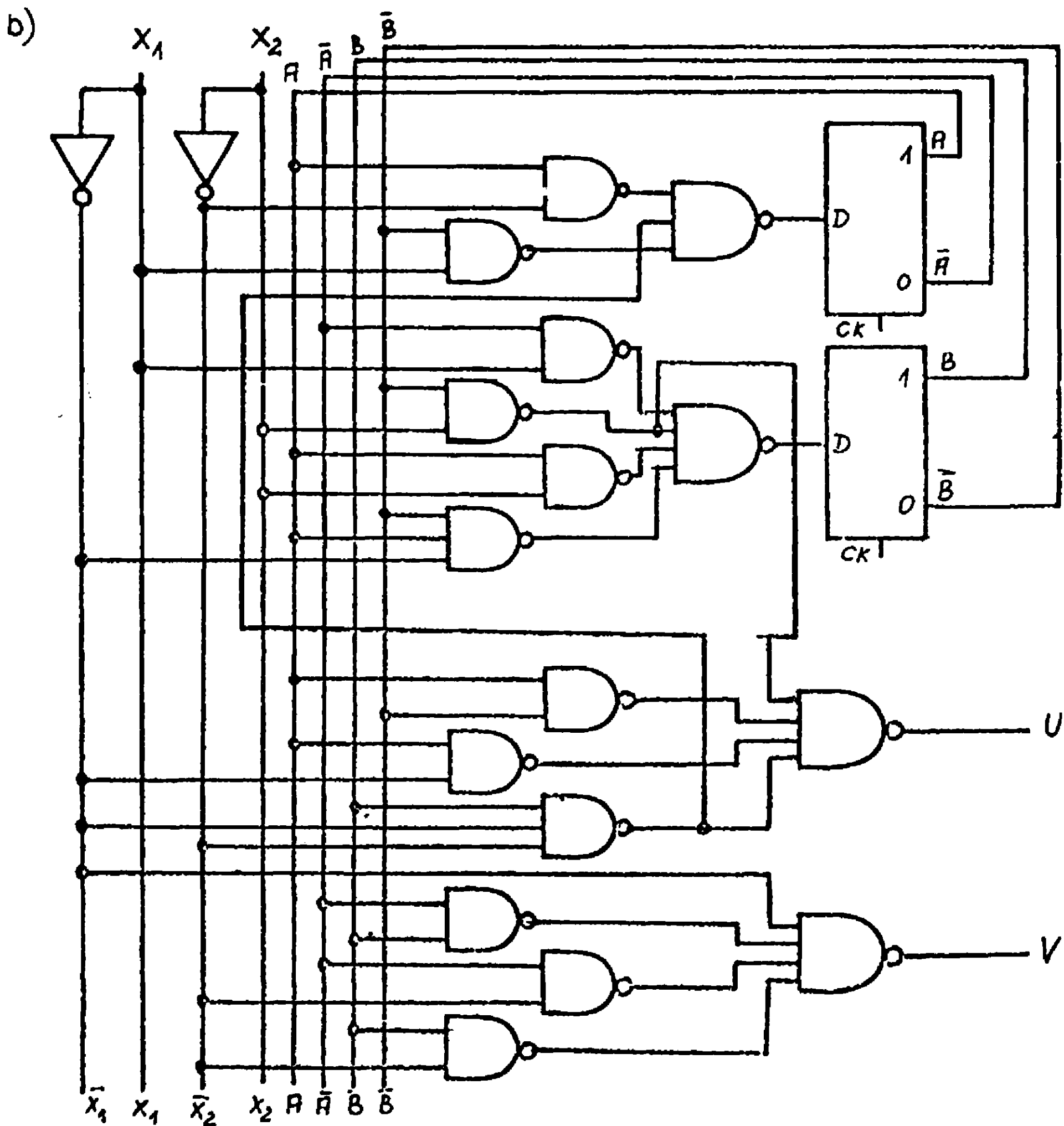
a)	a	b	c	a	b	c	b)	00	01	10	00	01	10
p ₁ (1,3,4)	p ₁	p ₂	p ₄	y ₁	y ₂	y ₁	00	00	01	11	01	10	01
p ₂ (2,6)	p ₃	p ₁	p ₂	y ₃	y ₁	y ₁	01	10	00	01	11	01	01
p ₃ (5,8)	p ₄	p ₂	p ₃	y ₂	y ₂	y ₃	10	11	01	10	10	10	11
p ₄ (1,7)	p ₃	p ₂	p ₃	y ₃	y ₂	y ₁	11	10	01	10	11	10	01

Rys. 5.52. Tablica przejść - wyjść automatu minimalnego i odpowiadająca jej tablica układu logicznego (przykład 5.6), (a,b)

Łatwo sprawdzić, że rodzina ta spełnia zarówno warunek zamkniętości, jak i pokrycia (p. 4.4), jest więc rodziną minimalną $\{P_j\}$. Na rysunku 5.52 przedstawiono tabelę przejść - wyjść automatu ilorazowego (minimalnego) $A/\{P_j\}$ określonego dla tej rodziny i tabelę zakodowaną na-

a)

A_t	B_t	x_1	x_2	A_{t+1}	B_{t+1}	DA	DB	u	v
0	0	0	0	0	0	0	0	0	1
0	0	0	1	0	1	0	1	1	0
0	0	1	0	1	1	1	1	0	1
0	0	X	X	X	X	X	X	X	X
0	1	0	0	1	0	1	0	1	1
0	1	0	1	0	0	0	0	0	1
0	1	1	0	0	1	0	1	0	1
0	1	X	X	X	X	X	X	X	X
1	0	0	0	1	1	1	1	1	0
1	0	0	1	0	1	0	1	1	0
1	0	1	0	1	0	1	0	1	1
1	0	X	X	X	X	X	X	X	X
1	1	0	0	1	0	1	0	1	1
1	1	0	1	0	1	0	1	1	0
1	1	1	0	1	0	1	0	0	1
1	1	X	X	X	X	X	X	X	X



Rys. 5.53. Tablica wzbudzeń (a) i schemat układu z przykładu 5.6 (b)

stępująco: $p_1 - 00$, $p_2 - 01$, $p_3 - 10$, $p_4 - 11$, $a - 00$, $b - 01$, $c - 10$, $y_1 - 01$, $y_2 - 10$, $y_3 - 11$ oraz tablicę wzbudzeń dla przerzutników typu D. Przyjmując do realizacji przerzutnik typu D na podstawie tabeli wzbudzeń pokazanej na rysunku 5.53 uzyskamy następujące równania dla wejść DA i DB oraz wyjść u , v :

$$DA = A \bar{x}_2 + \bar{B} x_1 + B \bar{x}_1 \bar{x}_2,$$

$$DB = \bar{A} x_1 + \bar{B} x_2 + A x_2 + A \bar{B} \bar{x}_1,$$

$$u = \bar{B} x_2 + A \bar{B} + A \bar{x}_1 + B \bar{x}_1 \bar{x}_2,$$

$$v = x_1 + \bar{A} B + \bar{A} \bar{x}_2 + B \bar{x}_2.$$

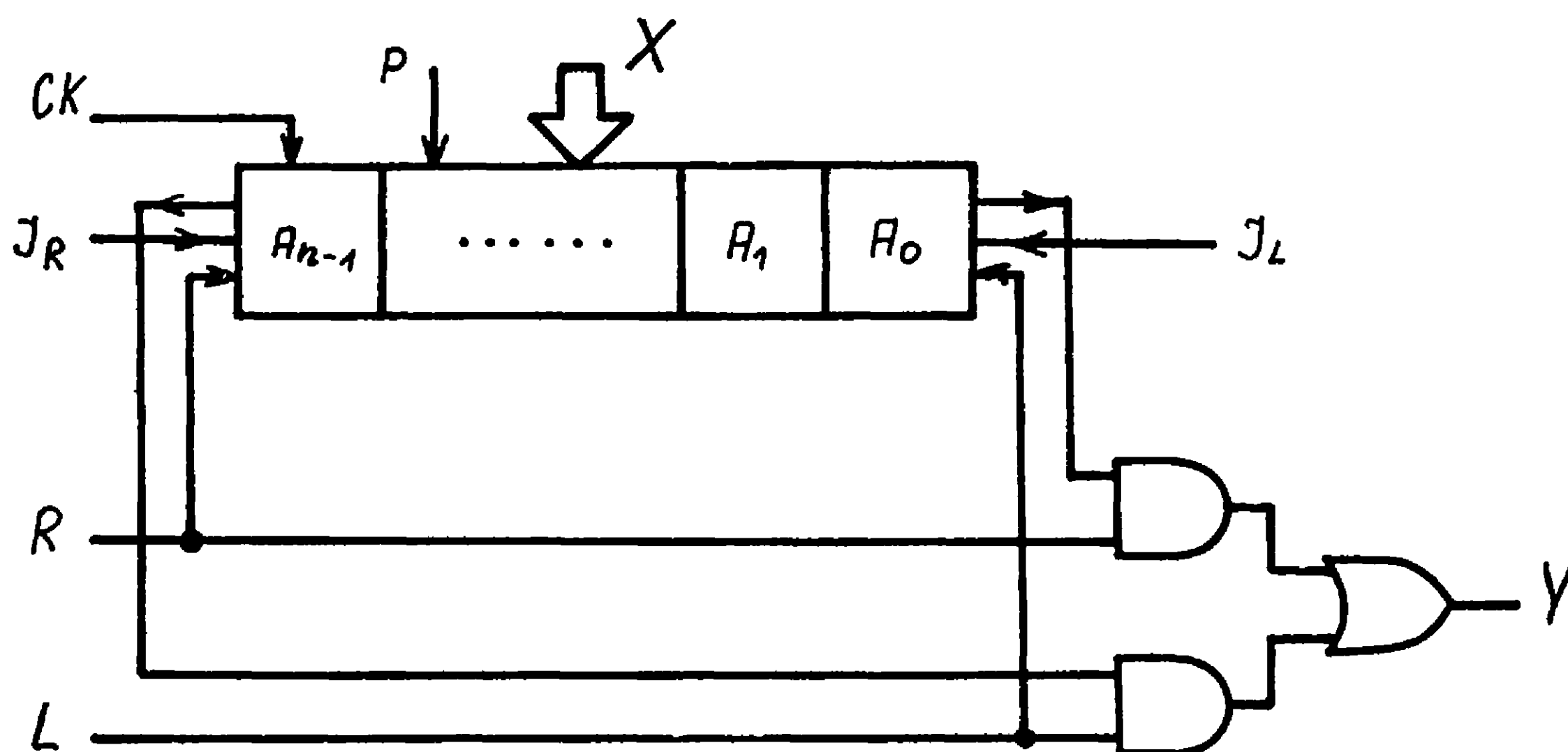
Schemat układu dostosowany do realizacji z bramek typu NAND przedstawiono na rysunku 5.53.

5.7. Iterowanie w czasie

Rejestry przesuwające umożliwiają rozłożenie informacji zawartej w sygnałach binarnych w czasie. Wektor X posiadający n składowych binarnych wpisany do rejestru n -bitowego, może być bowiem odczytany na wyjściu szeregowym w ciągu n kolejno po sobie następujących impulsów zegarowych. Dodatkowo, jeśli wyjście szeregowe rejestru jest połączone z wejściem przerzutnika pierwszego (licząc dla danego kierunku przesuwania), to informacja cyrkuluje w pętli charakterystycznej dla liczników pierścieniowych. Poddając kolejne bity wektora X przetwarzaniu kombinacyjnemu z odpowiednimi bitami innych wektorów, uzyskujemy rozłożony w czasie proces przekształceń funkcyjnych. Układy kombinacyjne służące do tego przetwarzania będą posiadały tylko po jednym wejściu binarnym dla każdego argumentu. Takie rozłożenie jednoczesnych wielowejściowych operacji logicznych na pojedyncze sekwencje czasowe nosi nazwę iterowania w czasie. Rezultat iteracji może być zapisany w rejestrze szeregowym bądź też może być wykorzystany w inny sposób.

Na rysunku 5.54 pokazano dwukierunkowy rejestr przesuwający z dołączonym układem kombinacyjnym, na którego wyjściu pojawia się sygnał odczytywany z rejestru w kolejnych sekwencjach czasowych, przy czym odczyt nie zależy od kierunku cyrkulacji informacji w rejestrze. Wek-

tor wejściowy X jest wpisywany równoległe wówczas, gdy sygnał $P = 1$. Sygnał P najczęściej posiada priorytet nad sygnałami przesuwania R oraz L . Przedstawiony rejestr jest klasycznym przykładem układu przetwarzającego informację równoległą, reprezentowaną przez wektor X , na informację szeregową odczytywaną z wyjścia y , przy czym pełny odczyt występuje w cyklu n po sobie następujących impulsów zegarowych. Rejestr ten może oczywiście również służyć do przetwarzania informacji podanej szeregowo na wejście I_R lub I_L , na informację równoległą, a więc dostępną jednocześnie na wszystkich jego bitach po n -impulsach zegarowych.



Rys. 5.54. Dwukierunkowy rejestr przesuwający z pojedynczym wyjściem szeregowym

Rozłożenie w czasie informacji zawartej w sygnałach binarnych upraszcza układy kombinacyjne, na wejścia których sygnały te są podawane. Jest to zrozumiałe, bowiem liczba wejść zostaje ograniczona do jednego dla każdego argumentu operacji. Przykładowo rozpatrzmy układ konwertera kodu Gray'a na kod binarny omówiony w punkcie 3.5. Przypisując indeksy wagowe kodu binarnego odpowiednim bitom kodu Gray'a, można dla czterech bitów napisać następujące formuły:

$$b_8 = g_8,$$

$$b_4 = g_8 \oplus g_4 = b_8 \oplus g_4,$$

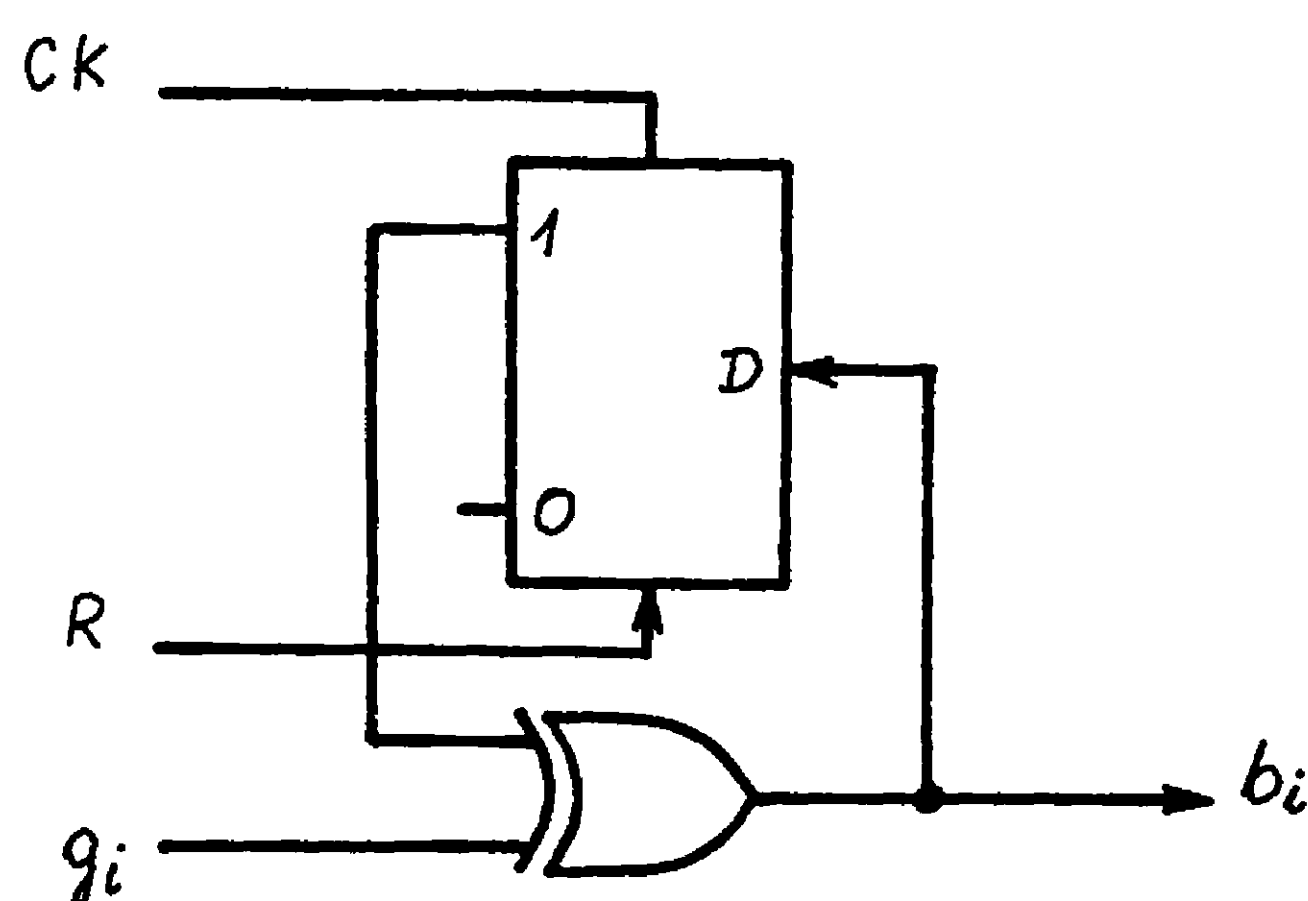
$$b_2 = g_8 \oplus g_4 \oplus g_2 = b_4 \oplus g_2,$$

$$b_1 = g_8 \oplus g_4 \oplus g_2 \oplus g_1 = b_2 \oplus g_1.$$

W ogólnym przypadku, otrzymamy:

$$b_i = b_{2i} \oplus g_i \quad (i = 1, 2, 4, \dots, 2^n),$$

przy czym $b_{2^{n+1}} = 0$. Zależność ogólną realizuje układ iteracyjny pokazany na rysunku 5.55. Składa się on z przerzutnika typu D oraz jednej bramki XOR, na którą jest bezpośrednio podawany sygnał g_i oraz z przerzutnika - sygnał b_{2i} . Kolejność sekwencji przebiega od bitów o najwyższej wadze. Zerowy warunek początkowy jest ustawiany wejściem R (reset). Tego typu iteracja, niezależnie od liczby bitów przetwarzanego kodu, może być wykonywana przez ten sam blok logiczny.



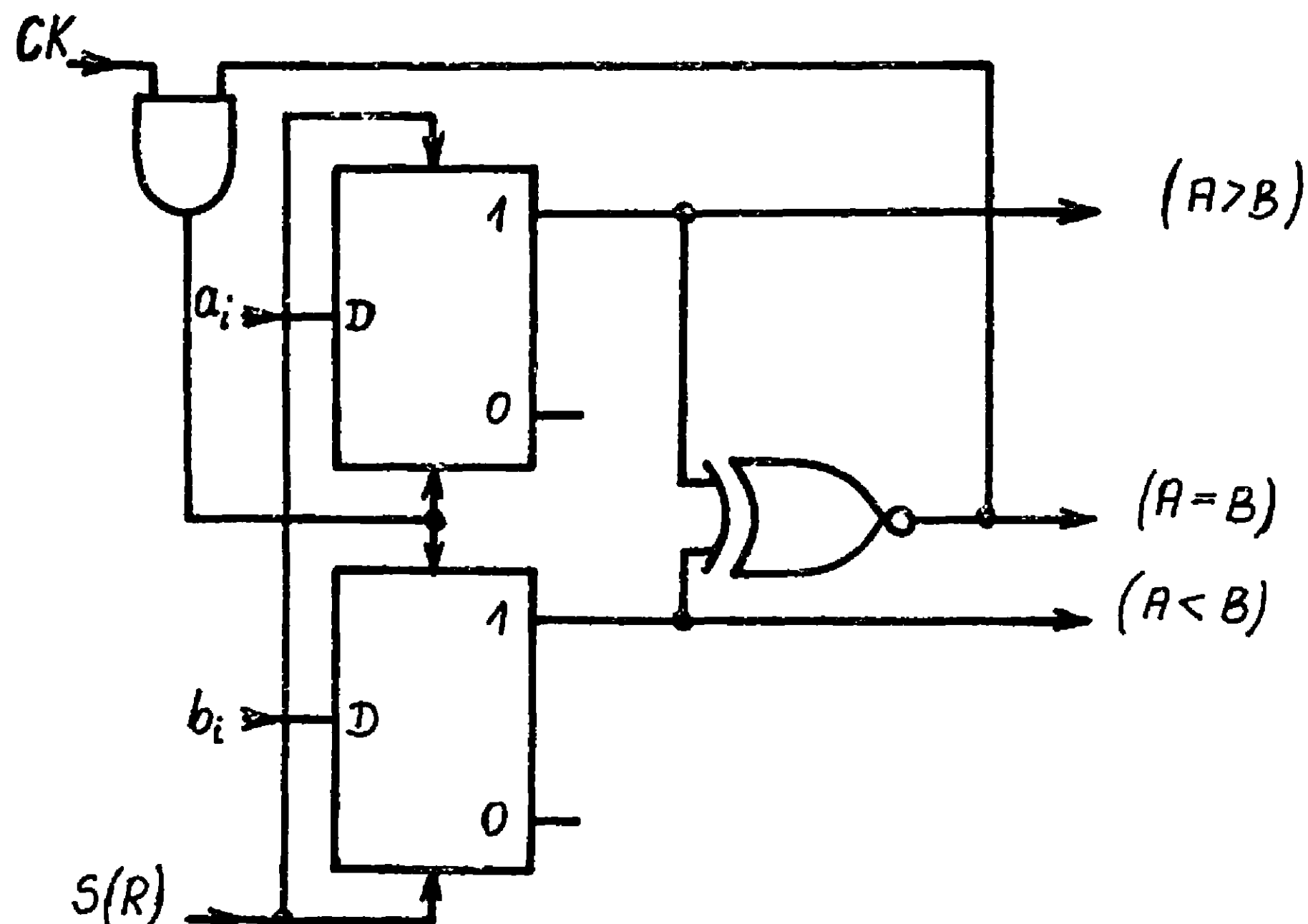
Rys. 5.55. Komórka do iteracji w czasie konwertera kodu Gray'a na kod binarny

Przykład 5.7. Zakładając, że sygnały sterujące są rozłożone w czasie określić schemat iterowanego w czasie komparatora dwóch liczb n -bitowych.

Poszczególne bity reprezentujące cyfry binarne porównywanych liczb $A(a_{n-1}, \dots, a_0)$, $B(b_{n-1}, \dots, b_0)$ muszą być podawane na wejścia układu kolejno od pozycji o najwyższej wadze. Na rysunku 5.56 pokazano schemat komórki iteracyjnej. Warunek początkowy - jednakowe stany na wyjściach obu przerzutników - jest ustawiany sygnałem S lub R. Na wyjściu bramki równoważności ($A = B$) jest wówczas sygnał 1 i impulsy CK dochodzą do wejść zegarowych przerzutników. Dopóki w kolejnych impulsach $a_i = b_i$, stan powyższy nie ulega zmianie. Jeśli po kolejnym impulsie $a_i \neq b_i$, jedynka pojawi się na wyjściach nierównościowych zgodnie ze schematem:

$$a_i > b_i \rightarrow (A > B) = 1, (A = B) = 0, (A < B) = 0,$$

$$a_i < b_i \rightarrow (A > B) = 0, (A = B) = 0, (A < B) = 1.$$



Rys. 5.56. Komórka iterowanego w czasie komparatora dwóch liczb n-bitowych (przykład 5.7)

Jednocześnie następuje trwała blokada impulsów zegarowych, a więc jest to rezultat ostateczny. Ponowna komparacja może być wykonywana po ustawieniu na wyjściach przerzutników jedynek lub zer za pomocą sygnału $S = 1$ ewentualnie $R = 1$.

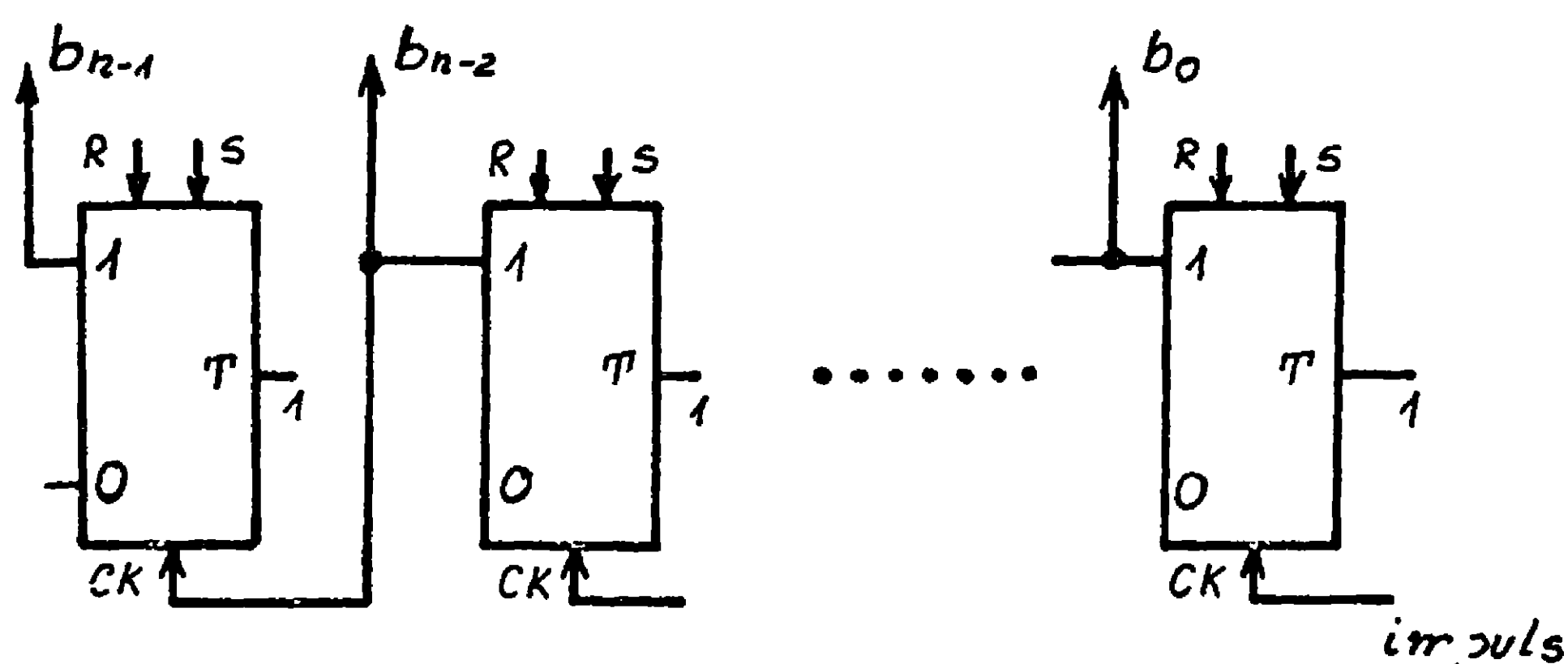
Przykład 5.8. Określić układ iteracyjny dodający do n-bitowej liczby binarnej liczbę 1.

Problem sprowadza się do określenia dla danej liczby $A(a_{n-1}, \dots, a_0)$ liczby

$$A + 1 = B(b_{n-1}, \dots, b_0),$$

przy czym znak $+$ oznacza tutaj operację arytmetyczną. Funkcję tę spełnia licznik szeregowy (rejestr) złożony z n przerzutników typu T posiadających wejścia R, S ustawiające stan początkowy każdego z nich. Połączenia między przerzutnikami pokazano na rysunku 5.57. Działanie układu jest analogiczne do działania omówionego w podpunkcie 5.5.1 podzielnika pokazanego na rysunku 5.32. Pojedynczy impuls podany na

wejście zegarowe przerzutnika zawierającego początkowo zapis cyfry a_0 , powoduje zwiększenie liczby wpisanej do tego rejestru o 1.

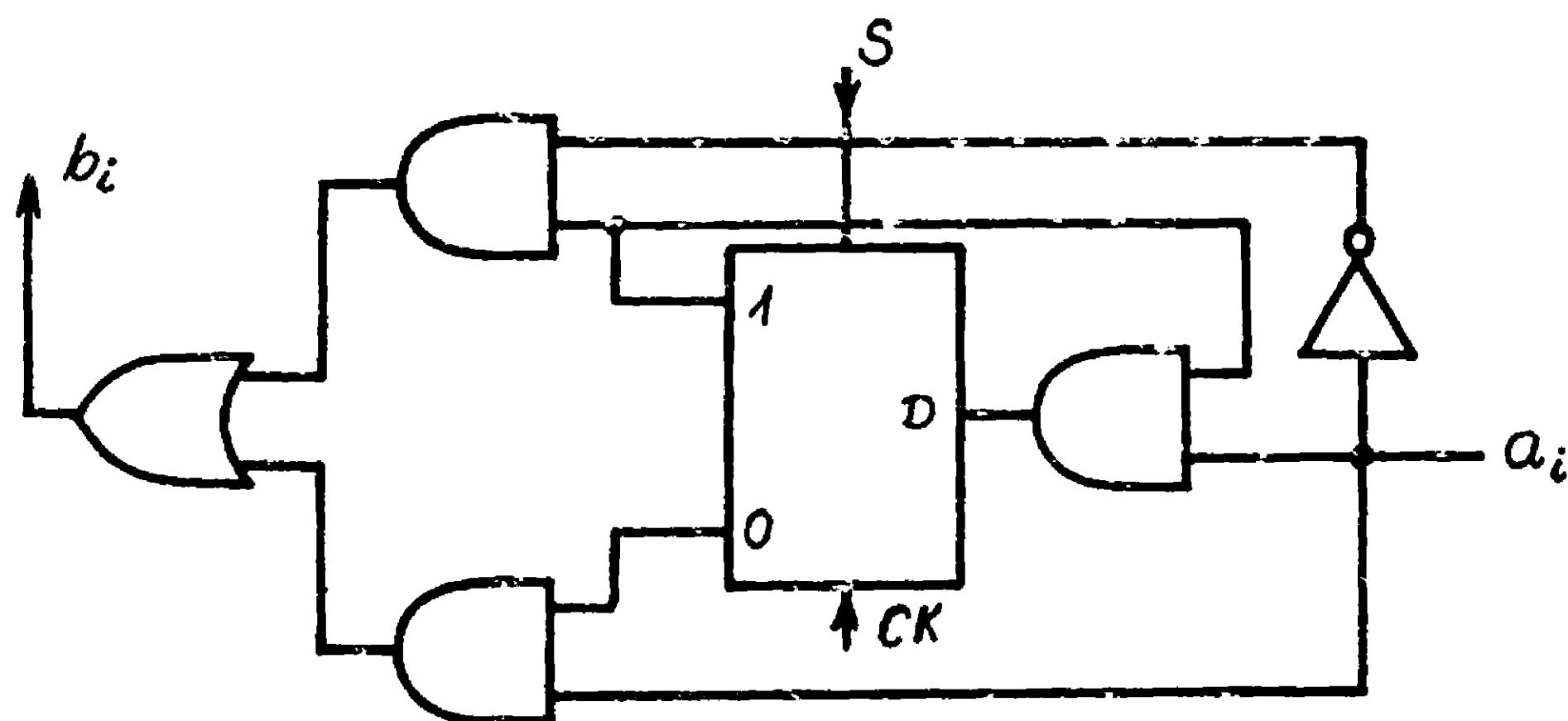


Rys. 5.57. Schemat układu licznika realizującego zwiększenie o 1 liczby wpisanej do poszczególnych pozycji

Aby omówione zadanie zrealizować w oparciu o pojedynczą komórkę iteracyjną, na jej wejście muszą być podawane kolejne cyfry liczby A , począwszy od pozycji o najniższej wadze, a więc od a_0 . Dodanie jedynki do liczby wyrażonej w postaci binarnej sprowadza się do:

1) wpisanie wartości $b_0 = 1$, gdy $a_0 = 0$; pozostałe cyfry nie ulegają zmianie, tzn. $b_i = a_i$ ($i = 1, \dots, n-1$),

2) wyzerowanie dla $a_0 = 1$ tych bitów liczby B , począwszy od b_0 , dla których występują kolejno jedynki w liczbie A , aż do pierwszej cyfry a_i , dla której wystąpi 0; cyfra b_i przyjmie wówczas wartość $b_i = 1$, natomiast cyfry o wagach wyższych będą takie same, tzn. $b_{i+1} = a_{i+1}$, $b_{i+2} = a_{i+2}$, ..., $b_{n-1} = a_{n-1}$.



Rys. 5.58. Schemat komórki iteracyjnej z przykładu 5.8

Komórkę iteracyjną realizującą powyższe zadanie, pokazano na rysunku 5.58. Składa się ona z przerzutnika typu D, na którego wyjściu jest ustawiany stan początkowy 1 sygnałem logicznym $S = 1$. Jeśli $a_0 = 0$ do wyjścia poprzez negator i bramkę iloczynu przekazywany jest sygnał $b_i = 1$. Jednocześnie po przejściu impulsu zegarowego wyzerowany zostaje przerzutnik uaktywniając dolny i blokując górny tor propagacji sygnału do wyjścia. Stan ten jest trwały z uwagi na sprzężenie wyjścia przerzutnika z jego wejściem poprzez bramkę iloczynu. Dla dalszych impulsów $b_i = a_i$, a, więc praca przebiega wg wariantu 1.

Jeśli $a_0 = 1$, to na wyjście b_i jest podawany sygnał logiczny 0 i to tak długo, dopóki nie pojawi się cyfra o wartości 0. Wyjście b_i przyjmuje wówczas wartość $b_i = 1$, trwale zeruje się przerzutnik i dalej przekazywane już są tylko dolnym torem sygnały opisujące cyfry liczby A. Praca przebiega więc wg wariantu 2.

W odróżnieniu od iteracyjnych układów przestrzennych omówionych w rozdziale 3, układy iteracji czasowej wymagają sygnałów rozłożonych w czasie, mogą więc występować tylko jako podzespoły bardziej złożonych systemów. Należy dodać, że w zasadzie nie ma metod formalnych podających algorytm syntezy komórki iteracji czasowej. Projektowanie wymaga głównie podejścia intuicyjnego, wspartego metodami prób i błędów.

6. ASYNCHRONICZNE UKŁADY SEKWENCYJNE

6.1. Uwagi ogólne

Układy asynchroniczne dominowały w okresie, gdy funktory logiczne realizowano w oparciu o technikę dyskretną, a więc wykorzystującą bramki złożone z elementów indywidualnych takich jak: rezystory, diody czy tranzystory. Było to związane z mniejszą złożonością asynchronicznych schematów logicznych w porównaniu ze schematami synchronicznymi operującymi rozbudowanymi strukturami przerzutników wyzwanych przednim lub tylnym zboczem impulsu zegarowego (patrz p. 5.3). W odróżnieniu od układów synchronicznych, dla których nie ma ograniczeń, tylko niektóre automaty dadzą się bezpośrednio zrealizować jako układy logiczne asynchroniczne. Automaty te są nazywane automatami asynchronicznymi. Aby je określić należy wprowadzić pojęcie stanu stabilnego.

Definicja 6.1. Stanem stabilnym automatu nazywamy taki stan t , dla którego jeśli $\delta(x,s) = t$, to musi być także $\delta(x,t) = t$. Innymi słowy stanem stabilnym automatu nazywamy taki stan, który na grafie ma pętlę własną dla danej litery wejściowej.

Definicja 6.2. Automatem asynchronicznym nazywamy automat, którego wszystkie stany są stanami stabilnymi.

Problem techniczny stosunkowo rzadko daje się bezpośrednio sformułować w kategoriach automatu asynchronicznego. Najczęściej zrealizowany układ jest tylko przybliżeniem takiego automatu. Znalezienie właściwego rozwiązania wymaga pewnego doświadczenia konstrukcyjnego, proces projektowania jest trudniejszy i obwarowany licznymi ograniczeniami. I to stało się główną przyczyną ogromnej ekspansji zastosowań synchronicznych układów sekwencyjnych, datowanej od momentu masowej podaży układów scalonych. Obecnie układy asynchroniczne znajdują zastosowanie w małych blokach sterujących automatyki lokalnej, natomiast w systemach cyfrowych są stosowane rzadko, głównie jako liczniki (patrz p. 5.5.1).

Brak zewnętrznych impulsów synchronizujących powoduje, że w wielu zastosowaniach układy asynchroniczne pracują nieco szybciej niż synchroniczne. Pozwala to przypuszczać, że w przyszłości będzie nadal

istnieć zapotrzebowanie na optymalną syntezę projektową tych układów. Dotyczy to przede wszystkim tzw. układów potencjałowych (statycznych), w których odpowiednim poziomom napięć lub prądów są przypisane wartości logiczne 1 i 0. Mniejsze znaczenie mają asynchroniczne układy impulsowe lub impulsowo-potencjałowe (dynamiczne), gdzie wartości 0 i 1 przypisuje się również impulsom uzyskiwanym na zboczach sygnałów sterujących za pomocą bramek różniczkujących te zbocza. Najprostszą realizację takich bramek uzyskuje się wykorzystując obok klasycznych funkcyjów układy diodowo-kondensatorowe.

6.2. Analiza pracy asynchronicznych układów potencjałowych (statycznych)

Ogólny schemat asynchronicznego układu potencjałowego przedstawiono na rysunku 6.1. Zasadnicza różnica pomiędzy tym schematem a schematem układu synchronicznego przedstawionym na rysunku 5.1 polega na zastąpieniu bloku przerzutników synchronicznych blokiem opóźniającym oznaczonym symbolem Δ . Dla ułatwienia analizy wygodnie jest przyjąć, że całe opóźnienie w pętli: układ kombinacyjny, Z , Δ , S jest skomasygowane w bloku Δ . Zawiera więc on opóźnienia własne i opóźnienia bramek logicznych układu kombinacyjnego. Tym samym przyjmujemy, że podany na schemacie blok kombinacyjny żadnych opóźnień nie wnosi. Założenie to wbrew pozorom nie wprowadza ograniczeń, a jedynie porządkuje rozważania, bowiem układ funkcjonuje w wyniku istnienia sumarycznych opóźnień w wymienionej pętli, niezależnie od tego gdzie one wystąpią. Podane na schemacie pokazanym na rysunku 6.1 oznaczenia wektorów X , Y są takie same jak dla układu synchronicznego (rys. 5.1), natomiast wektor Z reprezentuje wyjścia kombinacyjne podawane na blok opóźnień. Po przejściu bloku Δ , wektor Z przechodzi w wektor stanu S . Załóżmy na początku, że opóźnienia dla wszystkich składowych binarnych wektora Z są jednakowe oraz wprowadźmy oznaczenia: X_t , Y_t , S_t , Z_t - wartości wektorów w dowolnej chwili t , X_{t+1} , Y_{t+1} , S_{t+1} , Z_{t+1} - wartości wektorów w chwili $t + \Delta$.

Można wówczas napisać:

$$Z_t = S_{t+1} \quad (1)$$

oraz

$$\lambda': \langle X_t, S_{t+1} \rangle \rightarrow Z_{t+1}, \quad (2)$$

co definiuje jednocześnie operację λ' . Analiza wymienionych zależności wskazuje na niebezpieczeństwo wystąpienia niestabilnej pracy układu. Dopóki wartość wektora Z się zmienia po każdym cyklu opóźnienia Δ , w układzie występują ciągłe zmiany wektora stanu S . Stany te są w danych warunkach stanami niestabilnymi, a układ generuje określone sekwencje binarne. Organia ustają w momencie, gdy

$$Z_t = Z_{t+1},$$

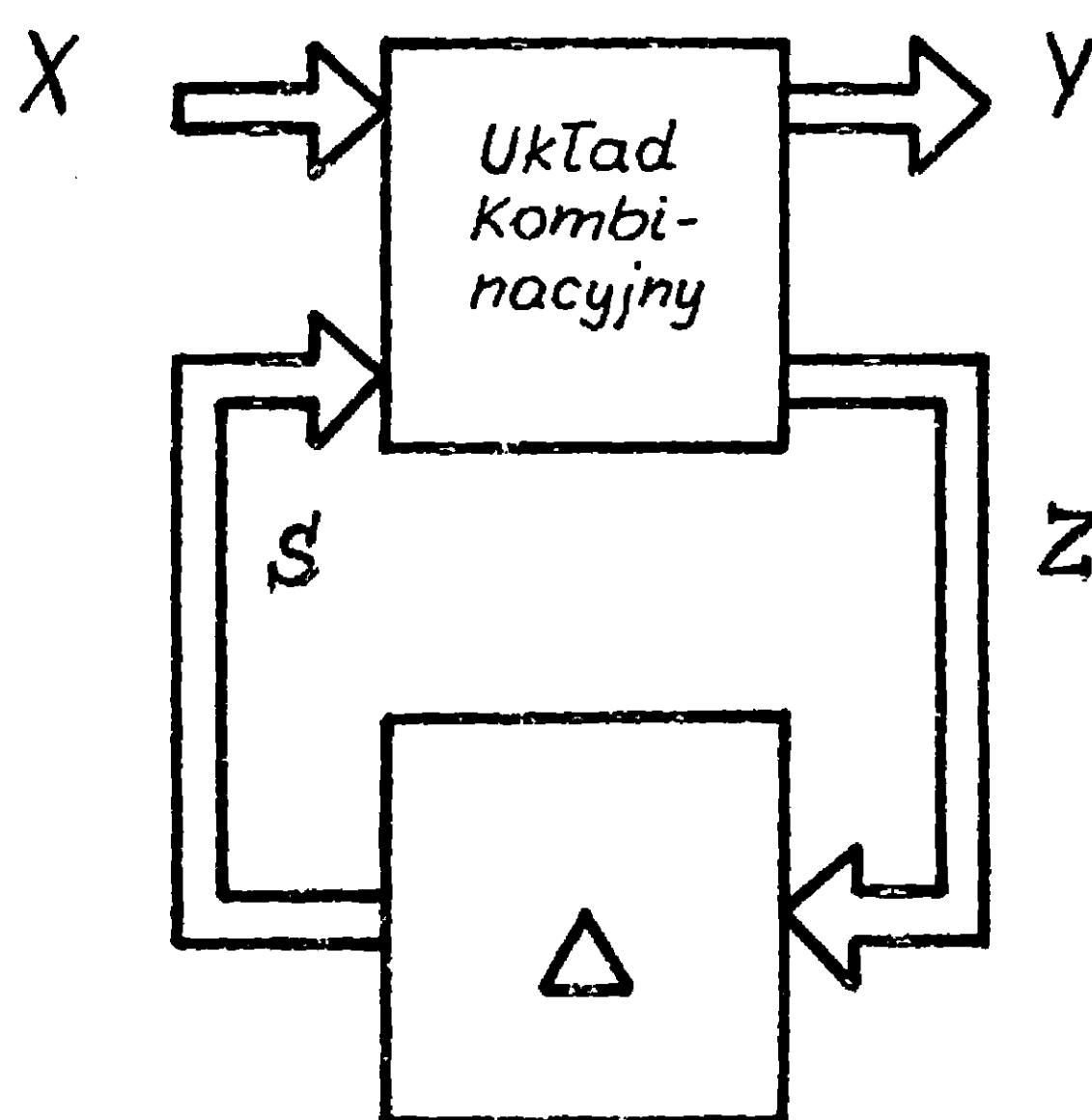
a więc również, gdy

$$S_{t+1} = Z_{t+1}.$$

Można więc powiedzieć, że dany stan jest stanem stabilnym wówczas, gdy

$$\langle X_t, S_{t+1} \rangle \rightarrow S_{t+1} = Z_{t+1},$$

co jest zgodne z definicją 6.1 stanu stabilnego automatu skończonego.



Rys. 6.1. Schemat ogólny asynchronicznego sekwencyjnego układu potencjałowego

Powyższe zjawiska nie występują w przypadku układów synchronicznych, bowiem blokując impulsy zegarowe można zatrzymać się w dowolnym stanie realizowanego grafu. Zadanie stojące przed projektantem układu asynchronicznego jest więc niewątpliwie trudniejsze tym bardziej, że dochodzi jeszcze jedno negatywne zjawisko. Dotychczas zakładano, że

opóźnienia wnoszone przez człon Δ są jednakowe dla wszystkich składowych binarnych wektora Z . Spełnienie tego wymagania jest bardzo trudne i w rzeczywistości sygnał binarny jest propagowany z różnymi opóźnieniami występującymi dla poszczególnych składowych. Wiąże się to z niebezpieczeństwem niewłaściwej pracy układu. Aby to naświetlić założmy, że wektor S w chwili $t-1$, a więc S_{t-1} , różni się od wektora Z_{t-1} , na dwóch pozycjach, np.:

$$S_{t-1} = \langle \dots, 1, \dots, 1, \dots \rangle,$$

$$Z_{t-1} = \langle \dots, 0, \dots, 0, \dots \rangle = S_t,$$

zakładamy, że wektor X nie ulega zmianie. Zgodnie z (1) i (2), przewidywana wartość wektora S_t będzie się różniła od wektora S_{t-1} również na tych dwóch pozycjach. Z uwagi jednak na to, że opóźnienia są różne dla poszczególnych składowych, układ kombinacyjny w chwili t odczyta wartość wektora stanu jako

$$S'_t = \langle \dots, 0, \dots, 1, \dots \rangle$$

bądź też jako

$$S''_t = \langle \dots, 1, \dots, 0, \dots \rangle.$$

Zgodnie z (2), doprowadzi to do wygenerowania wartości wektorów wyjściowych

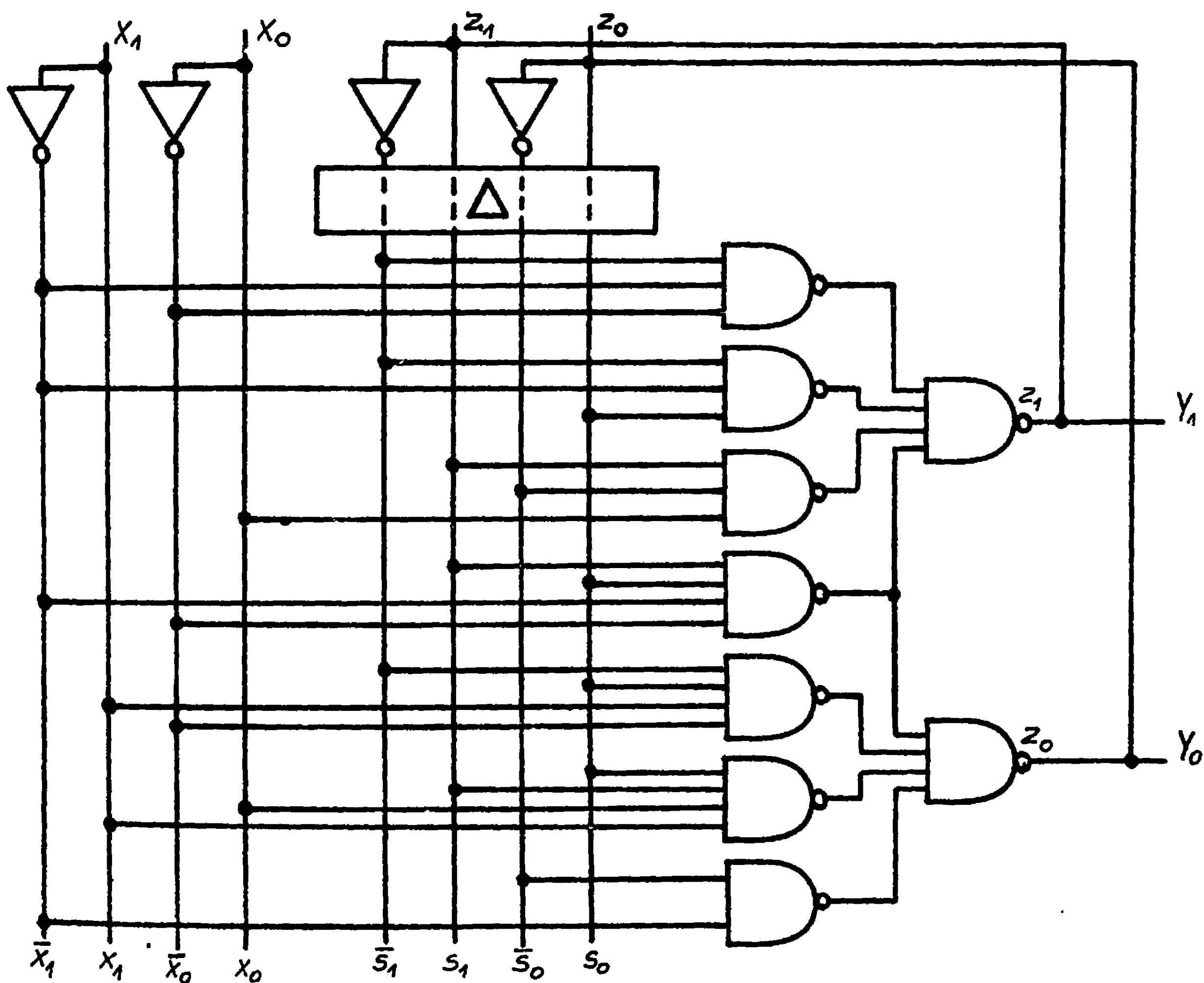
$$\langle X, S'_t \rangle \rightarrow Z'_t$$

lub

$$\langle X, S''_t \rangle \rightarrow Z''_t,$$

na ogół różnych od przewidzianych przez konstruktora. Może to doprowadzić - i najczęściej doprowadza - do nieprawidłowej pracy układu. Zjawisko powyższe nosi nazwę wyścigu. Wyścig jest niedopuszczalny i musi być przez projektanta wyeliminowany. Eliminację tę uzyskuje się, dopuszczając w kolejnych sekwencjach zmianę wektora stanu tylko na jednej pozycji. Jest to istotne ograniczenie, ze względów niezawodnościowych jednak konieczne często nawet wówczas, gdy układ po kilku cyklach nieprawidłowych, wróci do normalnej pracy. W ostatnim przypadku wyścig określa się mianem wyścigu niekrytycznego. Należy zwrócić uwagę na

fakt, że ograniczenie to obejmuje również dopuszczalne zmiany wartości wektora wejściowego X . Zmiana wartości wektora X na dwóch pozycjach binarnych, z uwagi na techniczną realizację sterowania, jest najczęściej dokonana z różnymi opóźnieniami dla poszczególnych składowych, co w konsekwencji może doprowadzić do wystąpienia wyścigu wygenerowanego przez wejście, tzw. wyścigu wejściowego. A więc zarówno dla wektora stanów S , jak i dla wektora wejściowego X , w poszczególnych sekwencjach czasowych dopuszcza się zmiany wartości tylko na jednej pozycji.



Rys. 6.2. Schemat układu z przykładu 6.1

Kolejne ograniczenie wiąże się z koniecznością eliminacji hazardu (p. 3.9) w bloku kombinacyjnym, bowiem wygenerowane dodatkowo "krótkie zera" lub "krótkie jedynki" mogą wywołać efekt analogiczny do zjawiska wyścigu. Blok kombinacyjny generalnie nie zawsze może więc być

budowany w oparciu o minimalne formuły boolowskie. Ograniczenie to oczywiście nie występuje w układach synchronicznych. Przytoczone przykłady zilustrują omawiane zjawiska.

Przykład 6.1. Przeprowadzić analizę pracy układu asynchronicznego, którego schemat logiczny przedstawiono na rysunku 6.2.

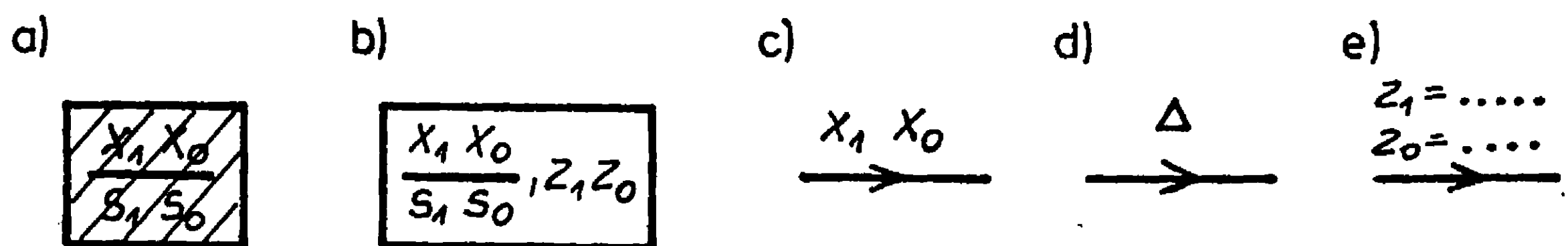
a)

	$x_1 x_0$	00	01	11	10	$y_1 y_0$
$s_1 s_0$	00	11	01	00	00	00
	01	10	10	00	01	01
	11	11	00	01	00	11
	10	01	11	10	00	10
		$z_1 z_0$				

b)

		00	01	11	10	
0	3	1	0	0	00	
1	2	2	0	1	01	
2	1	3	2	0	10	
3	3	0	1	0	11	

Rys. 6.3. Tablica przejść - wyjść układu pokazanego na rys. 6.2 (a) oraz ta sama tabela zapisana w konwencji automatowej (b) z kodowaniem binarno-dziesiętnym



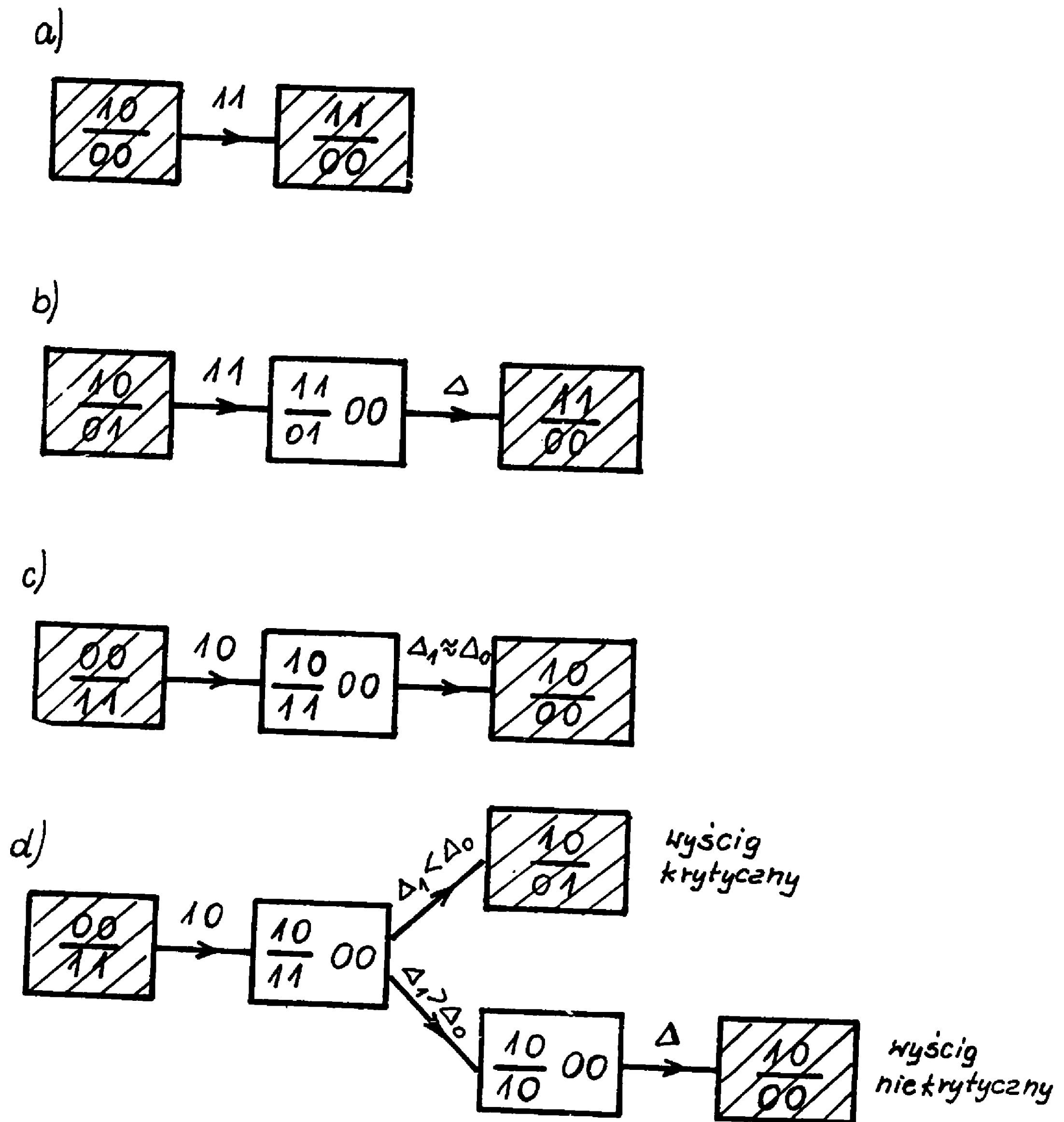
Rys. 6.4. Konwencje graficzne oznaczeń przejść między stanami; a) stan stabilny, b) stan niestabilny, następuje przejście do stanu $(z_1 z_0)_t = (s_1 s_0)_{t+1}$, c) wartość wektora wejściowego inspirowająca zmianę stanu, d) zmiana stanu wywołana samoczynnie z uwagi na jego niestabilność, e) zmiana stanu wymuszona

Tabelę przejść - wyjść układu przedstawiono na rysunku 6.3. Na rysunku tym przedstawiono również tabelę przejść - wyjść dostosowaną do konwencji automatowej, przy czym stanom wewnętrznym przypisano liczby dziesiętne będące odpowiednikami liczb binarnych utworzonych z ciągów zero-jedynkowych, reprezentujących te stany. Konwencje graficzne oznaczeń przejść między stanami przedstawiono na rysunku 6.4. Stan stabilny reprezentowany przez określoną wartość wektora $X (x_1, x_0)$ oraz $S (s_1, s_0)$ reprezentuje prostokąt pokazany na rysunku 6.4a, natomiast stan niestabilny prostokąt, w którym dodatkowo wpisano stan następny

$Z(z_1, z_0)$, (rys. 6.4b) [2]. Prostokąt stanu stabilnego dodatkowo zakreskowano. Przejścia między stanami mogą być wymuszone określoną zmianą wartości wektora X (rys. 6.4c) lub mogą dokonywać się samoczynnie dla stanów niestabilnych, co zaznaczono symbolem Δ (rys. 6.4d). Zmiany stanów mogą też być dokonywane "sztucznie" poprzez bezpośrednie pobudzenie bramek bloku kombinacyjnego. Zwykle zaznacza się to podając odpowiedni komentarz lub pisząc wymuszone wartości wektora Z nad kreśką łączącą stany (rys. 6.4e). Na rysunku 6.5 pokazano zachowanie się analizowanego układu dla różnych wymuszeń i różnych stanów wyjściowych. W przypadku a) układ będąc w stanie stabilnym (10/00) otrzymuje pobudzenie $X(11)$ i w wyniku przechodzi do drugiego stanu stabilnego (11/00). Przypadek b) ilustruje przejście do stanu stabilnego (11/00) poprzez stan niestabilny (11/01, 00). Rysunek c) przedstawia przejścia między stanami w przypadku zmiany wektora stanu na dwóch pozycjach jednocześnie ($11 \rightarrow 00$) przy założeniu jednakowych opóźnień dla obu składowych, natomiast rysunek d) przejścia dla przypadków $\Delta_1 \neq \Delta_0$. Jeżeli $\Delta_1 < \Delta_0$, to zamiast stanu 00 układ osiągnie stan następny 01 (zmiana na pierwszej pozycji jest szybsza), który w połączeniu z wartością 10 wektora wejściowego jest stanem stabilnym, innym od założonego. Wystąpiło tu zjawisko wyścigu krytycznego, układ osiągnął błędny stan docelowy. Gdy $\Delta_1 > \Delta_0$, stanem następnym będzie stan 10, który w połączeniu z wartością 10 wektora X jest stanem niestabilnym z przejściem do stanu 00 (10/10, 00). Przejście samoczynne Δ nie wprowadza dodatkowego wyścigu i układ osiąga stan stabilny (10/00), który jednocześnie jest stanem docelowym, jak w przypadku c). Ten wariant ilustruje zjawisko wyścigu niekrytycznego.

Jak już wspomniano, z analizy zależności (1) i (2) wynika, że możliwe są niestabilne cykle pracy układu asynchronicznego. Noszą one nazwę cykli granicznych. W omawianym przykładzie, dla wartości 01 wektora wejściowego w tabeli przejść - wyjść (rys. 6.3) nie ma stanów stabilnych. Na rysunku 6.6 zilustrowano możliwe zachowanie się układu po wejściu w cykl graniczny ze stanu stabilnego (11/00). Z uwagi na zjawisko wyścigów, informacja może cyrkulować trzema różnymi drogami:

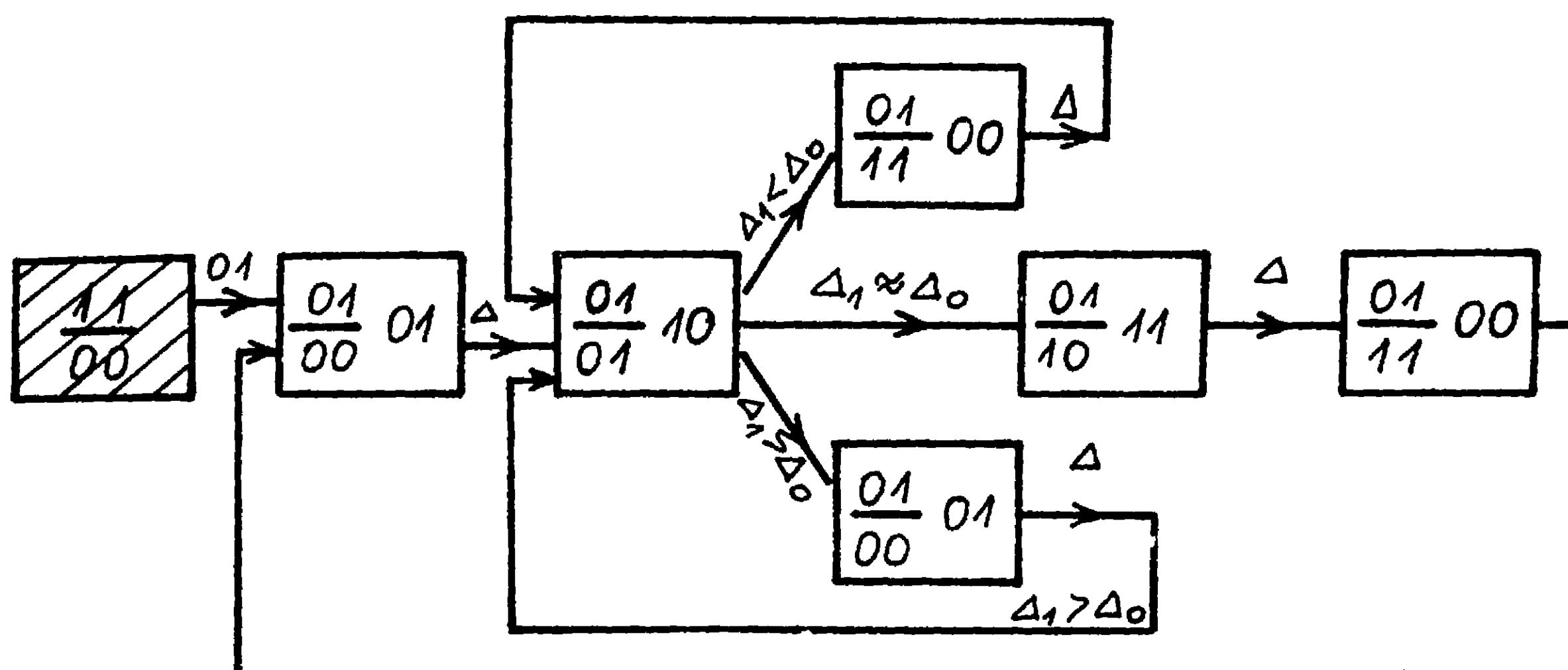
- a) dla $\Delta_1 < \Delta_0$ układ ze stanu niestabilnego (01/01, 10) przechodzi do stanu (01/11, 00); dalej (01/01, 10) i cykl się powtarza,
- b) dla $\Delta_1 \approx \Delta_0$ układ pracuje w pętli (01/00, 01), (01/01, 10), (01/10, 11), (01/11, 00), (01/00, 01) itd.,
- c) dla $\Delta_1 > \Delta_0$ cykl graniczny jest realizowany przez pętlę (01/00, 01), (01/01, 10), (01/00, 01).



Rys. 6.5. Ilustracja przejść między stanami dla układu pokazanego na rysunku 6.2 i opisanego tablicą pokazaną na rys. 6.3 (a, b, c, d)

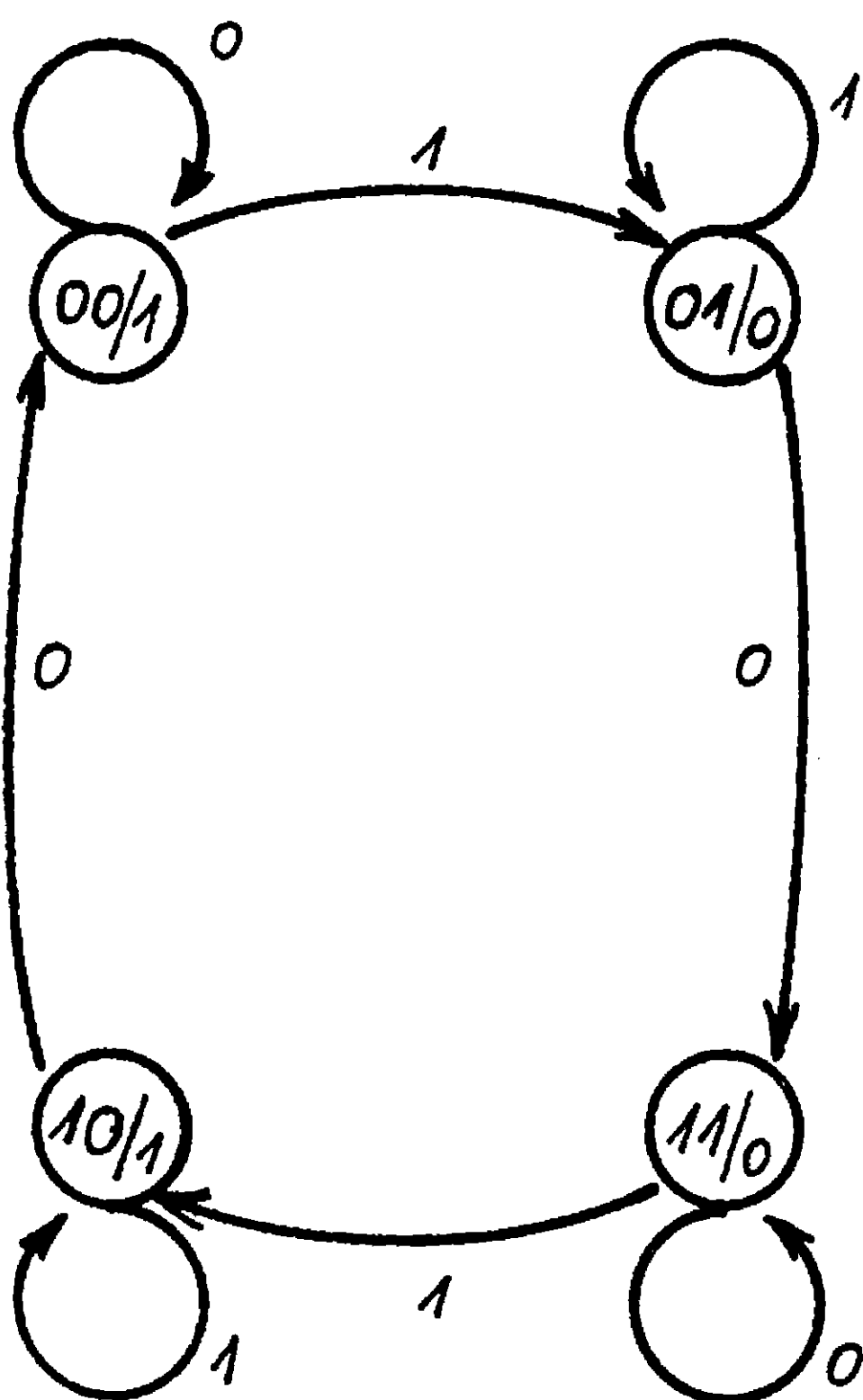
Z punktu widzenia standardowej pracy układu, cykle graniczne są niedopuszczalne. To samo dotyczy wyścigów krytycznych i na ogół niekrytycznych. Czasy propagacji bramek są bowiem silnie uzależnione od wpływu takich czynników jak np.: temperatura, starzenie, wymiana elementu i w konsekwencji wyścig niekrytyczny może stać się wyścigiem krytycznym. Jeśli jednak ze struktury układu wynika, że wyścig niekrytyczny nie stanie się krytycznym i nie będą generowane na wyjściu dodatkowe "krótkie zera" lub "krótkie jedynki" - to jest on dopuszczalny.

Wpływ zjawiska hazardu można zilustrować w oparciu o poniższy przykład.



Rys. 6.6. Ilustracja cykli granicznych w pracy układu pokazanego na rys. 6.2

a)



b)

$s_1 s_0$		y_0		y_0
		0	1	
00	00	01	1	
01	11	01	0	
11	11	10	0	
10	00	10	1	

$z_1 z_0$

Rys. 6.7. Graf (a) i tablica przejść - wyjść układu z przykładu 6.2 (b)

Przykład 6.2. Przeprowadzić analizę pracy układu, którego graf i tablicę przejść - wyjść pokazano na rysunku 6.7.

Każdy stan grafu jest stanem stabilnym, prezentowany układ jest więc automatem asynchronicznym. Rozpatrując jego pracę w kategoriach wejście - wyjście można zauważyć, że jeśli zmienna wejściowa ma wartość $x_0 = 0$, to wyjście y_0 pozostaje niezmienione, jeśli natomiast $x_0 = 1$ wartość wyjścia y_0 zmienia się na przeciwną. Układ realizuje więc tę samą funkcję co synchroniczny przerzutnik typu T. Przez analogię można go nazwać asynchronicznym przerzutnikiem typu T. Określone na podstawie tabeli boolowskie funkcje stanów następnych mają postać:

a) dla jedynek:

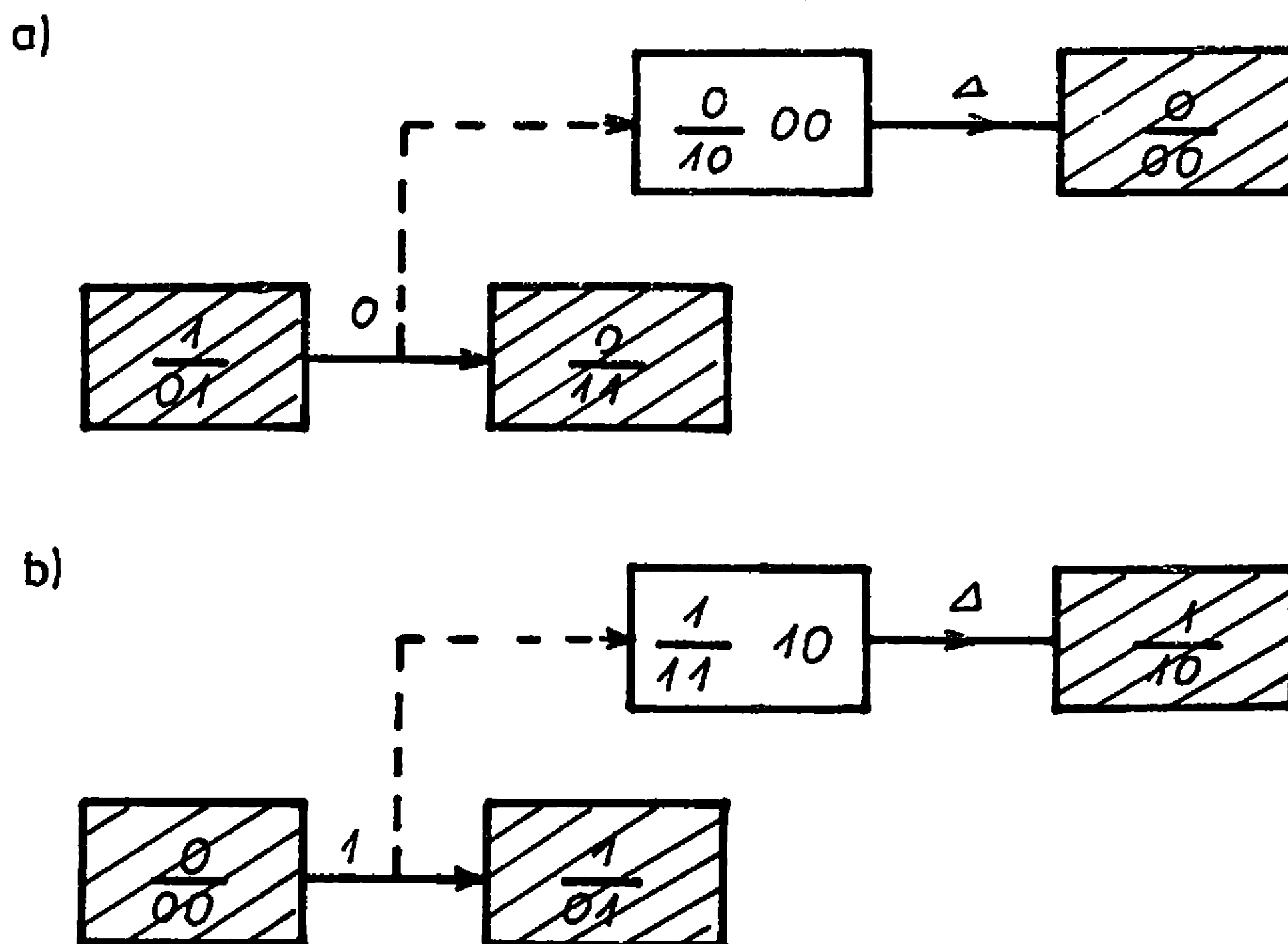
$$z_1 = s_0 \bar{x}_0 + s_1 x_0,$$

$$z_0 = s_0 \bar{x}_0 + \bar{s}_1 x_0,$$

b) dla zer:

$$z_1 = (s_0 + x_0)(s_1 + \bar{x}_0),$$

$$z_0 = (s_0 + x_0)(\bar{s}_1 + \bar{x}_0).$$



Rys. 6.8. Ilustracja wpływu hazardu na pracę układu opisanego grafem pokazanym na rys. 6.7a: a) hazard typu "krótkie zero" na pozycji z_0 , b) hazard typu "krótkka jedynka" na pozycji z_1

Obie postacie wykazują hazard względem zmiennej zewnętrznej x_0 , przy czym w przypadku a) wystąpi on dla zmiany typu $1 \rightarrow 0$. Zostanie wówczas wygenerowane "krótkie zero" dla wyjścia z_1 , jeśli $s_1 = 1$, $s_0 = 1$, a dla wyjścia z_0 jeśli $s_1 = 0$, $s_0 = 1$. Jeśli więc w układzie będącym np. w stanie 01 zmienna x_0 przyjmie wartość $x_0 = 0$, to w okresie trwania "krótkiego zera" jedynka na pozycji s_0 , przyjmie wartość $s_0 = 0$ i układ może "wskoczyć" w stan 10 zamiast planowanego 11 i dalej do stanu 00, bowiem dla wartości $x_0 = 0$ układ przechodzi ze stanu 10 do stanu 00 (rys. 6.8a). W przypadku b), sytuacja jest analogiczna z tym, że hazard wystąpi, gdy zmienna zewnętrzna x_0 zmienia wartość z 0 na 1. Przykładowo, gdy $s_1 = 0$, $s_0 = 0$ wystąpi hazard dla wyjścia z_1 , jeśli x_0 zmieni wartość z 0 na 1. Może zostać wówczas wygenerowana "krótka jedynka" na pozycji z_1 i układ zamiast przejść do stanu 01, znajdzie się w stanie 11 i dalej w stanie 10, bowiem $x_0 = 1$ (rys. 6.8b). Omówione przypadki świadczą dobitnie o tym, że hazard może zupełnie zmienić pracę układu asynchronicznego i generalnie jest niedopuszczalny. Na rysunku 6.9 przedstawiono schemat układu zrealizowany z bramek NOR, przy czym z równań stanów wyeliminowano hazard, zastępując formuły minimalne formułami postaci (patrz p. 3.9):

$$z_1 = (s_0 + x_0)(s_1 + \bar{x}_0)(s_1 + s_0),$$

$$z_0 = (s_0 + x_0)(\bar{s}_1 + \bar{x}_0)(\bar{s}_1 + s_0).$$

Bezpośrednio z tabeli przejść - wyjść wynika, że wyjście y_0 jest opisane zależnością

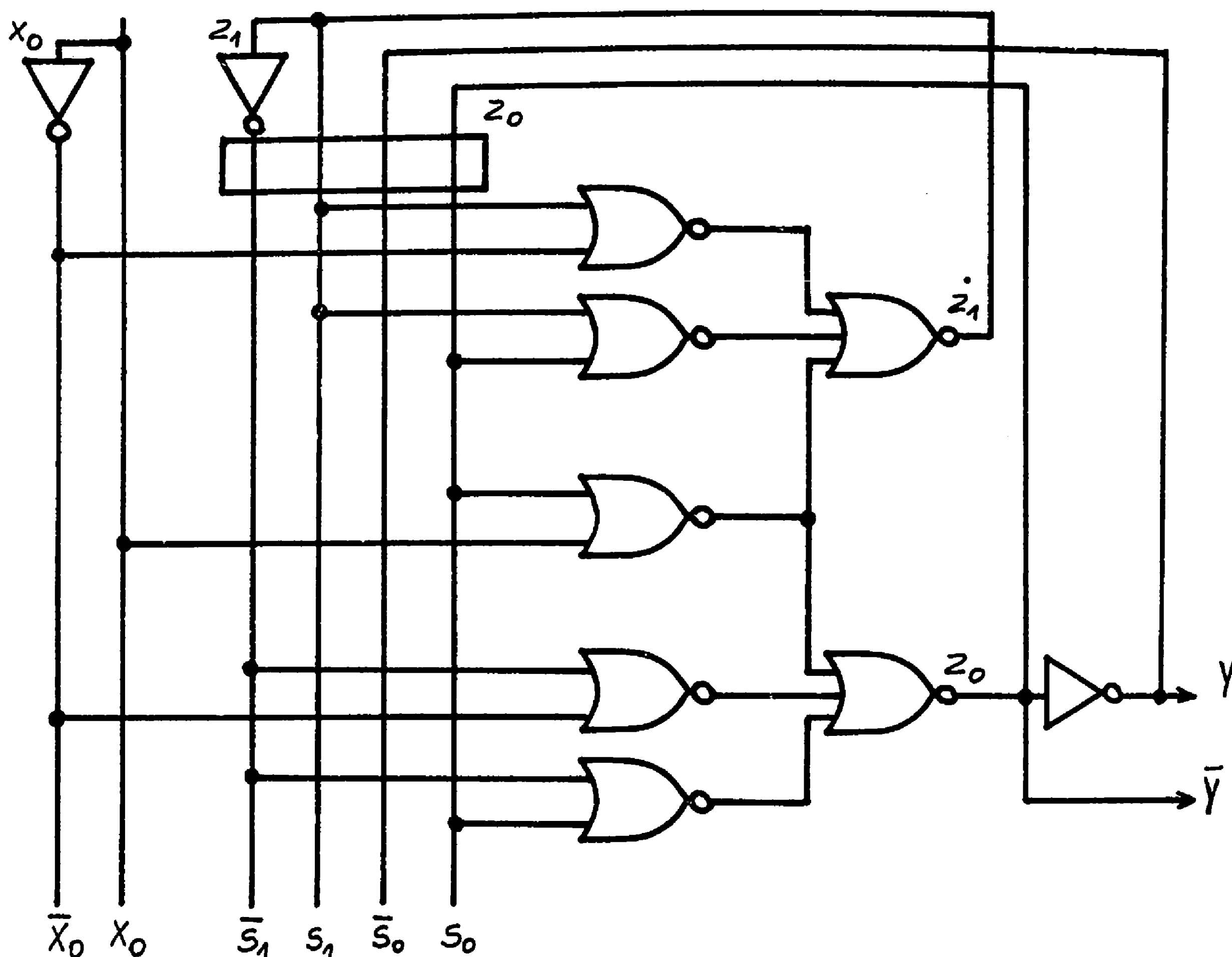
$$y_0 = \bar{s}_1 \bar{s}_0 + s_1 \bar{s}_0 = \bar{s}_0.$$

Zależności powyższe poprzez podwójne zanegowanie można dostosować do realizacji z bramek NOR:

$$z_1 = \overline{(s_0 + x_0) + (s_1 + \bar{x}_0) + (s_1 + s_0)},$$

$$z_0 = \overline{(s_0 + x_0) + (\bar{s}_1 + \bar{x}_0) + (\bar{s}_1 + s_0)}.$$

Jak już powiedziano w punkcie 3.9, dopełnienie i przekształcenie de Morgana nie zmieniają właściwości formuł boolowskich ze względu na hazard, a więc można je tutaj stosować bez ograniczeń.



Rys. 6.9. Układ z przykładu 6.2 zrealizowany z eliminacją hazardu

6.3. Synteza asynchronicznych układów potencjałowych

6.3.1. Uwagi ogólne

Z rozważań przeprowadzonych w punkcie 6.2 wynika, że przed projektantem asynchronicznego układu potencjałowego stoi cały szereg ostrych ograniczeń. Uwzględnienie ich wymaga zaangażowania zarówno aparatu formalnego jak i intuicji, bowiem najczęściej zagadnienia projektowego nie da się sformułować w kategoriach automatu asynchronicznego (patrz definicje 6.1 i 6.2) i konieczne jest bądź wprowadzanie pośrednich stanów niestabilnych lub stabilnych, bądź modyfikacja założeń. Reasumując, można stwierdzić, że w standardowym asynchronicznym, potencjałowym układzie sekwencyjnym:

a) nie może występować zjawisko wyścigu krytycznego; wyścig niekrytyczny jest dopuszczalny tylko wtedy, gdy ze struktury układu wyni-

ka, że nie stanie się on wyścigiem krytycznym oraz gdy na wyjściu nie pojawią się dodatkowe "krótkie zera" i "krótkie jedynki"; zmiana wartości wektora wejściowego odbywa się dopiero po ustaleniu przebiegów w układzie,

b) nie może występować zjawisko hazardu,

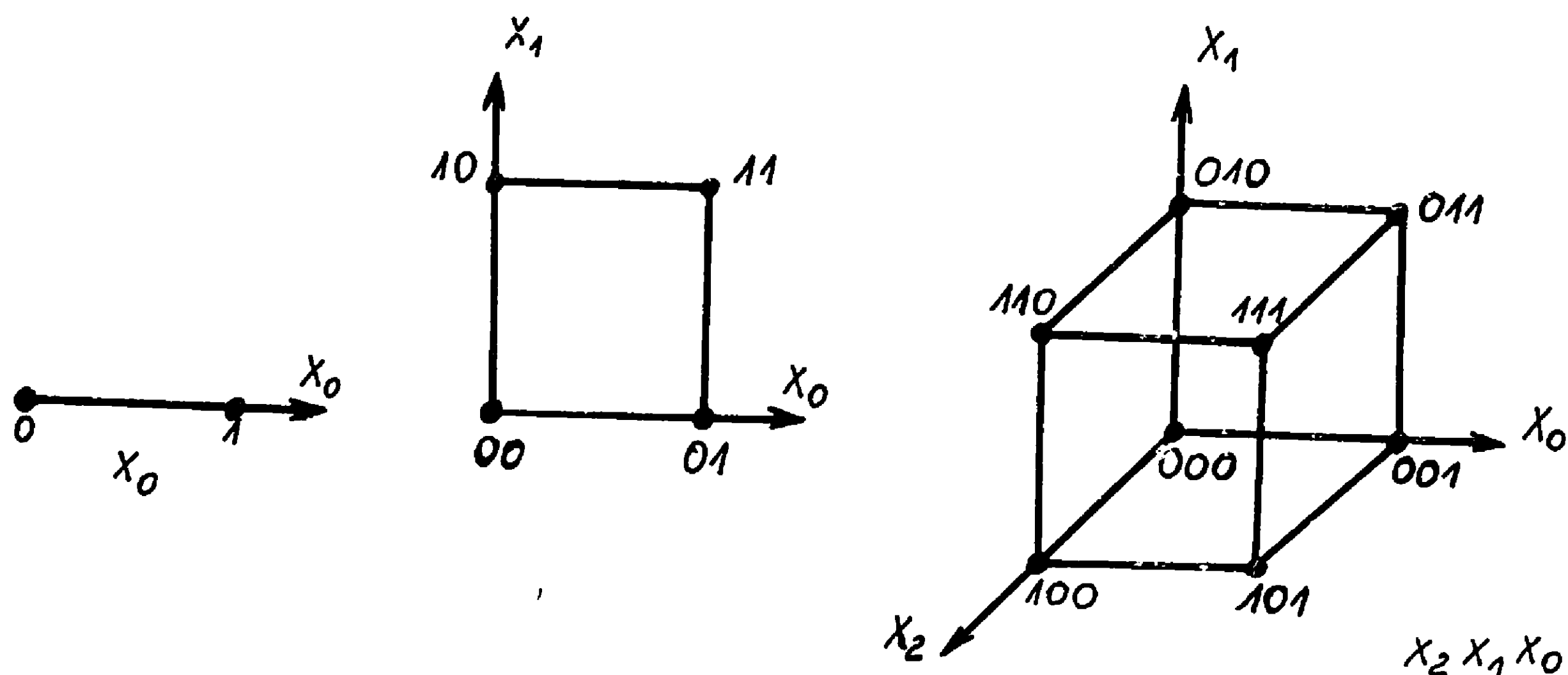
c) nie mogą występować cykle graniczne.

Warunek a) narzuca określone ograniczenia na przejścia między stanami oraz na kolejne sekwencje i szybkość zmian wektora wejściowego. W konsekwencji do procesu kodowania liter wejściowych i stanów wewnętrznych nie można podejść tak dowolnie, jak miało to miejsce w układach synchronicznych. Ponadto, jeśli wyjścia układu sterują dalszym blokiem asynchronicznym, wspomniane ograniczenia przenoszą się również na wektor wyjściowy. Spełnienie warunku b) uniemożliwia często korzystanie w procesie syntezy z minimalnych formuł boolowskich, co zilustrowano w przykładzie 6.2. Eliminacja cykli granicznych (warunek c) wymaga najczęściej dokonania zmian strukturalnych w założeniach projektowych.

Proces kodowania stanów wewnętrznych z eliminacją wyścigu można usprawnić, jeśli skorzystamy z właściwości ortogonalnych układów współrzędnych, w których przesuwanie punktu w kierunku równoległym do dowolnej osi powoduje zmianę tylko jednej jego współrzędnej. Wpisując w układ współrzędnych hipersześcian jednostkowy w ten sposób, by wersory tworzyły jego krawędzie, otrzymuje się gotowy wzorzec kodowania stanów układu asynchronicznego. Umieszczając poprzez kodowanie stany w narożach hipersześcianu w ten sposób, by przejścia między nimi odbywały się tylko po krawędziach, uzyskujemy układ wolny od wyścigu. Jeśli nie jest to możliwe, występują przejścia między stanami po przekątnych hipersześcianu i wyścigu wyeliminować się nie da. Konieczne jest wówczas zmodyfikowanie realizowanego grafu, np. poprzez wprowadzenie dodatkowych stanów niestabilnych. Na rysunku 6.10 przedstawiono hipersześciany jednostkowe w przestrzeni jedno, dwu i trójwymiarowej. Tworzą go kolejno figury geometryczne: odcinek, kwadrat i sześcian. Na rysunku tym zaznaczono również współrzędne wierzchołków tych figur, oznaczone wg konwencji wagowej: x_0 , x_1x_0 oraz $x_2x_1x_0$. Powyższa metoda kodowania jest bardzo łatwa w zastosowaniu, gdy liczba zmiennych boolowskich nie przekracza trzech. Dla większej liczby zmiennych nie można posługiwać się bezpośrednio obrazem geometrycznym hipersześcianu, można go jednak przedstawić bądź to w postaci sześcianów skła-

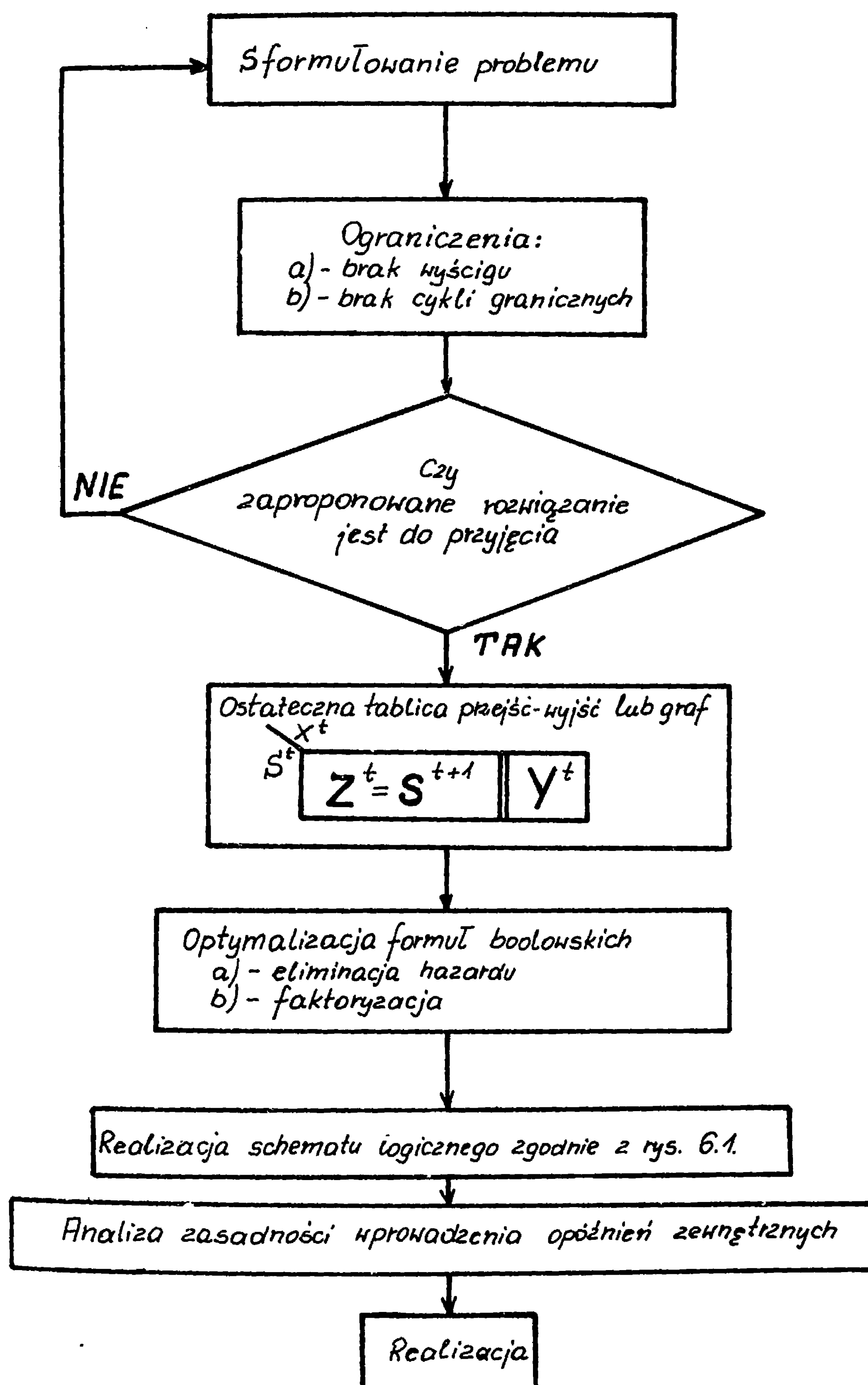


dowych bądź też wykorzystać fakt, że występowanie kolejnych zmian współrzędnych wierzchołków jest takie same jak dla sekwencji zmian kodu Gray'a (patrz p. 3.5).



Rys. 6.10. Jednostkowe hipersześciany jedno, dwu i trójwymiarowe, których część krawędzi jest utworzona z wektorów danego układu współrzędnych

Podobnie jak w przypadku układów synchronicznych, projektowanie asynchronicznego układu potencjałowego sprowadza się do określenia warunków sterowania wejść kombinacyjnych i ustalenia sygnałów wyjściowych, przy czym najwygodniejszą postacią zapisu formalnego warunków pracy jest tablica przejść - wyjść lub graf. Formalizacji problemu dokonujemy w ramach syntezy właściwej, wykonanej w zasadzie intuicyjnie, z uwzględnieniem wymienionych uprzednio ograniczeń. Dysponując tablicą przejść - wyjść lub grafem, określamy metodami opisanymi w rozdziale 3 formuły boolowskie dla poszczególnych wyjść tworzących wektor Y oraz wektor Z . Należy zwrócić uwagę na konieczność eliminacji hazardu z otrzymanych zależności. Dotyczy to zarówno formuł otrzymanych poprzez minimalizację niezależną poszczególnych wyjść, jak też i formuł minimalnych uzyskanych dla zespołów funkcji zredukowanych łącznie w procesie tabulacji. Etap ten można określić ogólnie jako optymalizację formuł boolowskich opisujących blok kombinacyjny. Następnie rysujemy schemat logiczny układu zgodnie z ogólnym schematem podanym na rysunku 6.1. Etap końcowy obejmuje analizę zasadności wprowadzania opóźnień zewnętrznych w pętlach sprzężeń zwrotnych. Znając czasy propagacji sygnałów przez zastosowane bramki, można ocenić łączne opóźnienia układu kombinacyjnego dla poszczególnych wyjść i określić czy

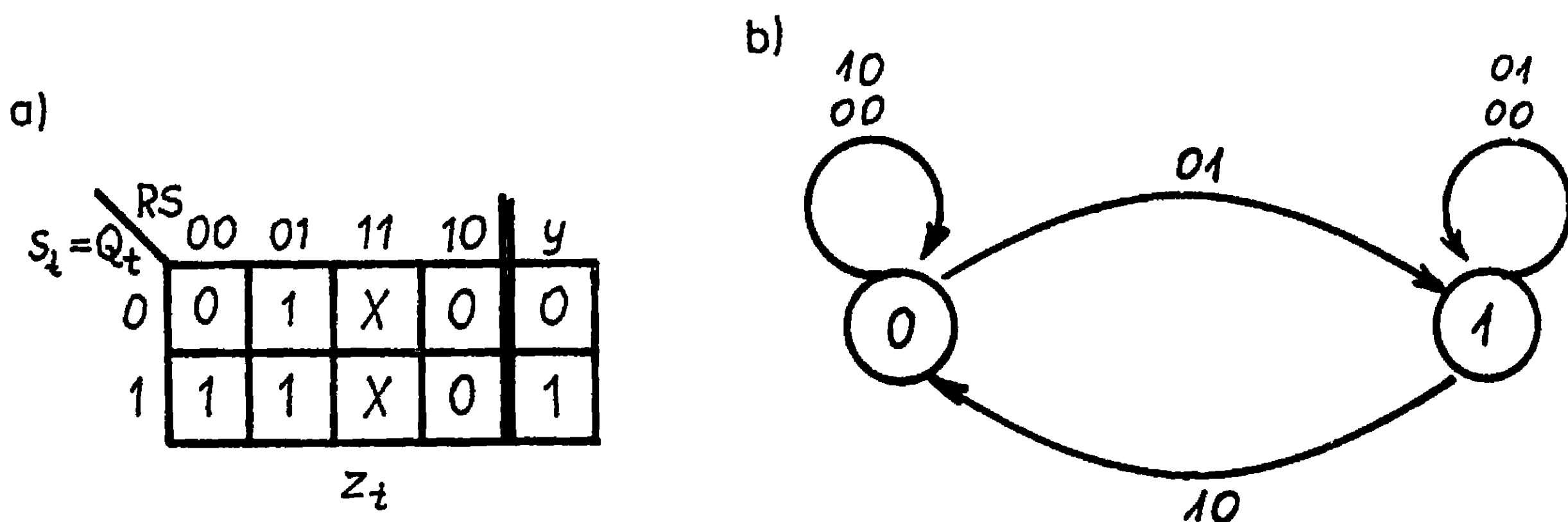


Rys. 6.11. Ogólna procedura projektowania logicznego asynchronicznego układu potencjałowego

zdążą się ustalić poziomy sygnałów w układzie. Przyjmuje się, że różnica czasów opóźnień równa czasowi propagacji sygnału przez jeden poziom bramek, jest wystarczająca do tego, by układ pracował poprawnie. Jeśli warunek ten nie jest spełniony, należy wprowadzić dodatkowe opóźnienia, np. poprzez włączenie bramek buforowych (rys. 2.10). Na rysunku 6.11 przedstawiono ogólną procedurę projektową asynchronicznego układu potencjałowego, obejmującą wyżej omówione etapy. Zagadnienia szczegółowe zostaną zilustrowane na przykładach. Należy dodać, że omówiona procedura nie dotyczy zastosowań specjalnych układów asynchronicznych.

6.3.2. Przykłady syntezy asynchronicznych układów potencjałowych

Przykład 6.3. Zaprojektować asynchroniczny przerzutnik potencjałowy, którego tabela przejść - wyjść odpowiada tabeli przerzutnika synchronicznego typu RS (rys. 5.2 i rys. 5.6).



Rys. 6.12. Tabela przejść - wyjść (a) oraz graf asynchronicznego przerzutnika typu RS (b)

Tabelę przejść - wyjść oraz graf przerzutnika z oznaczeniami dostosowanymi do schematu ogólnego układu asynchronicznego (rys. 6.1) pokazano na rysunku 6.12. Litera wyjściowa została utożsamiona ze stanem. Dla uniknięcia niejednoznaczności, oznaczenie wektora stanów S zastąpiono symbolem Q . Analiza grafu wskazuje na to, że przejścia między stanami mogą być realizowane bez wprowadzania wyścigu z wejścia, tzn. poprzez zmianę wartości tego wektora tylko na jednej pozycji. Przykładowo, jeśli układ znajduje się w stanie stabilnym (10/0), to wprowadzie do stanu 1 nie wolno przejść bezpośrednio poprzez zmianę wektora wejściowego z 10 na 01, ale można tego dokonać poprzez stan stabilny

(00/0), a więc zmieniając wektor wejściowy wg sekwencji: $10 \rightarrow 00 \rightarrow 01$. Informacje podane na rysunku pozwalają na określenie następujących formuł minimalnych:

$$z_t = Q_{t+1} = S + Q_t \bar{R}, \quad (1)$$

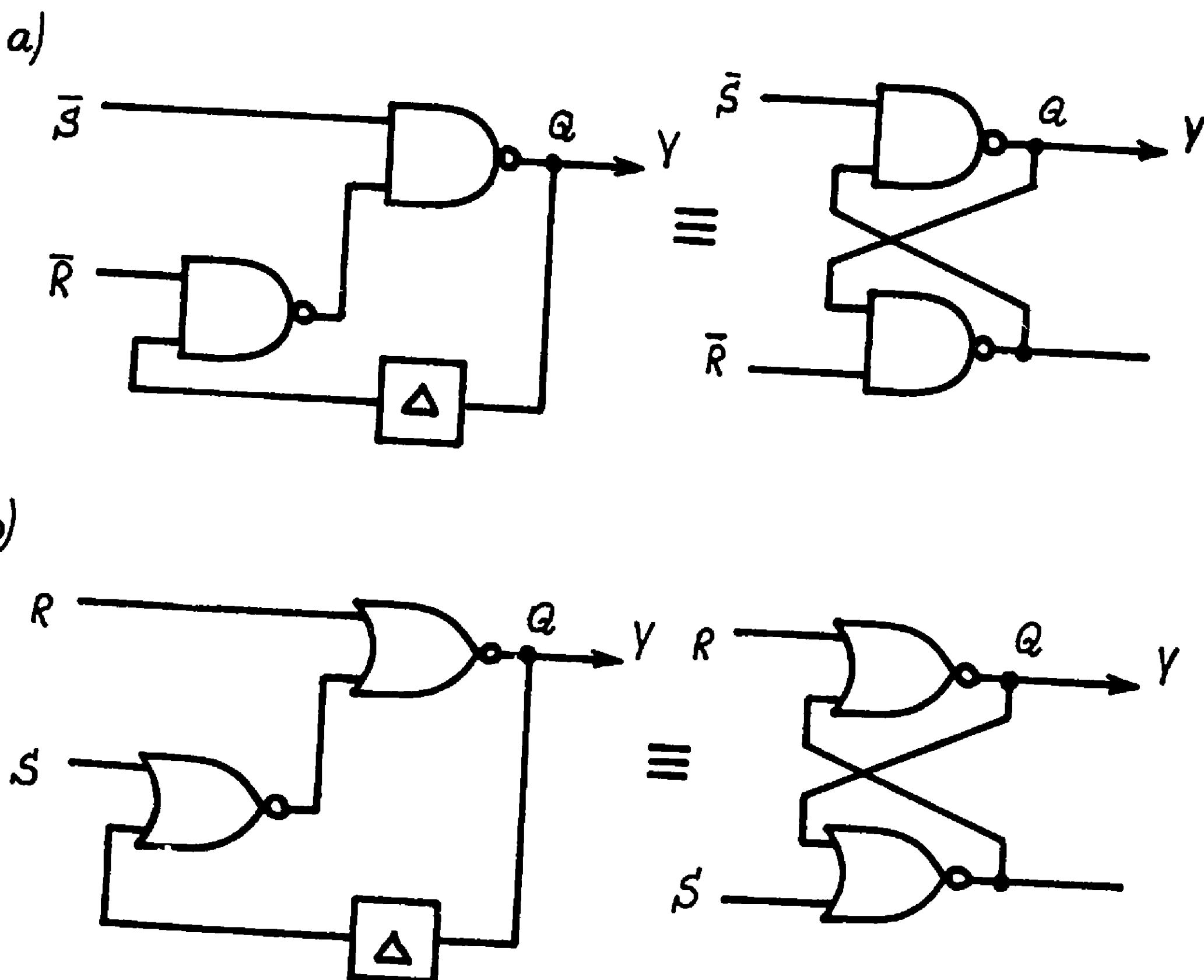
lub

$$z_t = Q_{t+1} = \bar{R}(Q_t + S). \quad (2)$$

Formuły nie zawierają hazardu. W obu przypadkach $RS = 0$. Formuły (1) i (2) dostosowane do realizacji NAND i NOR przyjmą postać:

$$z_t = Q_{t+1} = \overline{\bar{S} \overline{Q_t} \bar{R}}, \quad (1a)$$

$$z_t = Q_{t+1} = \overline{R + (\overline{Q_t} + S)}. \quad (2a)$$

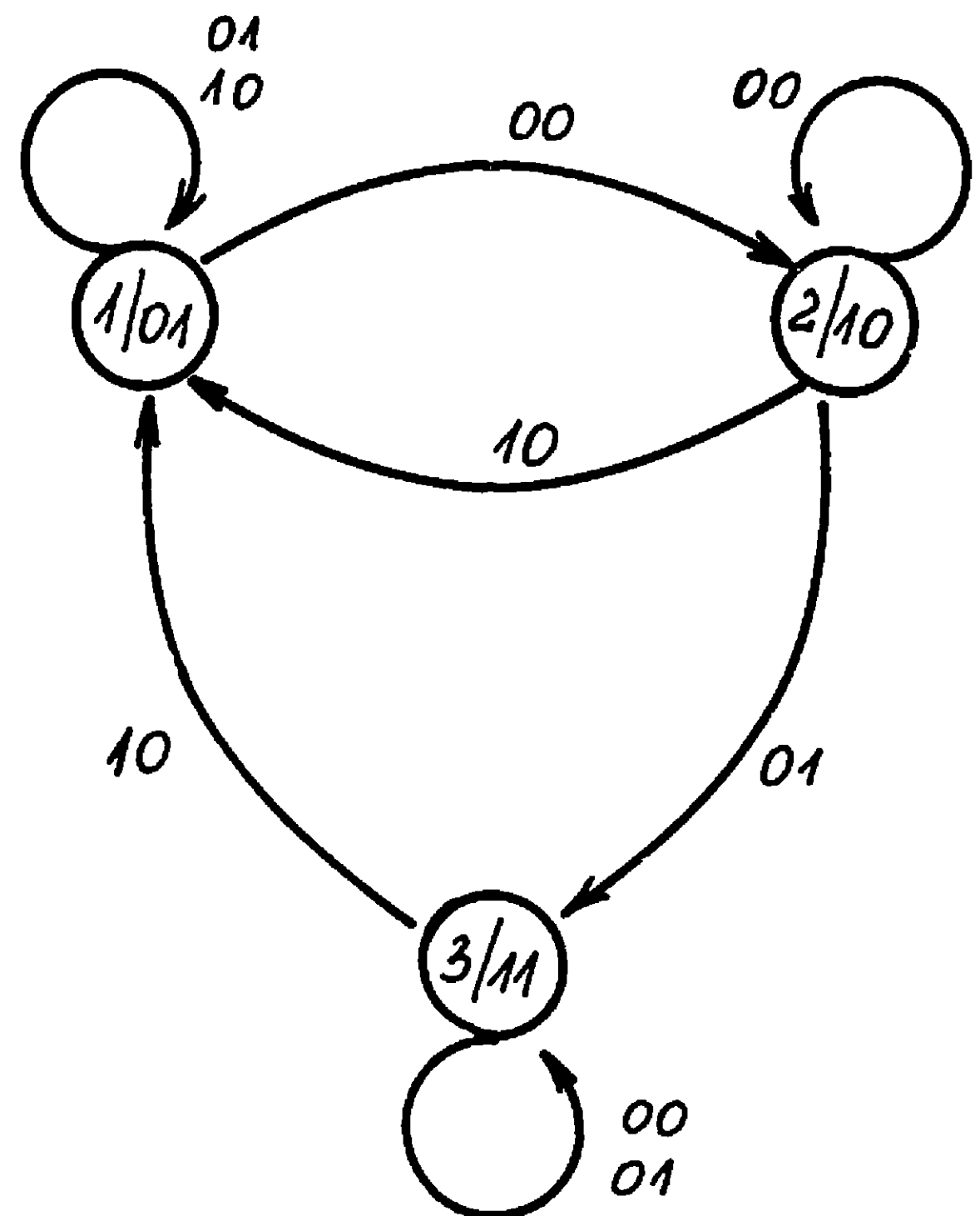


Rys. 6.13. Schematy asynchronicznego przerzutnika typu RS (a,b)

a)

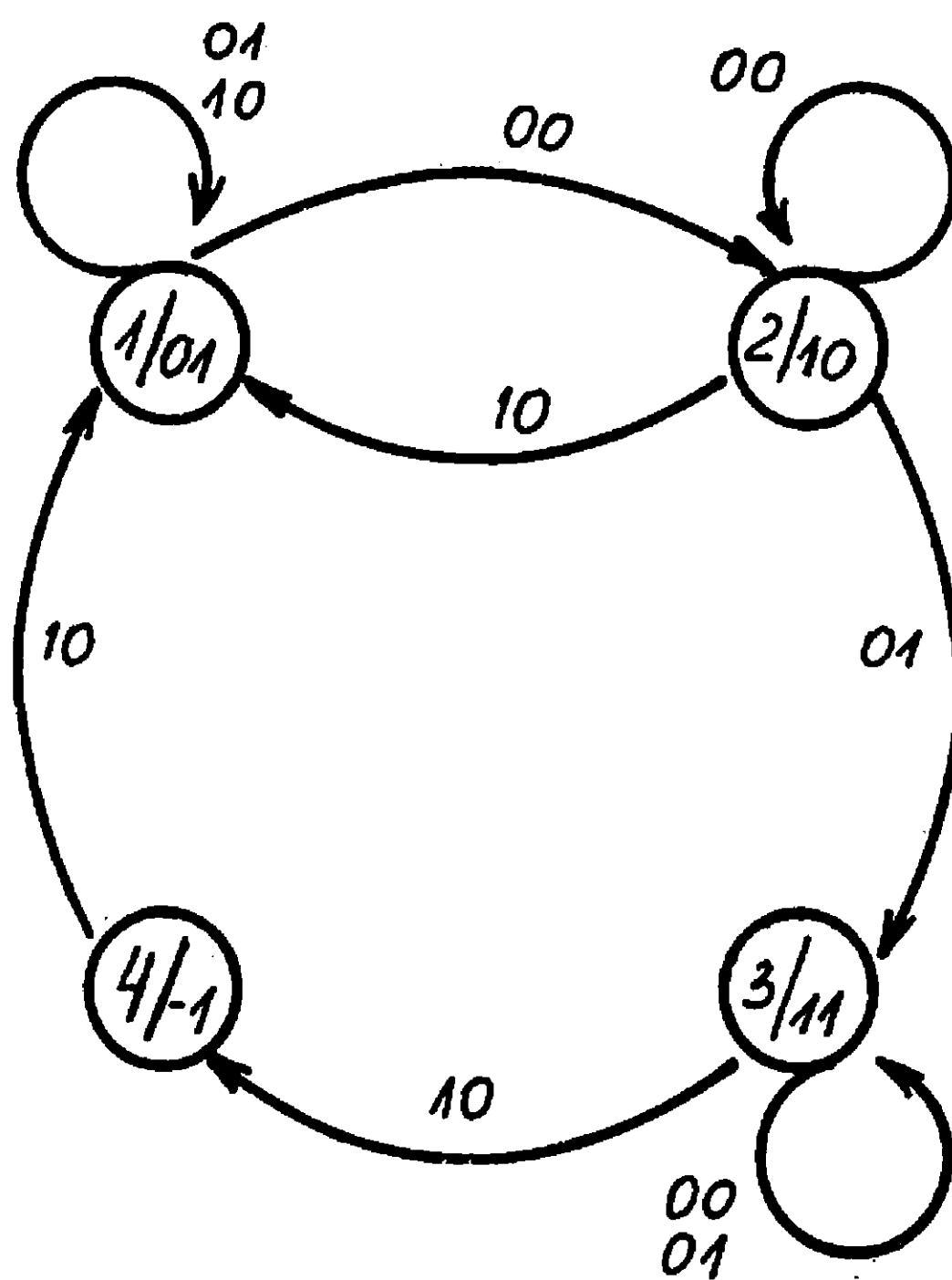
	00	01	10	11	
1	2	1	1	—	01
2	2	3	1	—	10
3	3	3	1	—	11

b)



Rys. 6.14. Tablica przejść - wyjść (a) i graf automatu z przykładu 6.4 (b)

a)



b)

		$x_1 x_0$				$y_1 y_0$
$s_1 s_0$		00	01	11	10	
(1) 00		01	00	—	00	01
(2) 01		01	11	—	00	10
(3) 11		11	11	—	00	11
(4) 10		—	—	—	00	-1
		$z_1 z_0$				

Rys. 6.15. Zmodyfikowana wersja realizowanego grafu i tablicy przejść - wyjść układu z przykładu 6.4

Na rysunku 6.13 pokazano schematy logiczne otrzymane na podstawie formuł (1a) i (2a). Analiza opóźnień wykazuje, że sygnał Q jest podawany zwrotnie na bramkę drugiego poziomu, dodatkowe elementy opóźniające są zbędne. Schematy przekształcają się wówczas do postaci elementarnych przerzutników dwubramkowych, opisanych w punkcie 5.2 (rys. 5.3 i rys. 5.4).

Przykład 6.4. Zaprojektować asynchroniczny układ potencjałowy realizujący automat opisany tablicą przejść - wyjść i grafem pokazanym na rysunku 6.14.

Analiza grafu doprowadza do wniosku, że nie ma możliwości zakodowania stanów w sposób taki, by wyeliminować zjawisko wyścigu. Stanom nie da się przypisać współrzędnych naroży kwadratu tak, aby przejścia odbywały się tylko wzdłuż krawędzi. Problem można rozwiązać modernizując graf poprzez wprowadzenie czwartego stanu niestabilnego w ten sposób, by litera wyjściowa związana z nim była taka jak dla któregoś z sąsiedniego stanu stabilnego. Wystąpią wówczas pewne przesunięcia w czasie ustalania się wartości binarnych na wyjściach, ale układ pozostanie stabilny. Na rysunku 6.15a przedstawiono tak zmodyfikowany graf. Czwarty stan umieszczono między stanem pierwszym i trzecim. Litery wyjściowe związane z nimi mają na pozycjach o niższej wadze wartość logiczną 1. a więc czwarty stan musi też mieć na tej pozycji tę wartość. Inaczej nie byłby zachowany warunek ciągłości sygnału wyjściowego - w czwartym stanie zostałoby wygenerowane na tym wyjściu "krótkie zero". Pozycja o wyższej wadze może mieć wartość 0 lub 1. W pierwszym przypadku zostanie "przyspieszony" rezultat wyjściowy związany ze stanem pierwszym, w drugim - "przeciągnięty" wynik wyjściowy związany ze stanem trzecim. Można więc na tej pozycji postawić kreskę, co ułatwi poszukiwanie optymalnej formuły boolowskiej. Na rysunku 6.15b przedstawiono tablicę przejść - wyjść otrzymaną ze zmodyfikowanego grafu przez przypisanie stanom wewnętrznym wartości binarnych w następujący sposób:

- 1 $\rightarrow$ 00,
- 2 $\rightarrow$ 01,
- 3 $\rightarrow$ 11,
- 4 $\rightarrow$ 10.

łatwo sprawdzić wpisując ten rezultat w naroża kwadratu, że wszystkie przejścia odbywają się wzdłuż jego krawędzi, a więc bez ryzyka wystą-

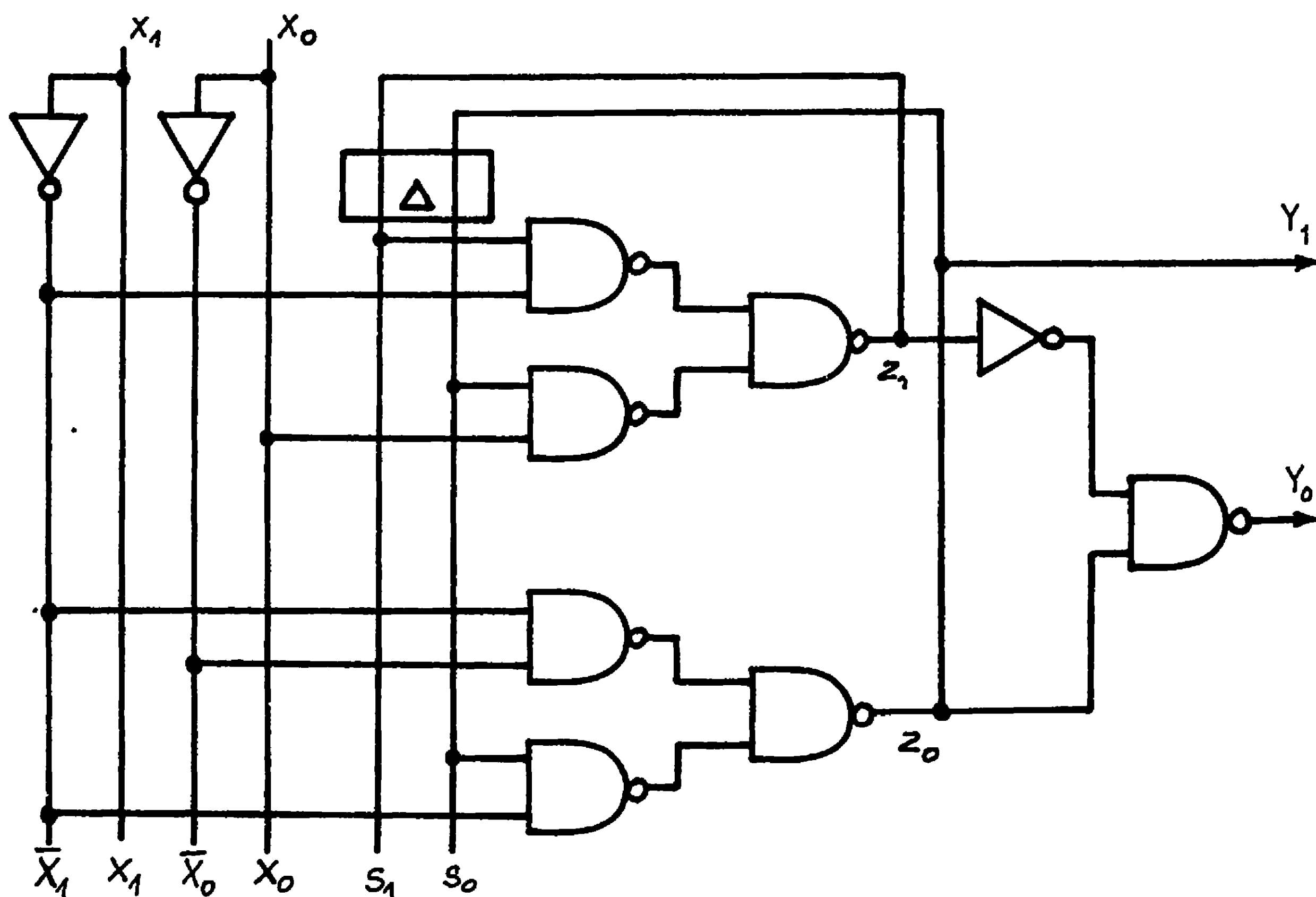
pienia wyścigu. Tabela ta zapoczątkowuje proces syntezy formalnej układu. Optymalne formuły boolowskie otrzymane na podstawie tabeli mają postać:

$$z_1^t = s_1^{t+1} = s_1^t \bar{x}_1 + s_0^t x_0 = \overline{s_1^t \bar{x}_1} \overline{s_0^t x_0},$$

$$z_0^t = s_0^{t+1} = \bar{x}_1 \bar{x}_0 + s_0^t \bar{x}_1 = \overline{\bar{x}_1 \bar{x}_0} \overline{s_0^t \bar{x}_1},$$

$$y_1 = s_0,$$

$$y_0 = s_1 + \bar{s}_0 = \overline{\bar{s}_1 s_0}.$$

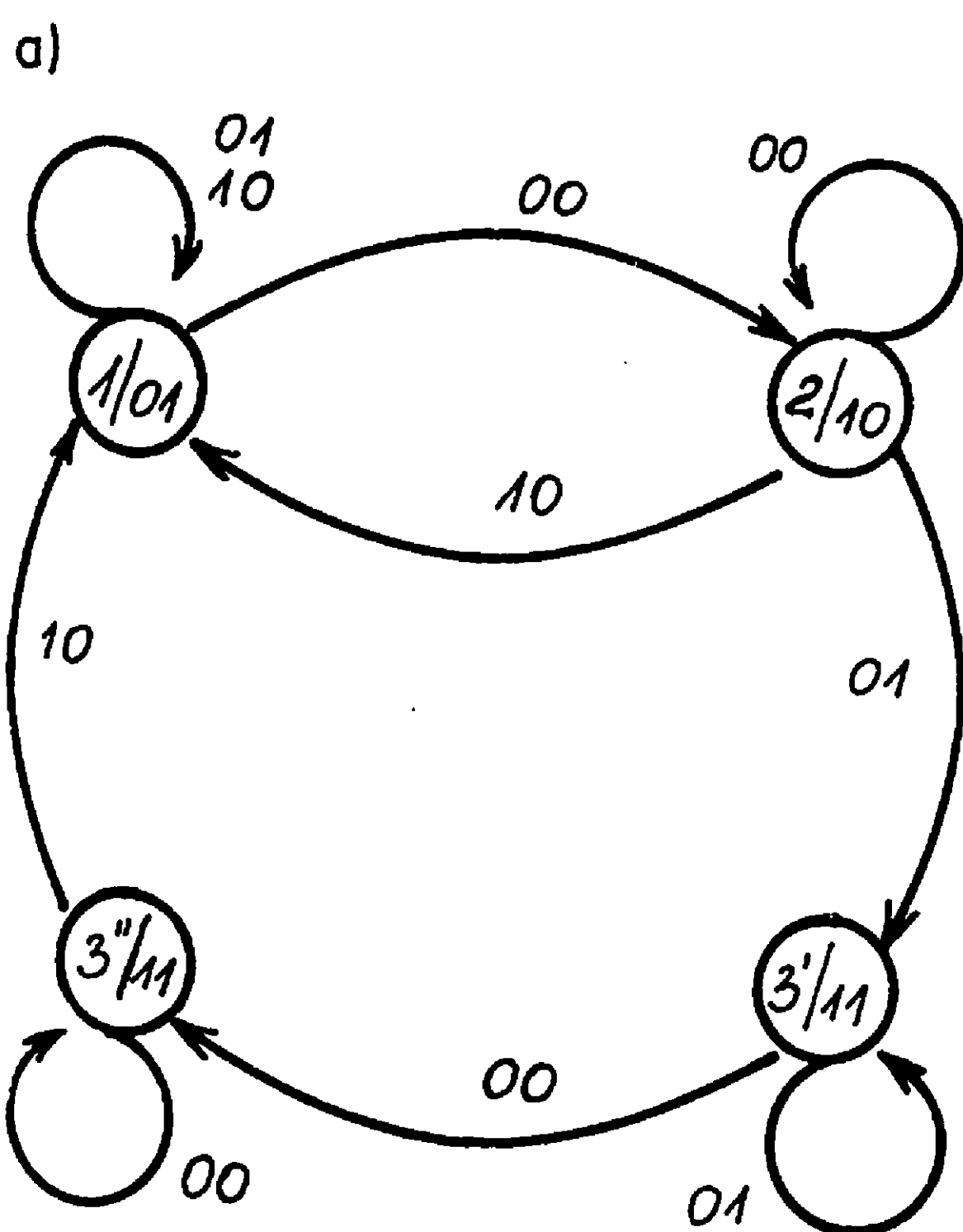


Rys. 6.16. Schemat układu opisanego tablicą przejść - wyjść pokazaną na rys. 6.15a (przykład 6.4)

Nie zawierają one hazardu. Na rysunku 6.16 przedstawiono schemat logiczny układu zrealizowany z bramek NAND. Człon opóźniający został narysowany jedynie symbolicznie, gdyż podobnie jak poprzednio wystarczająco opóźnienia realizowane na bramkach. Podobnie jak w poprzednim przy-

kładzie, przejścia między stanami mogą odbywać się bez wprowadzania wyścigu wejściowego. Należy jednak pamiętać, że do stanu 3 przechodzi się ze stanu 2 po pobudzeniu wektorem wejściowym 01. Następne przejście odbywa się po zmianie wektora wejściowego na 10 i musi być poprzedzone przejściem 00. Wyłania się przy tym inna możliwość modyfikacji grafu pokazanego na rysunku 6.14b. Zamiast wprowadzać stan niestabilny, można "rozbić" stan stabilny 3 na dwa stany stabilne: 3' z pętlą własną dla wartości wektora wejściowego 01 oraz 3'' z pętlą własną dla wartości 00. Litery wyjściowe związane ze stanami 3' i 3'' muszą być takie same jak dla stanu 3. Inaczej układ asynchroniczny realizowałby inną funkcję wyjściową, niezgodną z grafem pierwotnym. Na rysunku 6.17 przedstawiono zmodyfikowany wariant grafu pierwotnego oraz tablicę przejść - wyjść układu otrzymaną zgodnie z kodem:

1 $\rightarrow$ 00,
 2 $\rightarrow$ 01,
 3' $\rightarrow$ 11,
 3'' $\rightarrow$ 10.



b)

$s_1 s_0$		$x_1 x_0$				$y_1 y_0$
		00	01	11	10	
(1) 00		01	00	--	00	01
(2) 01		01	11	--	00	10
(3') 11		10	11	--	--	11
(3'') 10		10	--	--	00	11

$z_1 z_0$

Rys. 6.17. Wariant zmodyfikowanej wersji grafu (a) i tablicy przejść - wyjść układu z rys. 6.4 (b)

Zależności opisujące układ przyjmą postać:
 - dla stanów:

$$z_1^t = s_1^t \bar{x}_1 + s_0^t x_0,$$

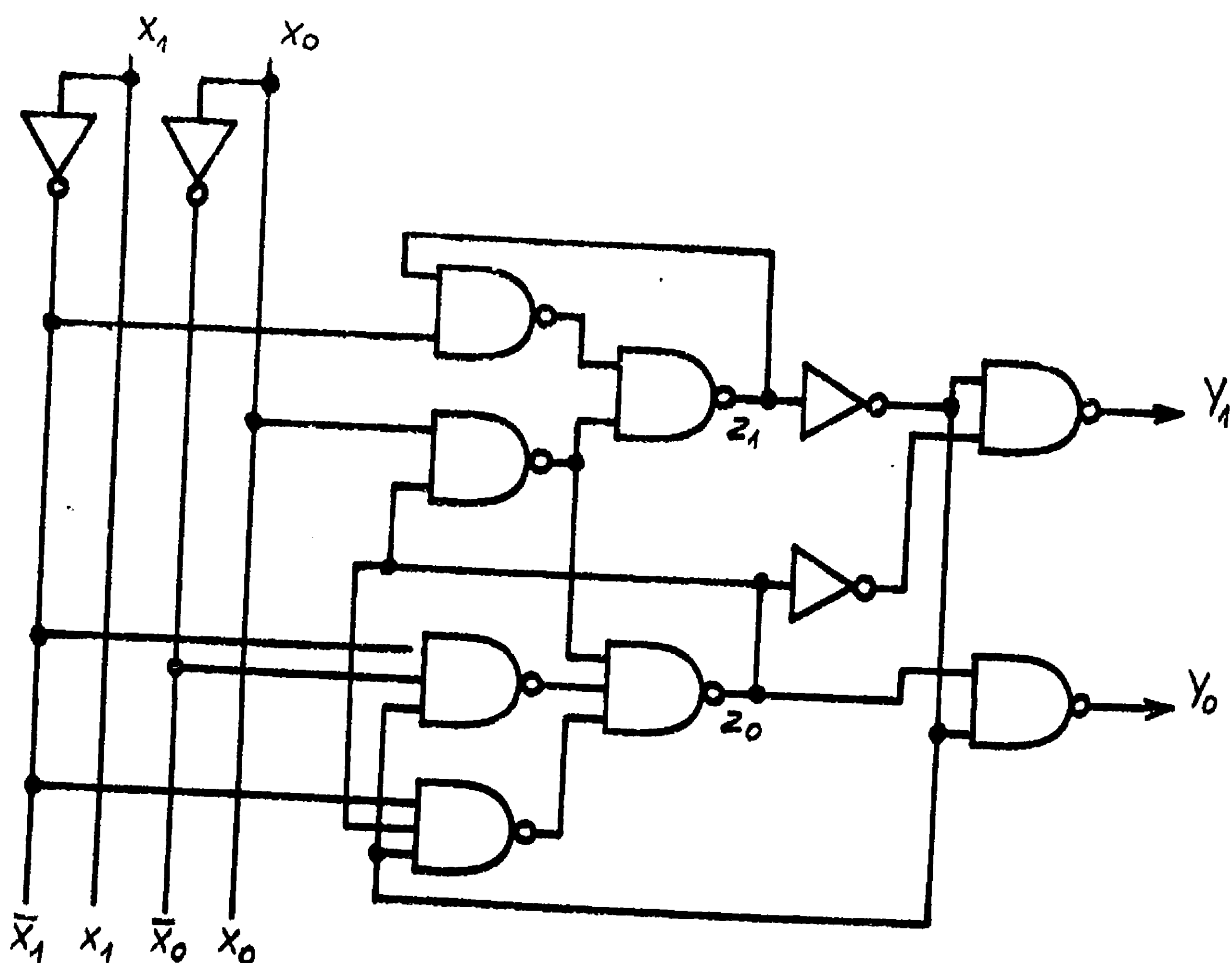
$$z_0^t = \bar{s}_1^t \bar{x}_1 \bar{x}_0 + s_0^t x_0 + (\bar{s}_1^t s_0^t \bar{x}_1),$$

- dla wyjść:

$$y_1 = s_1 + s_0$$

$$y_0 = s_1 + \bar{s}_0$$

Formuła minimalna dla z_0^t zawiera Hazard względem x_0 ; aby go wyeliminować dodano człon w nawiasie. Schemat układu zrealizowany z bramek NAND przedstawiono na rysunku 6.18.



Rys. 6.18. Schemat asynchronicznego układu potencjałowego o działaniu opisanym tablicą pokazaną na rys. 6.17. (przykład 6.4)

Należy zwrócić uwagę na znaczne ograniczenia narzucone na układ sterujący wejściami x_1 , x_0 w związku z koniecznością eliminacji zjawiska wyścigu wejściowego. Powyższa uwaga dotyczy również poprzedniego przykładu.

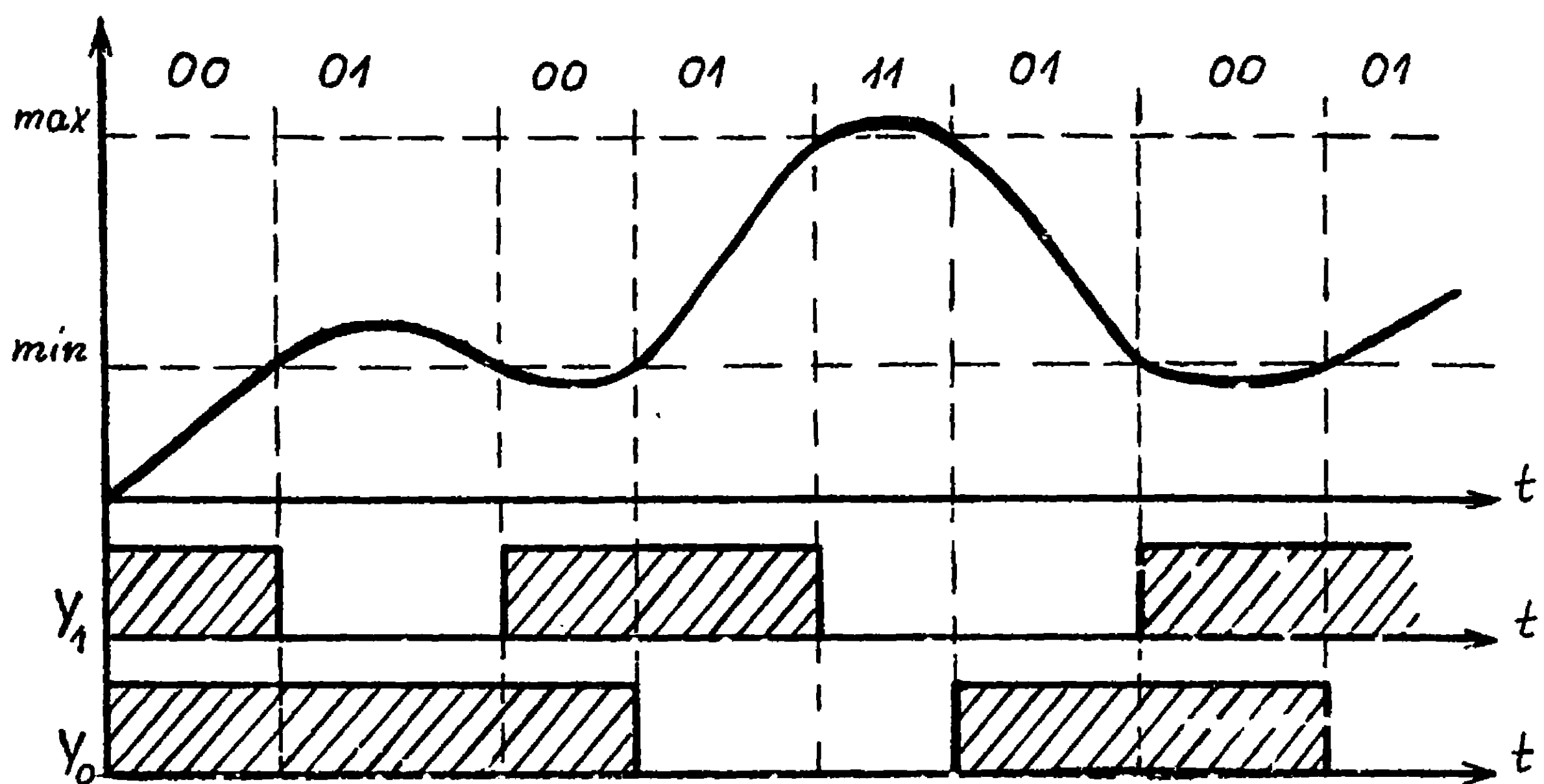
Przykład 6.5. Dwie pompy tłoczą wodę do zbiornika wyrównawczego zasilającego proces technologiczny. Poziom wody jest kontrolowany przez dwa czujniki x_1 , x_0 . Jeśli spada on poniżej minimalnego, to są włączane dwie pompy y_1 , y_0 - sygnalizuje to wartość $\langle x_1, x_0 \rangle = \langle 0, 0 \rangle$. Gdy poziom wody przekroczy wartość dopuszczalną maksymalną, pompy są wyłączane - sygnalizuje to wartość $\langle x_1, x_0 \rangle = \langle 1, 1 \rangle$. Jeśli lustro wody znajduje się między poziomem maksymalnym i minimalnym, to jest włączona tylko ta pompa, która pracowała w ostatniej sekwencji krócej. Sygnalizuje to wartość $\langle x_1, x_0 \rangle = \langle 0, 1 \rangle$. Zaprojektować układ stabilizujący poziom wody w zbiorniku wyrównawczym.

Aby uwzględnić wszystkie możliwe sytuacje należy prześledzić pracę układu od momentu, gdy zbiornik jest pusty, a więc gdy $x_1 = 0$, $x_2 = 0$. Obie pompy są wówczas włączone, czyli $y_1 = 1$, $y_0 = 1$. Jeśli poziom wody przekroczy minimum - zgodnie z warunkami zadania - wyłączy się ta pompa, która uprzednio pracowała dłużej. Pompy po starcie pracują jednakowo długo, przyjmijmy zatem, że wyłączy się pompa y_1 , a pompa y_0 będzie pracować nadal. Jeśli teraz poziom wody znów spadnie poniżej minimalnego, to włączy się dodatkowo pompa y_1 i tym razem po przekroczeniu minimum wyłączy się już pompa y_0 , bowiem pracuje dłużej. Mechanizm ten dla różnych sytuacji zilustrowano na rysunku 6.19. Na rysunku tym zaznaczono stany wejść x_1, x_0 dla poszczególnych sekwencji.

Układ może znajdować się w następujących sytuacjach:

- 1) start - obie pompy pracują, potem gdy poziom wody przekroczy minimum pracuje tylko y_0 ,
- 2) obie pompy pracują, uprzednio pracowała tylko y_0 ,
- 3) obie pompy pracują, uprzednio pracowała tylko y_1 ,
- 4) pracuje tylko y_0 ,
- 5) pracuje tylko y_1 ,
- 6) obie pompy nie pracują, uprzednio pracowała y_0 ,
- 7) obie pompy nie pracują, uprzednio pracowała y_1 .

Wymienione sytuacje tworzą bezpośrednio stany wewnętrzne automatu zrealizowanego jako automat typu Moore'a. Stan 1, charakteryzuje się tym, że obie pompy pracują. W kolumnie wyjściowej wartość wektora wynosi w związku z tym $\langle 1, 1 \rangle$. Dopóki poziom wody w zbiorniku jest po-



Rys. 6.19. Wykres czasowy do przykładu 6.5

a)

$s_1 s_0$	$x_1 x_0$				$y_1 y_0$
	00	01	11	10	
1	①	4	—	—	11
2	②	5	—	—	11
3	③	4	—	—	11
4	2	④	6	—	01
5	3	⑤	7	—	10
6	—	5	⑥	—	00
7	—	4	⑦	—	00

$z_1 z_0$

b)

$s_1 s_0$	$x_1 x_0$				$y_1 y_0$
	00	01	11	10	
$p_1(1,3)$	①	p_3	—	—	11
$p_2(2)$	②	p_4	—	—	11
$p_3(4)$	p_2	③	p_5	—	01
$p_4(5)$	p_1	④	p_6	—	10
$p_5(6)$	—	p_4	⑤	—	00
$p_6(7)$	—	p_3	⑥	—	00

$z_1 z_0$

Rys. 6.20. Tablice przejść - wyjść automatów pierwotnych z przykładu 6.5

nijżej minimalnego, oba sygnały wejściowe są równe zeru, czyli $x_1 = 0$, $x_0 = 0$. W rubryce stanów następnych dla tej wartości wektora, układ będąc w stanie 1 pozostaje w nim nadal. Gdy poziom wody przekroczy minimum, sygnał $x_0 = 1$, a stanem następnym dla stanu 1 będzie stan 4, gdyż zgodnie z założeniem - pompa y_1 zostanie wyłączona, a więc jest to sy-

tuacja opisana pod numerem 4. Z uwagi na bezwładności fizyczne nie ma możliwości przejścia ze stanu 1 do stanu następnego w kolumnie wartości wektora wejściowego $x_1 = 1$, $x_0 = 1$ (nagłe zapełnienie zbiornika nie jest możliwe), a więc w daną kratkę można wpisać kreskę. Również kreski wpisujemy we wszystkie kratki $x_1 = 1$, $x_0 = 0$, gdyż taka wartość sygnałów sterujących nie może wystąpić. W powyższy sposób wypełniamy całą tabelę przejść - wyjść (rys. 6.20a). Poddając tak otrzymany automat pierwotny procesowi minimalizacji, otrzymamy następujące wyrażenie określające rodzinę maksymalnych zbiorów stanów niesprzecznych (patrz p. 4.4)

$$\{Q_{\max}\} = \left\{ \{1, 3\}, \{2\}, \{4\}, \{5\}, \{6\}, \{7\} \right\}.$$

Jest to jednocześnie rodzina $\{P_j\}$. Przypisując zbiorom nazwy kolejno od p_1 do p_6 otrzymamy nową wersję automatu (rys. 6.20b). Jest to automat minimalny, a więc dalsza jego redukcja jako automatu typu Moore'a nie jest możliwa. Można natomiast tabelę zredukować, jeśli określimy automat podobny typu Mealy'ego. Na rysunku 6.21 przedstawiono tabelę przejść - wyjść automatu podobnego typu Mealy'ego. Rodzina $\{Q_{\max}\}$ tego automatu ma postać

$$\{Q_{\max}\} = \left\{ \{p_1, p_6\}, \{p_2, p_5\}, \{p_3\}, \{p_4\} \right\}.$$

$x_1 x_0$	$x_1 x_0$				$x_1 x_0$			
	00	01	11	10	00	01	11	10
p_1	p_1	p_3	-	-	11	01	--	--
p_2	p_2	p_4	-	-	11	10	--	--
p_3	p_2	p_3	p_5	-	11	01	00	--
p_4	p_1	p_4	p_6	-	11	10	00	--
p_5	-	p_4	p_5	-	--	10	00	--
p_6	-	p_3	p_6	-	--	01	00	--
					$y_1 y_0$			

Rys. 6.21. Tablica przejść - wyjść automatu Mealy'ego podobnego do automatu Moore'a z rys. 6.20b

Jest to jednocześnie rodzina P_j , gdyż wszystkie zbiory są niezbędne do pokrycia stanów automatu pierwotnego. Wprowadzając przyporządkowanie:

$$m_1 \rightarrow \{p_1, p_6\},$$

$$m_2 \rightarrow \{p_2, p_5\},$$

$$m_3 \rightarrow \{p_3\},$$

$$m_4 \rightarrow \{p_4\},$$

otrzymamy automat minimalny, opisany tabelą przejść - wyjść pokazaną na rysunku 6.22. Na rysunku 6.23 przedstawiono graf oraz tabelę przejść - wyjść układu logicznego otrzymaną po zakodowaniu stanów zgodnie z regułą hipersześcianu, którym w omawianym przypadku jest kwadrat. Po szczególnym stanom przypisano kody:

$$m_1 \rightarrow 00,$$

$$m_2 \rightarrow 11,$$

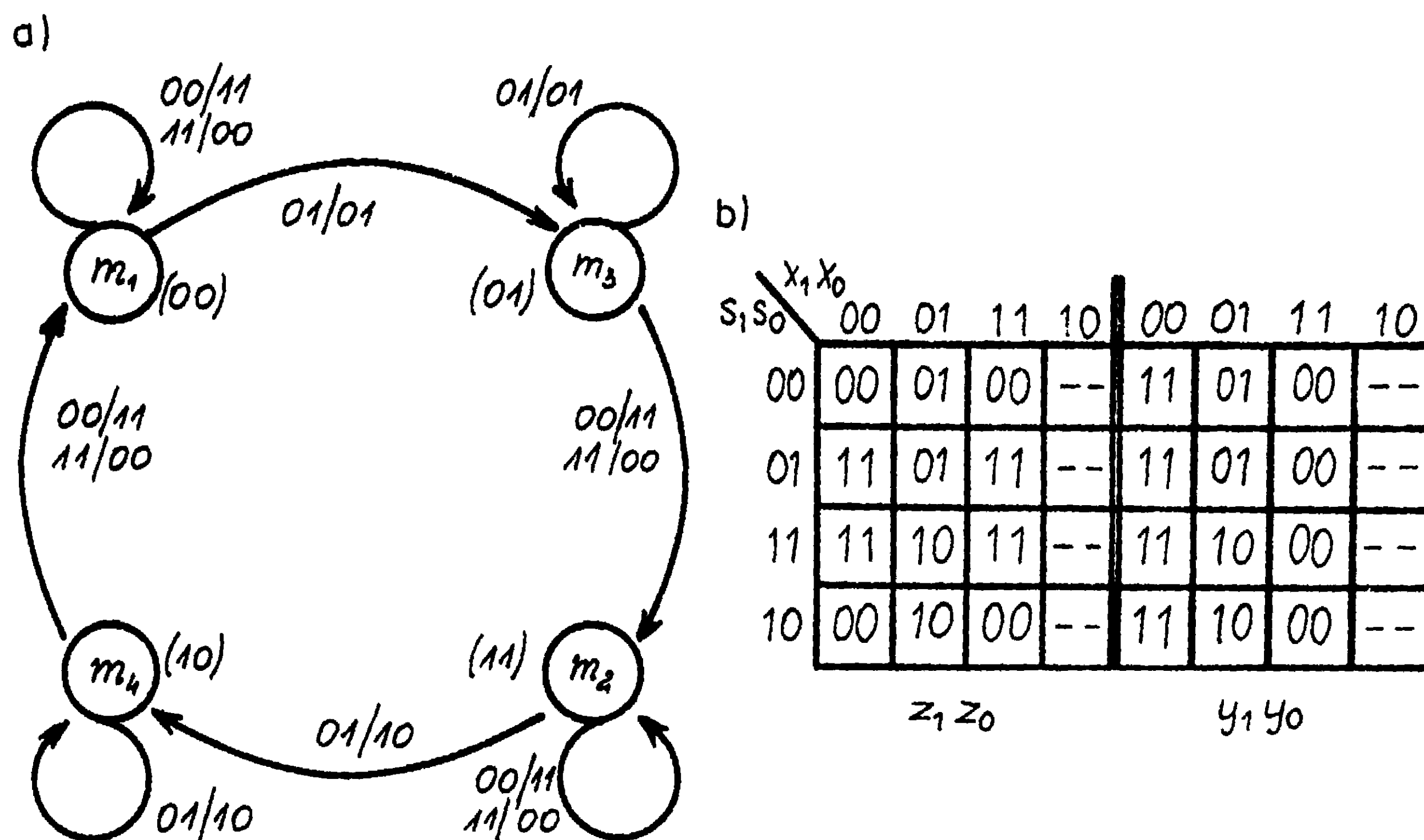
$$m_3 \rightarrow 01,$$

$$m_4 \rightarrow 10.$$

$x_1 x_0$									
		00	01	11	10	00	01	11	10
$m_1(p_1, p_6)$	m_1	m_3	m_1	—	11	01	00	—	—
$m_2(p_2, p_5)$	m_2	m_4	m_2	—	11	10	00	—	—
$m_3(p_3)$	m_2	m_3	m_2	—	11	01	00	—	—
$m_4(p_4)$	m_1	m_4	m_1	—	11	10	00	—	—

$y_1 y_0$

Rys. 6.22. Tablica przejść - wyjść automatu minimalnego z przykładu 6.5



Rys. 6.23. Graf oraz tablica przejść - wyjść układu logicznego realizującego automat minimalny z przykładu 6.5 (a,b)

Formuły boolowskie określone na podstawie tablicy przejść - wyjść dla jedynek funkcji przyjmą postać:

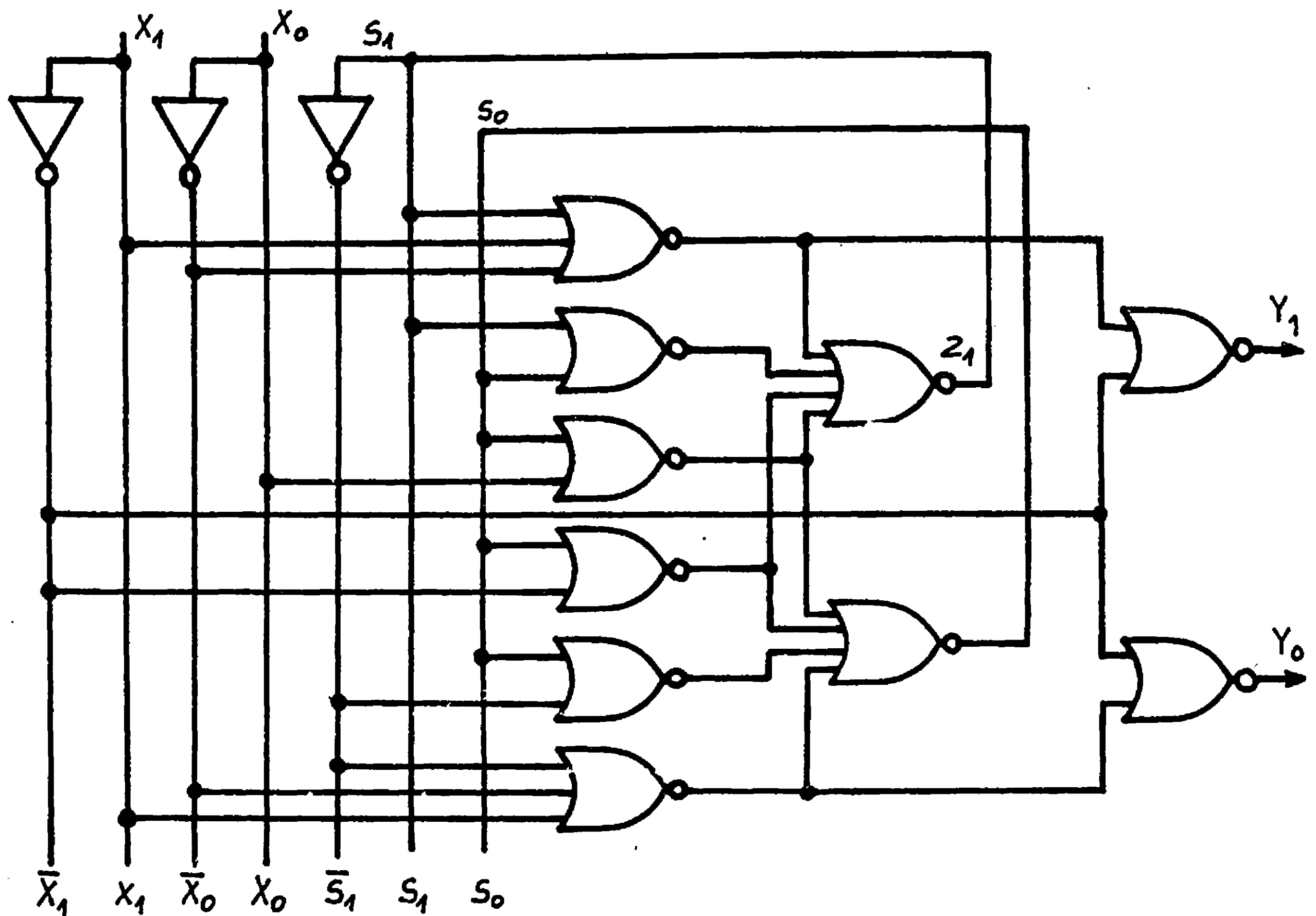
$$z_1^t = s_1^{t+1} = s_0^t \bar{x}_0 + s_0^t x_1 + s_1^t \bar{x}_1 x_0 + (s_1^t s_0^t),$$

$$z_0^t = s_0^{t+1} = s_0^t \bar{x}_0 + s_0^t x_1 + \bar{s}_1^t \bar{x}_1 x_0 + (\bar{s}_1^t s_0^t),$$

$$y_1 = \bar{x}_0 + s_1 \bar{x}_1 (x_0),$$

$$y_0 = \bar{x}_0 + \bar{s}_1 \bar{x}_1 (x_0).$$

Dla wyjść z_1 oraz z_0 do formuł minimalnych dodano człony w nawiasach celem eliminacji hazardu względem x_1 oraz x_0 , występującego przy przejściach ze stanów $\langle 1,1 \rangle$ i $\langle 0,1 \rangle$. Jeśli do formuł minimalnych uzyskanych z tabel dla wyjść y_1 , y_0 dołączyć sygnał x_0 (co zaznaczono nawiasem), to uzyskane iloczyny składowe są takie same jak w przypadku wyjść z_1 i z_0 , a więc nie trzeba ich tworzyć osobno.



Rys. 6.24. Schemat układu sterowania pomp z przykładu 6.5

Analogiczny rezultat zostanie osiągnięty dla zer funkcji. Formuły optymalne przybiorą wówczas postać:

$$z_1^t = s_1^{t+1} = (s_0^t + x_0)(s_0^t + \bar{x}_1)(s_1^t + x_1 + \bar{x}_0) [s_1^t + s_0^t],$$

$$z_0^t = s_0^{t+1} = (s_0^t + x_0)(s_0^t + \bar{x}_1)(\bar{s}_1^t + x_1 + \bar{x}_0) [\bar{s}_1^t + s_0^t],$$

$$y_1 = \bar{x}_1(s_1 + (x_1) + \bar{x}_0),$$

$$y_0 = \bar{x}_1(\bar{s}_1 + (x_1) + \bar{x}_0).$$

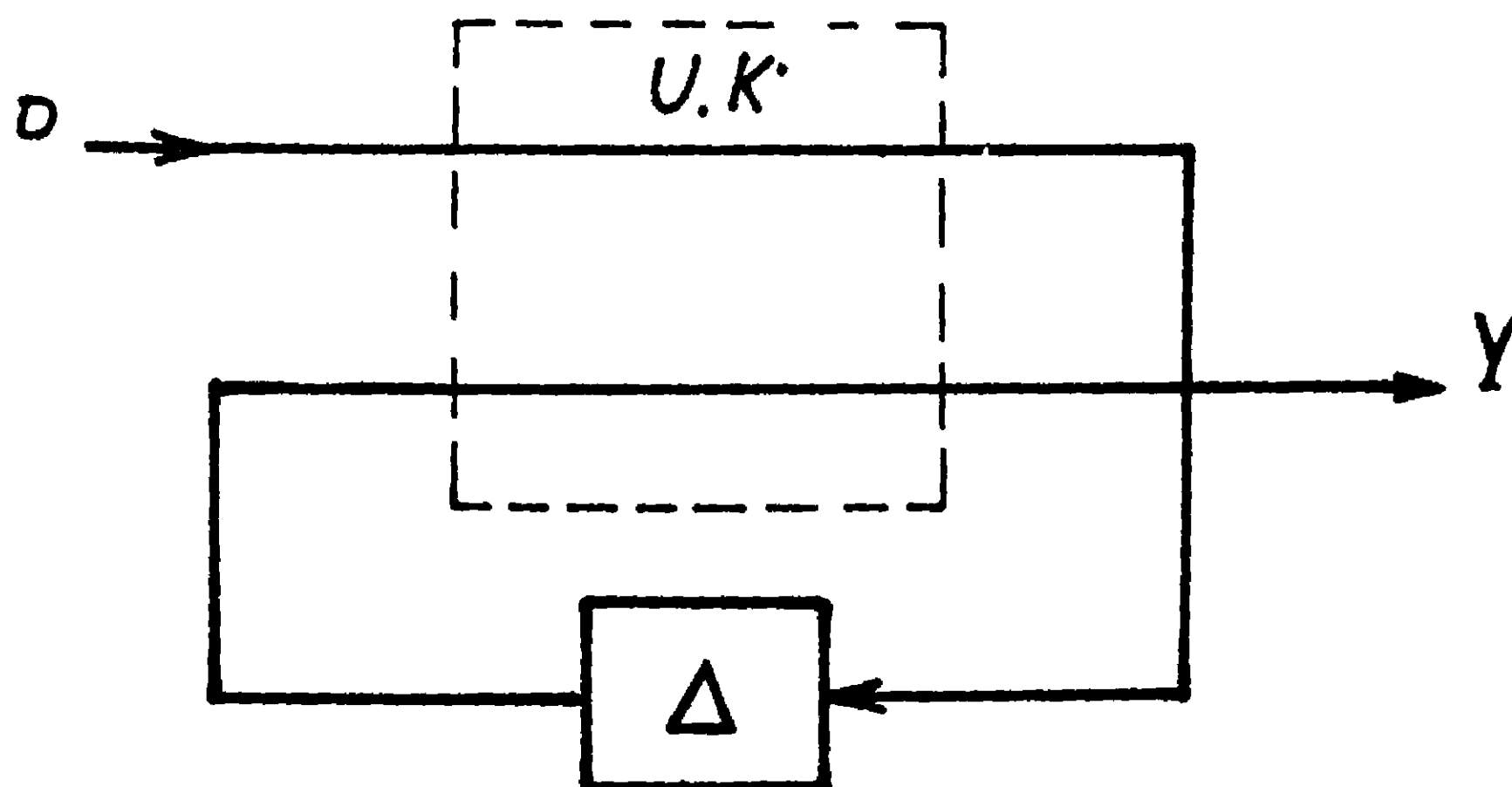
W dodatkowych nawiasach podano człony eliminujące hazard oraz wprowadzające człony wspólne. Na rysunku 6.24 przedstawiono schemat układu sterującego pompami zrealizowany z bramek NOR. Układ nie zawiera hazardu i nie występuje w nim zjawisko wyścigu wewnętrznego. Wyścig nie może być też wprowadzony przez wejścia, gdyż z fizycznej realizacji sygnałów wynika, że mogą się one zmieniać tylko i wyłącznie wg schematu

$$\langle 0,0 \rangle \rightleftharpoons \langle 0,1 \rangle \rightleftharpoons \langle 1,1 \rangle,$$

a więc zmiany występują każdorazowo tylko dla jednej składowej wektora wejściowego. Poddając formuły opisujące układ niewprowadzającym hazardu (patrz p. 3.9) manipulacjom algebraicznym, można uzyskać wielopoziomowe prostsze schematy logiczne niż schemat przedstawiony na rysunku 6.24. Tego typu zabiegi jednak najlepiej wykonywać mając ściśle sprecyzowane zestawy bramek scalonych.

Nie była dotąd dyskutowana sprawa wyścigu wprowadzonego przez wyjścia y_1, y_0 . Z analizy grafu pokazanego na rysunku 6.23b wynika, że podczas możliwych przejść, zmiany wektora $\langle y_1, y_0 \rangle$ następują też tylko na jednej pozycji, a więc wyścigu wyjścia nie wprowadzą. Ma to zresztą w tym przypadku niewielkie znaczenie z uwagi na relatywnie dużą bezwładność pomp sterowanych z wyjść y_1 i y_0 . Urządzenie sterowane sygnałami z zaprojektowanego układu jest więc niewrażliwe na wyścig wejściowy.

Przykład 6.6. Zaprojektować asynchroniczne odpowiedniki przerzutników synchronicznych omówionych w punkcie 5.2.



Rys. 6.25. Asynchroniczny przerzutnik typu D jako opóźnienie Δ (przykład 6.6)

Przerzutnik asynchroniczny typu RS odgrywa szczególną rolę zarówno w układach asynchronicznych jak i synchronicznych i dlatego został osobno omówiony w przykładzie 6.3. Przerzutnik typu D w układzie asynchronicznym jest reprezentowany tylko przez opóźnienie Δ (rys. 6.25), a więc problem jego syntezy nie występuje. Odpowiednikiem bloku kombinacyjnego są tu bezpośrednie połączenia wejść i wyjść bez obecności

bramek. Graf przerzutnika typu T pokazany na rysunku 5.16 i powtórzo-
ny na rysunku 6.26a nie zawiera stanów stabilnych. Modyfikując go do
postaci przedstawionej na rysunku 6.26b, otrzymujemy graf asynchro-
niczny, realizujący w kategoriach wejścia - wyjścia te same funkcje
co i graf przerzutnika typu T. Kodując

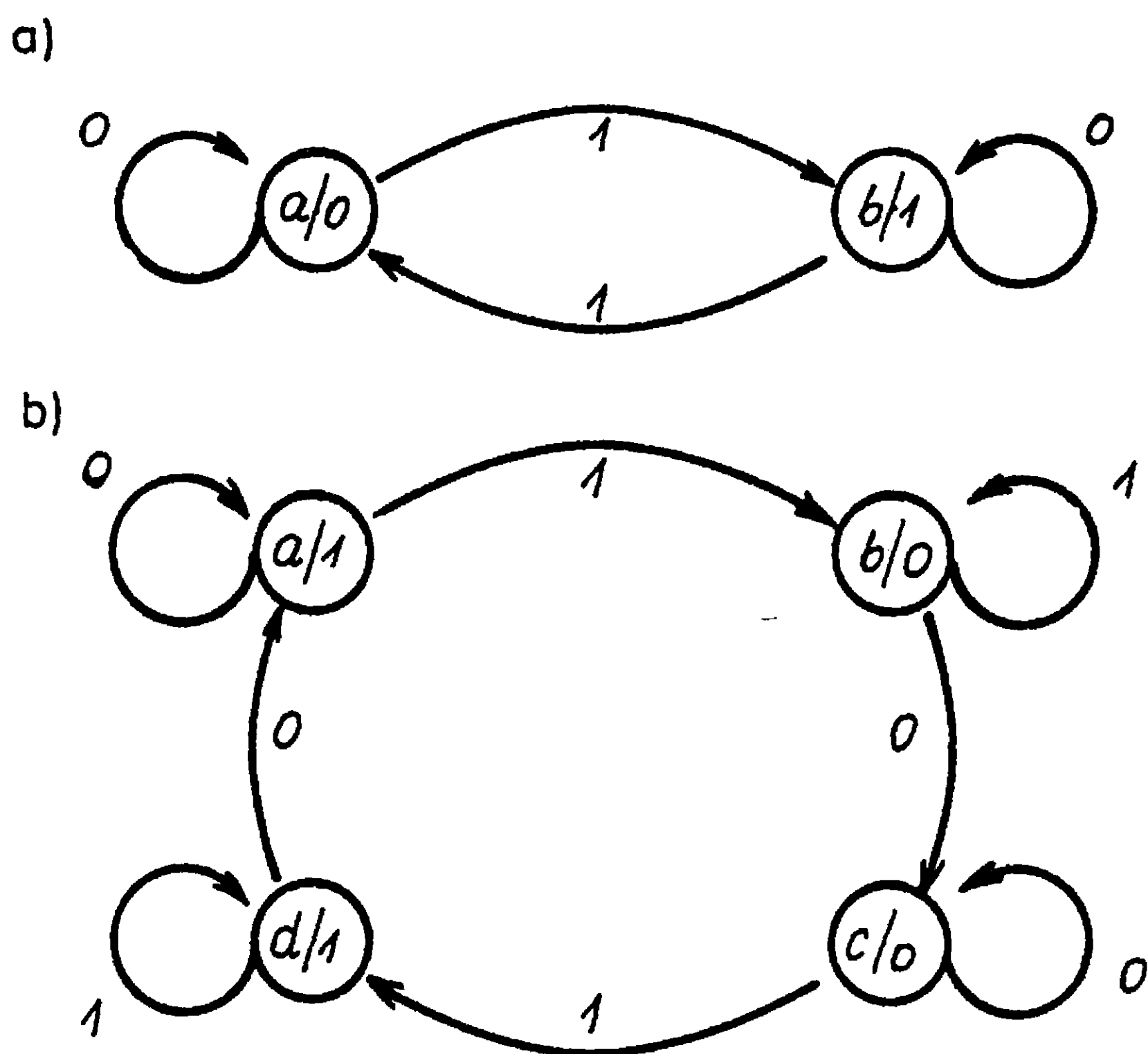
$a \rightarrow 00,$

$b \rightarrow 01,$

$c \rightarrow 11,$

$d \rightarrow 10,$

otrzymujemy tablicę przejść - wyjść oraz graf układu omówionego w przy-
kładzie 6.2.

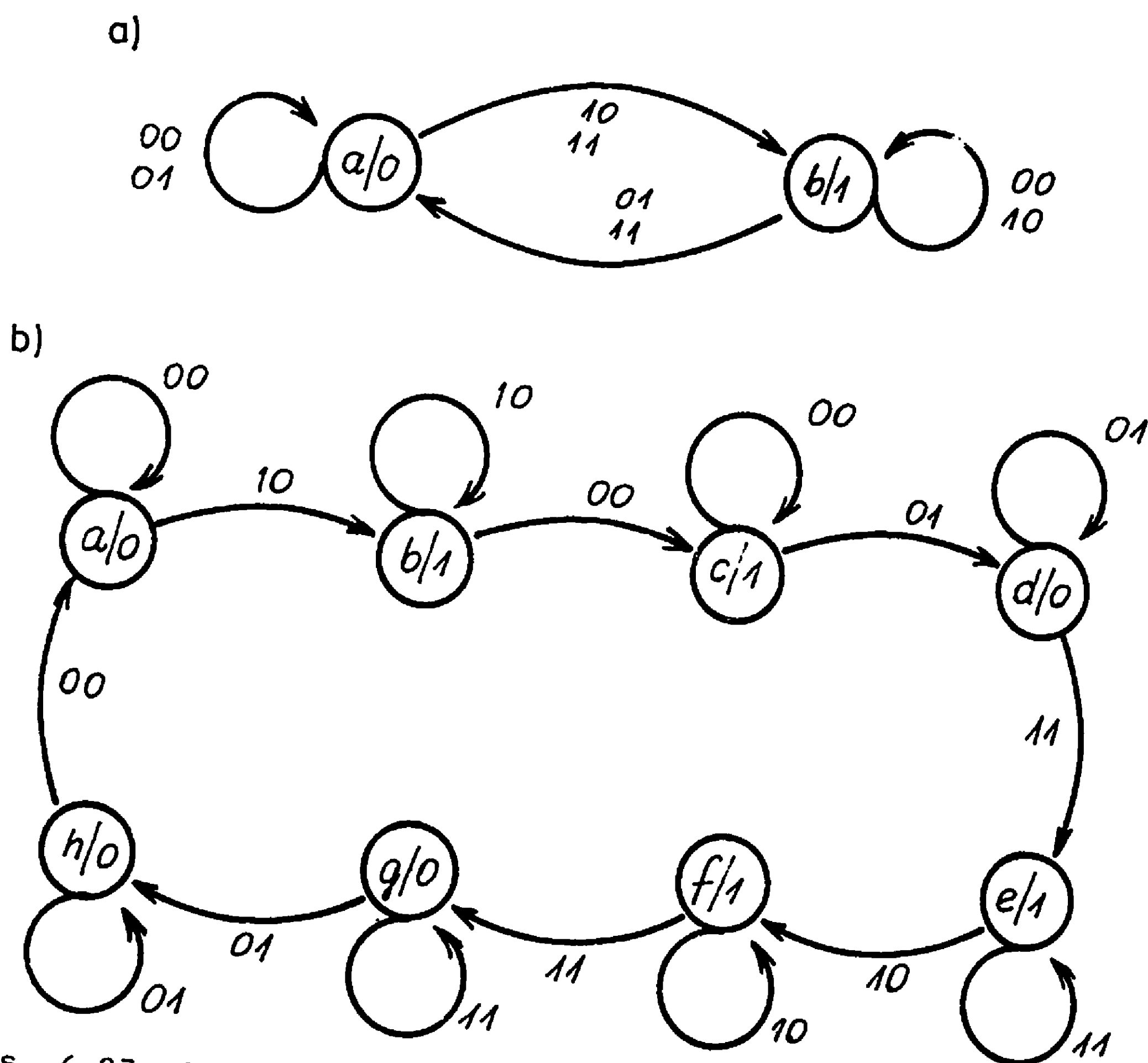


Rys. 6.26. Graf synchronicznego (a) i asynchro-
nicznego (b) przerzutnika typu T (przykład 6.6)

Graf przerzutnika synchronicznego typu JK (rys. 6.27a) jest nie-
stabilny dla przejść $\langle JK \rangle = \langle 11 \rangle$. Utwórzmy na jego podstawie graf,
w którym przejścia zarówno w ramach pętli własnej jak i między stana-
mi odbywać się będą tylko dla jednej wartości wektora wejściowego.
Graf ten pokazano na rysunku 6.27b. Zawiera on osiem stabilnych sta-

nów wewnętrznych i w kategoriach wejścia i wyjścia realizuje te same funkcje co i graf synchronicznego przerzutnika typu JK. Tabelę przejść - wyjść odpowiadającą grafowi pokazano na rysunku 6.28. Reprezentuje ona automat niezupełny, który może być poddany dalszej redukcji. Rodzina maksymalnych zbiorów stanów niesprzecznych określona dla tej tabeli ma postać

$$\{Q_{\max}\} = \left\{ \{b, c, e\}, \{b, c, f\}, \{a, g, h\}, \{a, d, h\} \right\}.$$



Rys. 6.27. Graf synchronicznego (a) i asynchronicznego (b) przerzutnika typu JK (przykład 6.6)

Ponieważ do pokrycia wszystkich stanów automatu pierwotnego potrzebne są wszystkie zbiory, więc rodzina $\{Q_{\max}\}$ jest jednocześnie rodziną $\{P_j\}$. Przypisując zbiorom rodziny $\{P_j\}$ stany wewnętrzne automatu ilorazowego (minimalnego) w następujący sposób:

$$\begin{aligned}
 p_1 &\rightarrow \{b \ c \ e\}, \\
 p_2 &\rightarrow \{b \ c \ f\}, \\
 p_3 &\rightarrow \{a \ g \ h\}, \\
 p_4 &\rightarrow \{a \ d \ h\}
 \end{aligned}$$

otrzymamy tabelę automatu ilorazowego (minimalnego) pokazaną na rysunku 6.29. Jak wynika z rysunku, istnieje wiele wariantów automatów minimalnych, odpowiadających asynchronicznemu przerzutnikowi typu JK. Najbardziej jednolitą postać ma wariant, którego tablicę przejść - wyjść pokazano na rysunku 6.30. Kodując stany wewnętrzne wg schematu:

$$\begin{aligned}
 p_1 &\rightarrow 00, \\
 p_2 &\rightarrow 01, \\
 p_3 &\rightarrow 11, \\
 p_4 &\rightarrow 10
 \end{aligned}$$

JK					
	00	01	11	10	
a	a	-	-	b	0
b	c	-	-	b	1
c	c	d	-	-	1
d	-	d	e	-	0
e	-	-	e	f	1
f	-	-	g	f	1
g	-	h	g	-	0
h	a	h	-	-	0

Rys. 6.28. Tablica przejść - wyjść odpowiadająca grafowi asynchronicznego przerzutnika typu JK (przykład 6.6)

JK					
	00	01	11	10	
$p_1(bce)$	$p_1(p_2)$	p_4	p_1	p_2	1
$p_2(bcf)$	$p_1(p_2)$	p_4	p_3	p_2	1
$p_3(agh)$	$p_3(p_4)$	$p_3(p_4)$	p_3	$p_1(p_2)$	0
$p_4(adh)$	$p_3(p_4)$	p_4	p_1	$p_1(p_2)$	0

Rys. 6.29. Tablica automatu minimalnego odpowiadającego asynchronicznemu przerzutnikowi typu JK (przykład 6.6)

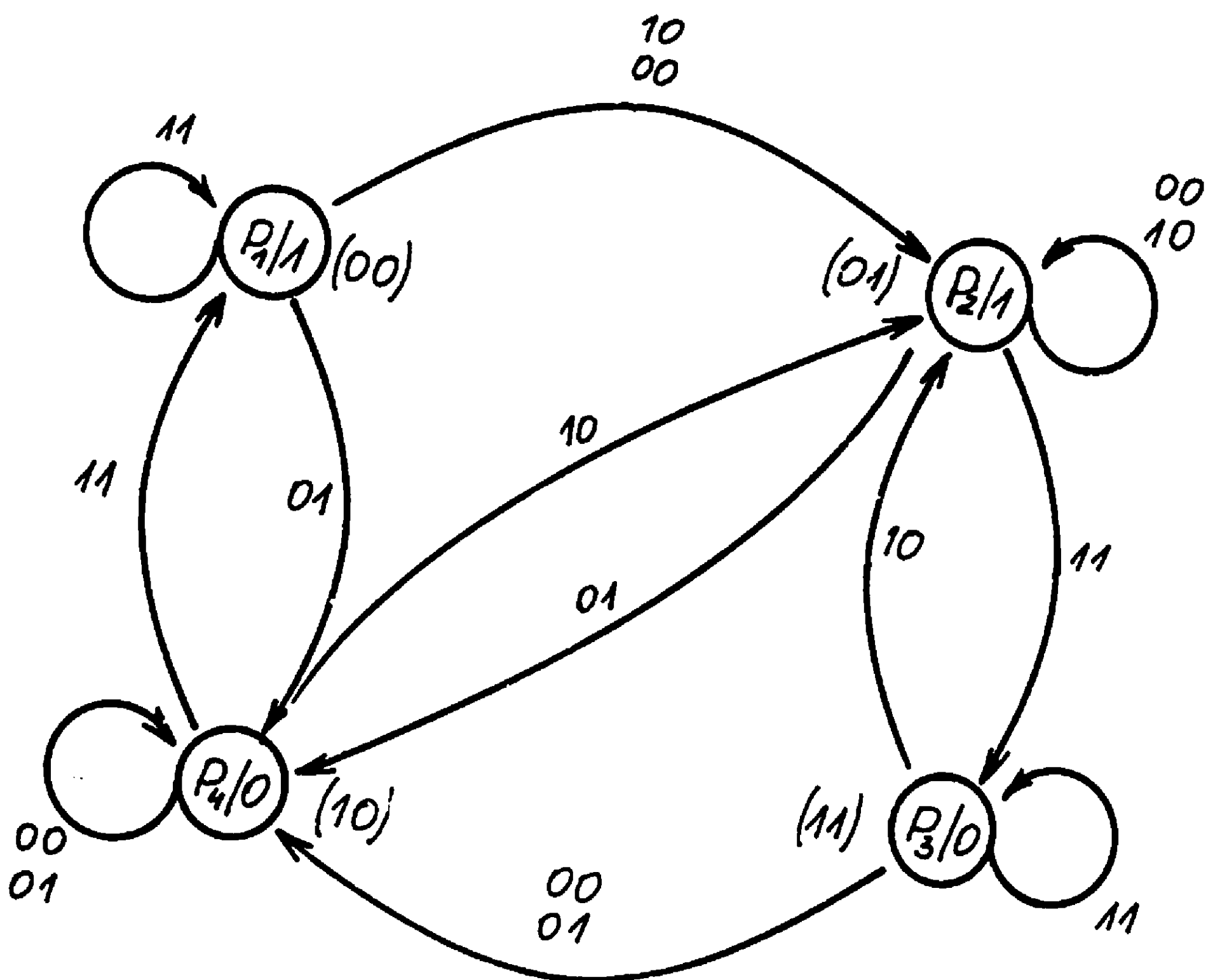
JK					
	00	01	11	10	
P_1	P_2	P_4	$\textcircled{P_1}$	P_2	1
P_2	$\textcircled{P_2}$	P_4	P_3	$\textcircled{P_2}$	1
P_3	P_4	P_4	$\textcircled{P_3}$	P_2	0
P_4	$\textcircled{P_4}$	$\textcircled{P_4}$	P_1	P_2	0

Rys. 6.30. Wybrany wariant automatu odpowiadający asynchronicznemu przerzutnikowi typu JK (przykład 6.6)

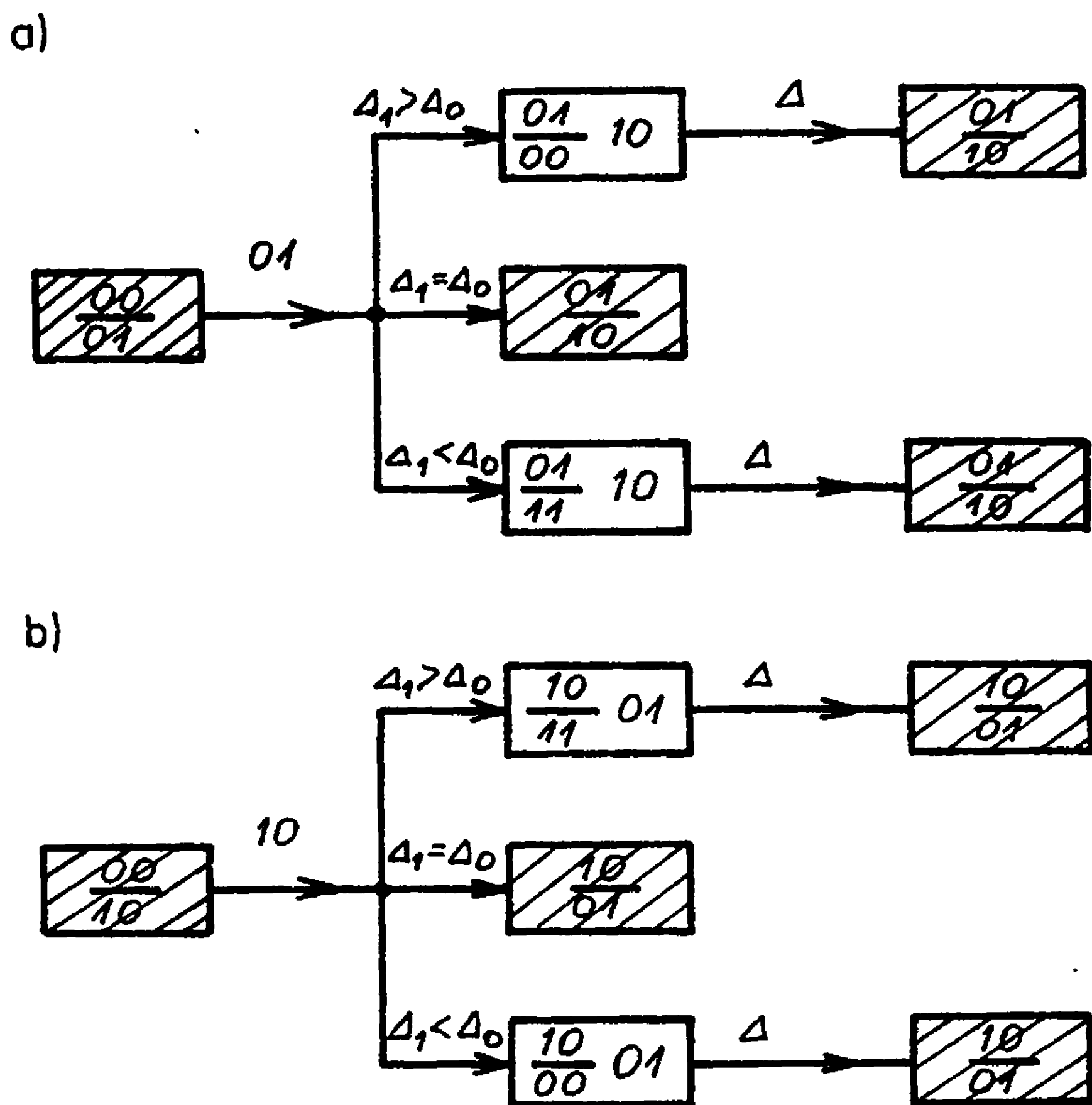
$S_1 S_0$					
	00	01	11	10	
00	01	10	00	01	1
01	01	10	11	01	1
11	10	10	11	01	0
10	10	10	00	01	0

$z_1 z_0$ y

Rys. 6.31. Tablica przejść - wyjść wybranego wariantu asynchronicznego przerzutnika typu JK (przykład 6.6)



Rys. 6.32. Graf asynchronicznego przerzutnika typu JK (przykład 6.6)



Rys. 6.33. Zjawisko wyścigu niekrytycznego w asynchronicznym przerzutniku typu JK (przykład 6.6), (a,b)

otrzymamy tablicę przejść - wyjść układu asynchronicznego pokazaną na rysunku 6.31. Tablicy odpowiada graf pokazany na rysunku 6.32. Jak wynika z rysunku grafu nie da się bezpośrednio zakodować w ten sposób, by przejścia między stanami odbywały się wyłącznie wzdłuż krawędzi kwadratu jednostkowego. Podczas przejścia między stanami p_2 (01) i p_4 (10) wystąpi zjawisko wyścigu. Analiza pracy grafu wskazuje jednak na to, że wyścig ten jest wyścigiem strukturalnie niekrytycznym. Istotnie, ze stanu p_4 (10), przechodzimy do stanu p_2 (01) pobudzając układ wektorem wejściowym o wartości $\langle JK \rangle = \langle 10 \rangle$. Jednocześnie dla takiego samego pobudzenia przechodzimy do stanu p_2 także z pozostałych stanów, a więc zawsze układ osiągnie stan p_2 , czy to poprzez stan p_1 czy też poprzez stan p_3 . Ilustruje to rysunek 6.33a. Analogiczna sytuacja wystąpi w przypadku pobudzenia układu znajdującego się w stanie p_2 (01) wektorem $\langle JK \rangle = \langle 01 \rangle$. Dla tych wartości pobudzenia układ przechodzi do stanu p_4 (10) z każdego z pozostałych stanów. Ilustruje to rysunek 6.33b. Omawiany wyścig nie wprowadza również na

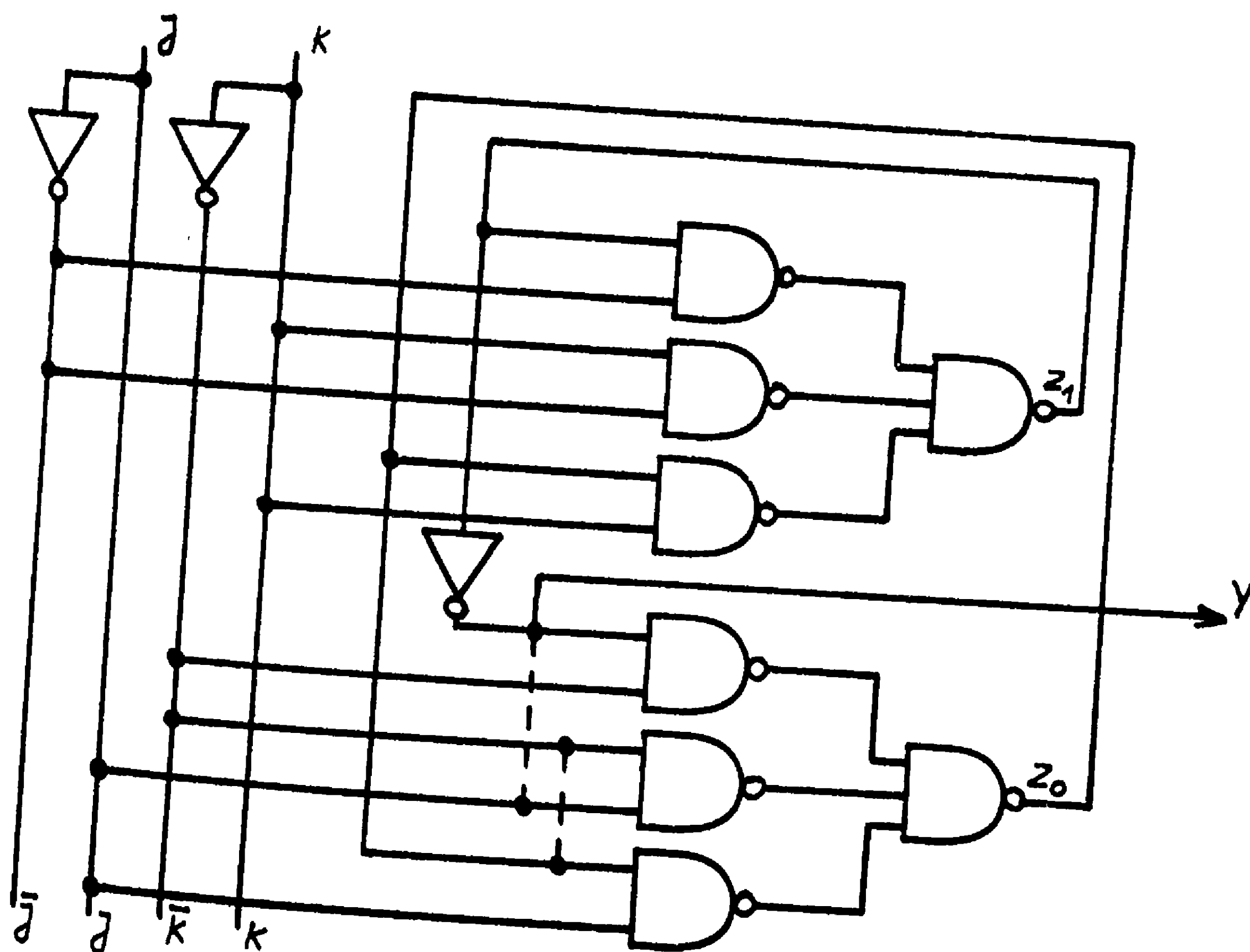
wyście dodatkowych impulsów typu: "krótka jedynka" czy też krótkie zero". Podczas przejścia $p_4 \rightarrow p_2$, poprzez stan p_1 , wyście osiągnie stan 1 nieco wcześniej niż podczas przejścia poprzez stan p_3 . Podobnie dla przejścia $p_2 \rightarrow p_4$, poprzez stan p_1 , zero na wyście pojawi się nieco później niż podczas przejścia przez stan p_3 . Graf nadaje się więc zasadniczo do asynchronicznej realizacji bramkowej. Formuły opisujące poszczególne wyście, wyznaczone na podstawie tabeli przejść - wyście, mają postać:

$$z_1^t = s_1^{t+1} = s_1^t \bar{J} + \bar{J} K + s_0^t K,$$

$$z_0^t = s_0^{t+1} = \bar{s}_1^t \bar{K} + J \bar{K} + s_0^t J,$$

$$y = \bar{s}_1.$$

Nie zawierają one hazardu. Realizację z bramek NAND asynchronicznego przerzutnika typu JK przedstawiono na rysunku 6.34.

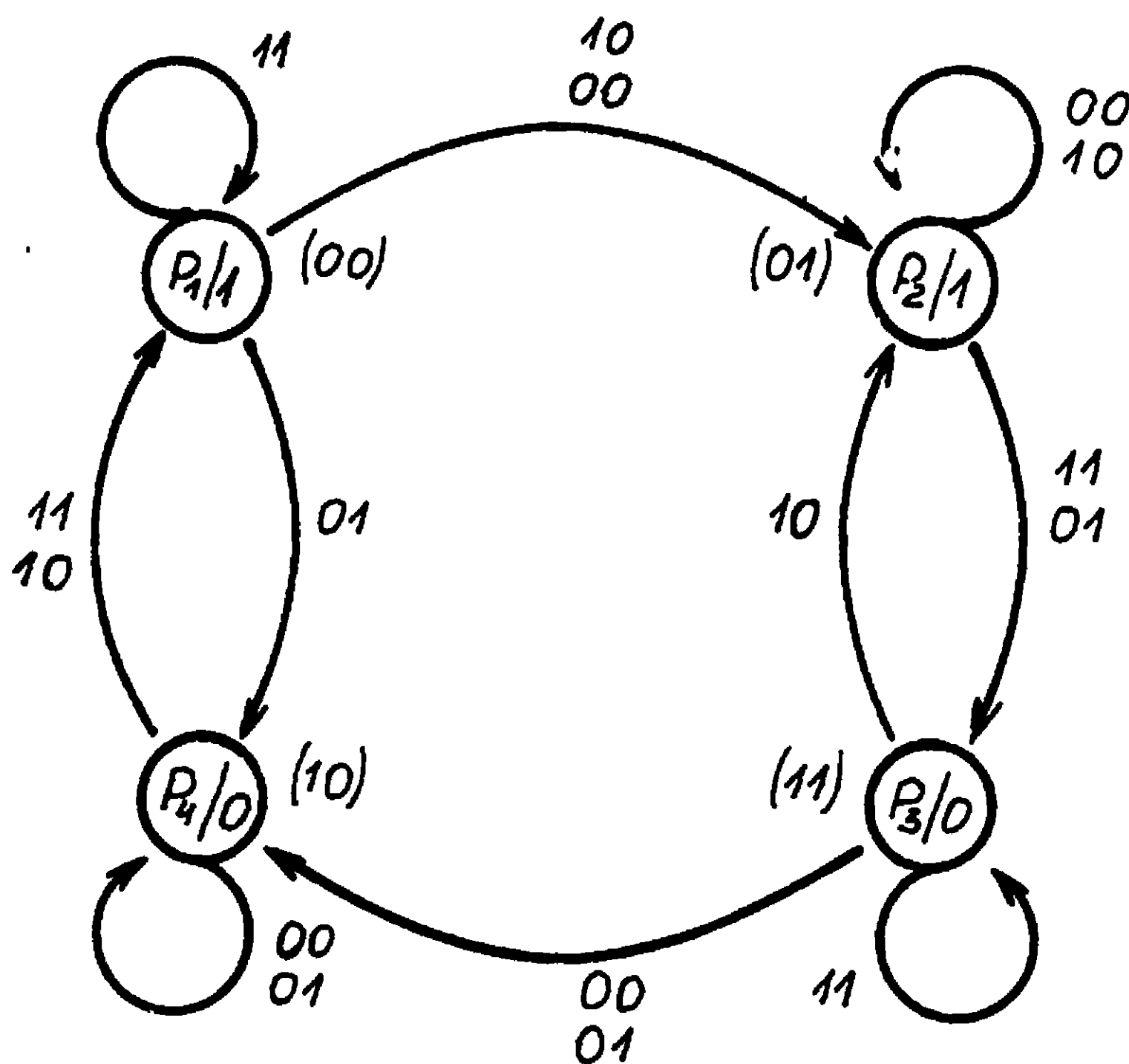


Rys. 6.34. Realizacja z bramek NAND asynchronicznego przerzutnika typu JK (przykład 6.6)

Graf pokazany na rysunku 6.32 można zmodyfikować w ten sposób, by wyeliminować omówiony wyścig niekrytyczny. Wystarczy w tym celu przejście ze stanu p_2 do p_4 dokonać poprzez stan niestabilny p_3 , natomiast ze stanu p_4 do p_2 poprzez stan niestabilny p_1 . Inaczej mówiąc, w odróżnieniu od przypadku poprzedniego, precyzujemy warunki przejść przez stany niestabilne. Na rysunku 6.35 pokazano zmodyfikowany graf. Ze stanu p_4 po pobudzeniu wartością wektora wejściowego 10, układ przechodzi do stanu niestabilnego p_1 i następnie do stanu stabilnego p_2 . W odwrotną stronę, a więc ze stanu p_2 , układ po pobudzeniu wartością wektora wejściowego 01, poprzez stan niestabilny p_3 , przechodzi do stanu stabilnego p_4 . Łatwo sprawdzić, że formuły wyjściowe dla z_1 oraz y nie ulegną wówczas zmianie, natomiast funkcja wyjściowa z_0 przyjmie postać

$$z_0^t = s_0^{t+1} = \bar{s}_1^t \bar{K} + \bar{s}_1^t s_0^t + s_0^t J.$$

Na schemacie przedstawionym na rysunku 6.34 wystarczy wówczas zmienić tylko połączenia ciągle środkowej bramki drugiego poziomu wyjścia z_0 , na połączenia zaznaczone linią przerywaną.



Rys. 6.35. Graf asynchronicznego przerzutnika JK z eliminacją wyścigu poprzez przejścia przez stany niestabilne (przykład 6.6)

Należy jeszcze dodać, że w odróżnieniu od przerzutników synchronicznych, sekwencje zmian wektora wejściowego przerzutników asynchronicznych nie mogą wprowadzać wyścigu wejściowego. Narzuca to istotne ograniczenia na sposób sterowania wejść zarówno przerzutnika JK jak i RS.

6.3.3. Synteza z zastosowaniem przerzutników asynchronicznych

Projektowanie układu asynchronicznego jest prostsze, jeśli do dyspozycji są scalone przerzutniki asynchroniczne. Procedura syntezy formalnej nie różni się wówczas zasadniczo od omówionej w punkcie 5.6 syntezy układów synchronicznych - należy tylko pamiętać, by równania wejściowe nie zawierały hazardu. Oczywiście proces formalizacji zagadnienia, a więc tzw. synteza właściwa, przebiega w omówiony uprzednio sposób, typowy dla układów asynchronicznych potencjałowych. Tablice wzbudzeń przerzutników asynchronicznych są takie same jak i przerzutników synchronicznych - analogia jest więc pełna. W praktyce jednak najczęściej dysponuje się statycznymi przerzutnikami typu RS zintegrowanymi bądź też realizowanymi z bramek NOR lub NAND. W ostatnim przypadku mamy w zasadzie do czynienia z dwoma wariantami przerzutników: RS - dla bramek NOR oraz $\bar{R}\bar{S}$ - dla bramek NAND (rys. 6.36).

a)			
Q_t	Q_{t+1}	R	S
0	0	X	0
0	1	0	1
1	0	1	0
1	1	0	X

b)			
Q_t	Q_{t+1}	$\bar{R}$	$\bar{S}$
0	0	X	1
0	1	1	0
1	0	0	1
1	1	1	X

Rys. 6.36. Tablice wzbudzeń statycznego przerzutnika RS i $\bar{R}\bar{S}$ (a,b)

Analogicznie jak dla układów synchronicznych procedura projektowa sprowadza się do określenia tabeli wzbudzeń i wyznaczenia formuł sterujących dla wejść przerzutników i formuł dla wyjść kombinacyjnych. Zakłada się przy tym, że wektor wejściowy w kolejnych sekwencjach zmienia się tylko na jednej pozycji i nie wcześniej niż ustala się przebiegi w układzie. Procedurę projektową ilustruje przykład.

Przykład 6.7. Zaprojektować układ sterowania pompami z przykładu 6.5, stosując asynchroniczne przerzutniki RS.

$A_t B_t x_1 x_0$	$A_{t+1} B_{t+1}$	RA	SA	RB	SB	$y_1 y_0$
0 0 0 0	0 0	X	0	X	0	1 1
0 0 0 1	0 1	X	0	0	1	0 1
0 0 1 0	X X	X	X	X	X	X X
0 0 1 1	0 0	X	0	X	0	0 0
0 1 0 0	1 1	0	1	0	X	1 1
0 1 0 1	0 1	X	0	0	X	0 1
0 1 1 0	X X	X	X	X	X	X X
0 1 1 1	1 1	0	1	0	X	0 0
1 0 0 0	0 0	1	0	X	0	1 1
1 0 0 1	1 0	0	X	X	0	1 0
1 0 1 0	X X	X	X	X	X	X X
1 0 1 1	0 0	1	0	X	0	0 0
1 1 0 0	1 1	0	X	0	X	1 1
1 1 0 1	1 0	0	X	1	0	1 0
1 1 1 0	X X	X	X	X	X	X X
1 1 1 1	1 1	0	X	0	X	0 0

Rys. 6.37. Tablica wzbudzeń układu z przykładu 6.7

Na podstawie tabeli przejść - wyjść pokazanej na rysunku 6.23a, tworzymy tabelę wzbudzeń układu. Wyjścia przerzutników określamy nazwami A i B (rys. 6.37). Wartość wektora wejściowego $\langle x_1 x_0 \rangle = \langle 10 \rangle$ w układzie nie wystąpi, dla odpowiednich wierszy wystąpią więc pozycje nieokreślone. Wykorzystując je, uzyskamy następujące formuły minimalne:

$$RA = \bar{B} (x_1 + \bar{x}_0) = \overline{B + x_1 + \bar{x}_0},$$

$$SA = B (x_1 + \bar{x}_0) = \overline{\bar{B} + x_1 + \bar{x}_0},$$

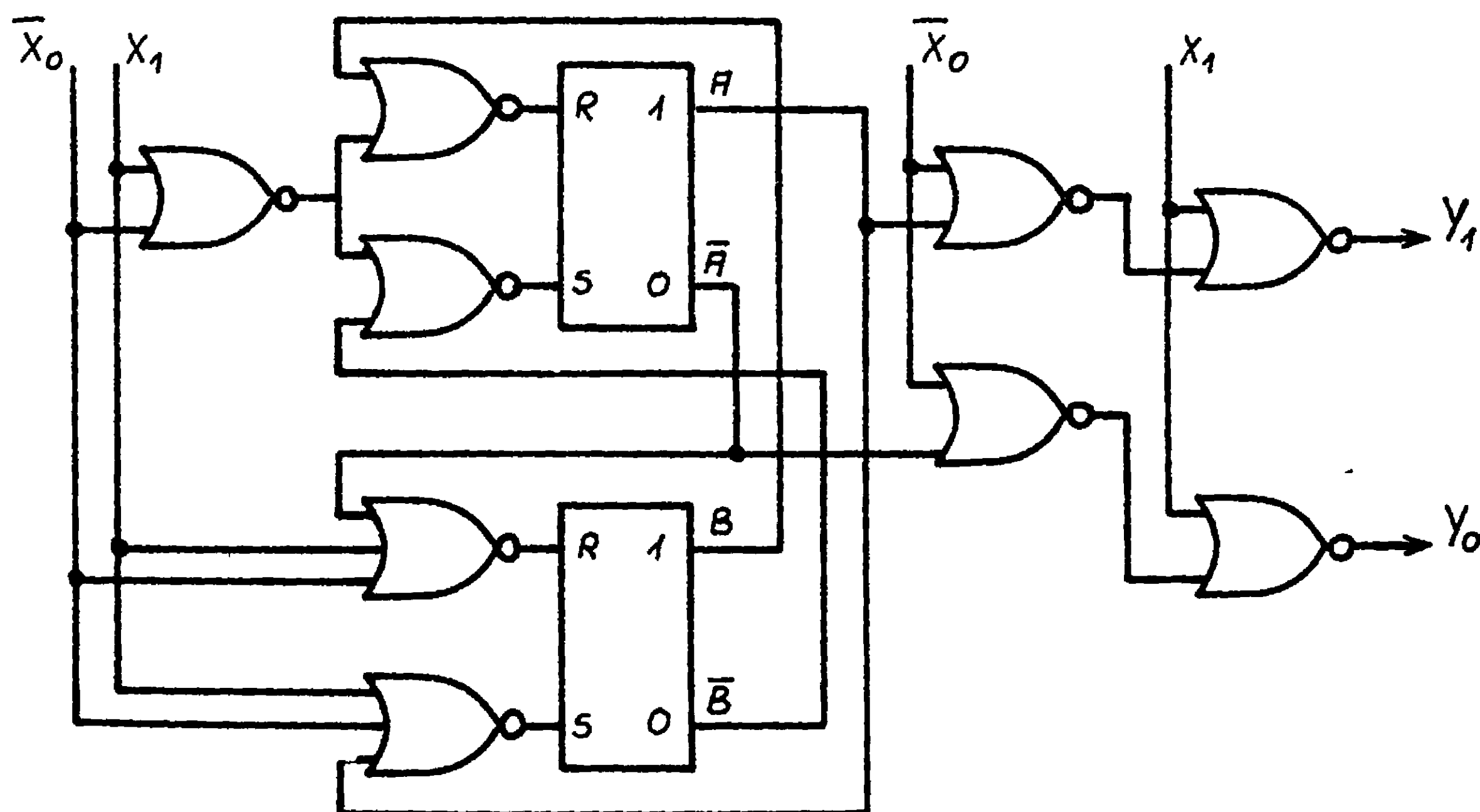
$$RB = A \bar{x}_1 x_0 = \overline{\bar{A} + x_1 + \bar{x}_0},$$

$$SB = \bar{A} \bar{x}_1 x_0 = \overline{A + x_1 + \bar{x}_0},$$

$$y_1 = \bar{x}_1 (A + \bar{x}_0) = \overline{x_1 + \overline{A + \bar{x}_0}},$$

$$y_0 = \bar{x}_1 (\bar{A} + \bar{x}_0) = \overline{x_1 + \overline{\bar{A} + \bar{x}_0}}.$$

Formuły te nie zawierają hazardu. Dostosowano je do realizacji z bramek NOR. Na rysunku 6.38 pokazano schemat układu odpowiadający wymienionym formułom. Układ składa się z 9 bramek i 2 przerzutników. Jeśli przerzutniki są złożone z bramek NOR, to łączna liczba bramek wyniesie 13, a wskaźnik kosztu $\langle 28, 13 \rangle$. Wskaźnik kosztu układu pokazanego na rysunku 6.24 wynosi $\langle 27, 11 \rangle$ (bez negatorów zmiennych sterujących), jest więc mniejszy. Jest to sytuacja typowa: synteza formalna z zastosowaniem przerzutników asynchronicznych jest prostsza, ale za to prowadzi do rozwiązań nieoptymalnych.



Rys. 6.38. Schemat układu sterowania pompami z asynchronicznymi przerzutnikami typu RS (przykład 6.7)

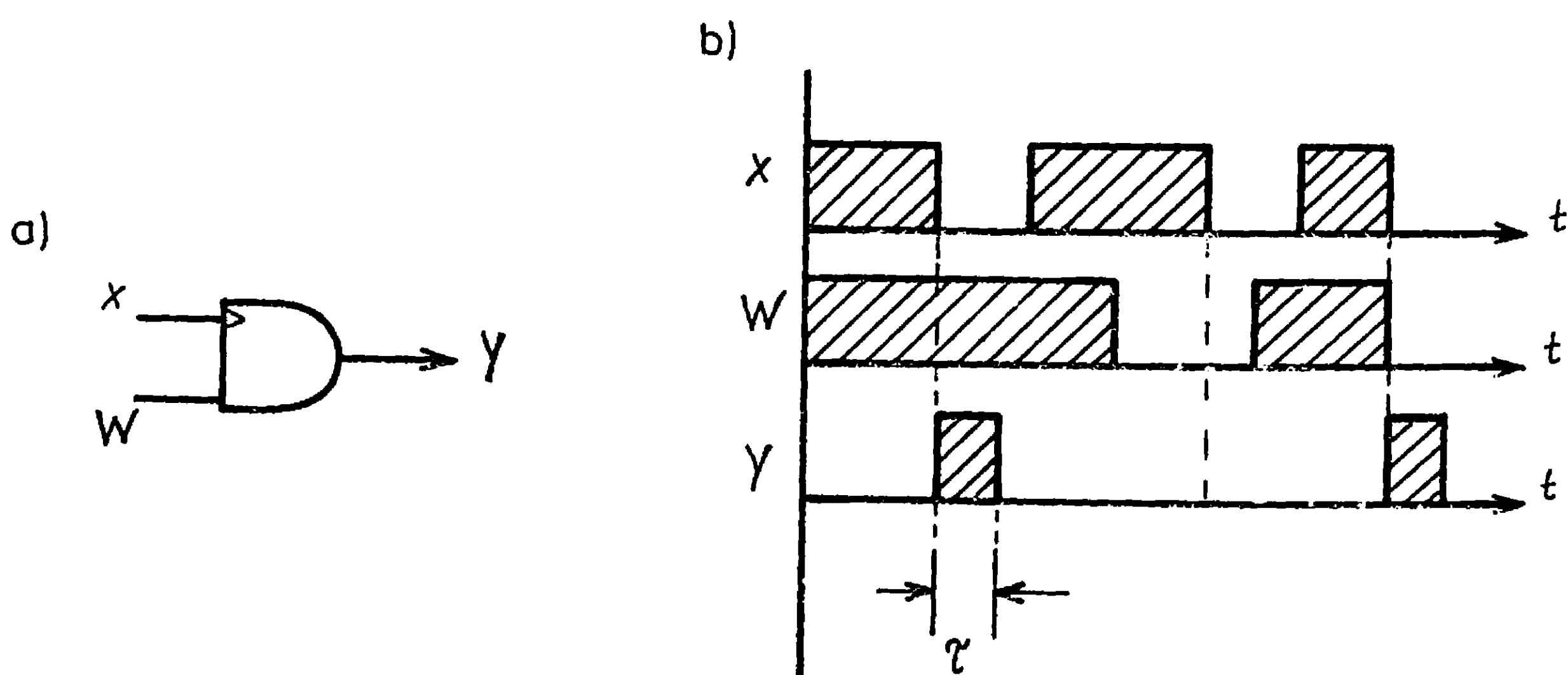
6.4. Asynchroniczne układy impulsowe (dynamiczne)

Ograniczenia związane ze zjawiskiem wyścigu można częściowo wyeliminować, stosując bramki impulsowe. Charakteryzują się one tym, że zmiana sygnału na tzw. wejściu impulsowym z 1 na 0 (na tylnym zbo-

czu impulsu) powoduje wygenerowanie impulsu wyjściowego w postaci "krótkiej jedynki" o czasie trwania wystarczającym do pobudzenia przerzutnika asynchronicznego. Pojawienie się impulsu jest uwarunkowane obecnością jedynki na wejściu potencjałowym. Na rysunku 6.39 pokazano symbol oraz charakterystyki czasowe bramki. Jeśli tylne zbocze sygnału na wejściu impulsowym x i tylne zbocze sygnału na wejściu warunkującym W wystąpi synchronicznie w tej samej chwili czasowej, to na wyjściu pojawi się jeszcze sygnał impulsowy - jeśli sygnał W stanie się zerem wcześniej, zmiana sygnału z 1 na 0 na wejściu impulsowym nie wywoła reakcji na wyjściu. Sygnał impulsowy wygenerowany zmianą wartości zmiennej wejściowej x z 1 na 0, określamy symbolem dx . Formuła wyjściowa bramki impulsowej z warunkiem W będzie więc opisana zależnością

$$y = W dx.$$

Bramkę tę symbolizuje bramka iloczynu z trójkątnym oznaczeniem wejścia impulsowego.



Rys. 6.39. Symbol (a) i charakterystyka czasowa bramki impulsowej (b)

Bramki impulsowe mogą być realizowane w postaci scalonej lub dyskretniej, złożonej z rezystorów i pojemności. Ich wykorzystanie prezentują poniższe przykłady.

Przykład 6.8. Zaprojektować impulsowy układ asynchroniczny, którego pracę opisuje tabela i graf przedstawiony w przykładzie 6.4 (rys. 6.14).

a)

	$x_1 x_0$				
	00	01	11	10	
1	2	①	—	①	01
2	②	3	—	1	10
3	③	③	—	1	11

b)

	$\begin{matrix} x_1^t & x_0^t \\ \hline x_1^{t+1} & x_0^{t+1} \end{matrix}$								
	$\frac{00}{10}$	$\frac{01}{11}$	$\frac{11}{01}$	$\frac{10}{00}$	$\frac{10}{11}$	$\frac{00}{01}$	$\frac{01}{00}$	$\frac{11}{10}$	
1	—	—	—	2	—	—	2	—	01
2	1	—	—	—	—	3	—	—	10
3	1	—	—	—	—	3	3	—	11

Rys. 6.40. Tablica automatu pierwotnego (a) i tablica impulsowa układu z przykładu 6.8 (b)

Z uwagi na to, że zjawisko wyścigu wewnętrznego nie stanowi ograniczenia projektowego, gdyż elementy pamięciowe są wyzwalane impulsami, nie ma potrzeby wprowadzania dodatkowych stanów stabilnych lub niestabilnych. Tabelę pierwotną automatu przekształcamy na tzw. tabelę impulsową, której kolumny stanów następnych są utworzone dla każdej wersji zmian wektora wejściowego. Ponieważ wejście nie może wprowadzać zjawiska wyścigu, więc dopuszczalne są zmiany wartości wektora tylko na jednej pozycji. Dla n -bitowego wektora daje to w konsekwencji $n \cdot 2^n$ kolumn. W omawianym przykładzie $n = 2$, a więc tabela impulsowa będzie zawierać 8 kolumn stanów następnych. W kratkach poszczególnych kolumn są umieszczane tylko stany stabilne, które układ osiągnie startując z określonego stanu, również stabilnego. W pozostałych kratkach wpisuje się kreski, a więc są one nieokreślone. Przykładowo dla stanu aktualnego (1) i litery wejściowej 00, w tabeli impulsowej wpisujemy kreskę, gdyż stan następny 2 wpisany w tabelę przejść automatu pierwotnego, jest stanem niestabilnym, natomiast dla stanu (2), po zmianie litery wejściowej z 00 na 10, automat przechodzi poprzez stan stabilny ② do stanu niestabilnego 1 i następnie poprzez ten stan do stanu stabilnego ①. W tabeli impulsowej w kolumnie odpowiadającej zmianie wektora 00 na 10 (00/10) dla wiersza 2 wpisujemy ten stan jako docelowy. W ten sposób wypełniamy całą tabelę (rys. 6.40). Każdy wiersz tabeli impulsowej jest związany z inną literą wyjściową, nie występują więc tu stany niesprzeczne i tabeli skrócić się dalej nie da. Kodując stany w ten sposób, że numerom opisującym te stany przypisujemy ich odpowiedniki w dziedzinie liczb binarnych, otrzymujemy tabelę pokazaną na rysunku 6.41. Dodatkowo do tabeli dodano wiersz

odpowiadający niewykorzystanej wartości wektora stanów oraz dokonano przedstawienia wierszy, ułatwiającego łączenie krutek sąsiednich. U dołu tabeli podano zapis analityczny zmian wektora wejściowego odpowiadający wartościom opisującym poszczególne kolumny przejść. Zmiany wartości x_1 z 0 na 1 utożsamiają się ze zmianą wartości dopełnienia $\bar{x}_1$ z 1 na 0. Zaznaczono to w tabeli symbolem $d\bar{x}_1$. Identycznie postąpiono ze zmianami x_0 , wprowadzając symbole $d\bar{x}_0$ i dx_0 . Zapis analityczny uzasadnia przyjętą kolejność występowania kolumn w tabeli impulsowej, pokazanej na rysunku 6.40b.

x_1x_0		00	01	11	10	10	00	01	11	y_1y_0
s_1s_0		10	11	01	00	11	01	00	10	
	00	--	--	--	--	--	--	--	--	--
(1)	01	--	--	--	10	--	--	10	--	01
(3)	11	01	--	--	--	--	11	11	--	11
(2)	10	01	--	--	--	--	11	--	--	10
		$x_0=0$	$x_0=1$	$x_0=1$	$x_0=0$	$x_1=1$	$x_1=0$	$x_1=0$	$x_1=1$	
		$d\bar{x}_1$		dx_1		$d\bar{x}_0$		dx_0		

Rys. 6.41. Zakodowana tablica impulsowa układu z przykładu 6.8

Obecnie można przystąpić do opisu analitycznego projektowanego układu. Można tego dokonać w różny sposób. Najbardziej formalna procedura polega na określeniu tabeli wzbudzeń dla każdej wartości wejścia impulsowego. Przyjmując do realizacji asynchroniczne przerzutniki typu RS, otrzymujemy zestaw tabel pokazany na rysunku 6.42. Funkcje r_A , s_A , r_B , s_B reprezentują sobą funkcje warunku dla odpowiedniej bramki impulsowej, której sygnał tworzy alternatywne sterowanie wejść przerzutników. Na wejścia R i S odpowiednich przerzutników są podawane sumy logiczne z bramek impulsowych z funkcjami warunku r_A , s_A , r_B , s_B , określonymi dla $d\bar{x}_1$, dx_1 , $d\bar{x}_0$, dx_0 na podstawie tabeli wzbudzeń przerzutnika RS. Minimalne formuły określone dla wejść przerzutników wyniosą:

$$RA = d\bar{x}_1 \cdot 1,$$

$$SA = dx_1 \cdot 1 + dx_0 \cdot 1,$$

a)

			$d\bar{x}_1$				dx_1							
x_0	s_1^t	s_0^t	s_1^{t+1}	s_0^{t+1}	rA	sA	rB	sB	s_1^{t+1}	s_0^{t+1}	rA	sA	rB	sB
0	0	0	X	X	X	X	X	X	X	X	X	X	X	X
0	0	1	X	X	X	X	X	X	1	0	0	1	1	0
0	1	0	0	1	1	0	0	1	X	X	X	X	X	X
0	1	1	0	1	1	0	0	X	X	X	X	X	X	X
1	0	0	X	X	X	X	X	X	X	X	X	X	X	X
1	0	1	X	X	X	X	X	X	X	X	X	X	X	X
1	1	0	X	X	X	X	X	X	X	X	X	X	X	X
1	1	1	X	X	X	X	X	X	X	X	X	X	X	X

b)

			$d\bar{x}_0$				dx_0							
x_1	s_1^t	s_0^t	s_1^{t+1}	s_0^{t+1}	rA	sA	rB	sB	s_1^{t+1}	s_0^{t+1}	rA	sA	rB	sB
0	0	0	X	X	X	X	X	X	X	X	X	X	X	X
0	0	1	X	X	X	X	X	X	1	0	0	1	1	0
0	1	0	1	1	0	X	0	1	X	X	X	X	X	X
0	1	1	1	1	0	X	0	X	1	1	0	X	0	X
1	0	0	X	X	X	X	X	X	X	X	X	X	X	X
1	0	1	X	X	X	X	X	X	X	X	X	X	X	X
1	1	0	X	X	X	X	X	X	X	X	X	X	X	X
1	1	1	X	X	X	X	X	X	X	X	X	X	X	X

Rys. 6.42. Tablice wzbudzeń otrzymane z tablicy pokazanej na rys. 6.4 (a,b)

$$RB = dx_1 \cdot 1 + dx_0 \cdot \bar{s}_1 \equiv dx_1 + dx_0 \cdot \bar{A},$$

$$SB = d\bar{x}_1 \cdot 1 + d\bar{x}_0 \cdot 1.$$

Powyższa procedura jest stosunkowo złożona i w przypadku gdy występują liczne nieokreśloności lepiej posłużyć się bezpośrednio tabelą pierwotną przejść - wyjść pokazaną na rysunku 6.41. W tym celu wygodnie jest w kolumnach stanów następnych wyróżnić (pogrubzić) te pozycje, dla których następuje zmiana w porównaniu ze stanem aktualnym. Przejściom typu $1 \rightarrow 0$ odpowiadają wartości 1 na wejściu R, natomiast przejściom $0 \rightarrow 1$ - wartości 1 na wejściu ustawiającym S. Przejścia typu $0 \rightarrow 0$, $1 \rightarrow 1$ są neutralne dla stanu wejść $\langle 00 \rangle$. Dodatkowo, dla przejścia $0 \rightarrow 0$ neutralną jest wartość wektora wejściowego $\langle 0X \rangle$ natomiast dla $1 \rightarrow 1$ - wartość $\langle X0 \rangle$. Przykładowo, dla kolumn odpowiadających impulsom $d\bar{x}_1$, zmiana stanu z 1 na 0 występuje dla przerzutnika A dla dwóch stanów wewnętrznych: 11 oraz 10. Dla ośmiu parami sąsiednich kratek, funkcja warunku przyjmie wartość 1. Ponieważ w pozostałych kolumnach nie ma na pierwszej pozycji przejść typu $1 \rightarrow 0$, więc ostatecznie dla wejścia RA można napisać

$$RA = d\bar{x}_1 \cdot 1 = d\bar{x}_1.$$

Jest to rezultat identyczny z otrzymanym poprzednio, jednak uzyskany znacznie szybciej. Podobnie dla wejścia ustawiającego SA, przejścia $0 \rightarrow 1$ występują w kolumnach dx_1 oraz dx_0 . Wykorzystując do minimalizacji stany nieokreślone, można napisać

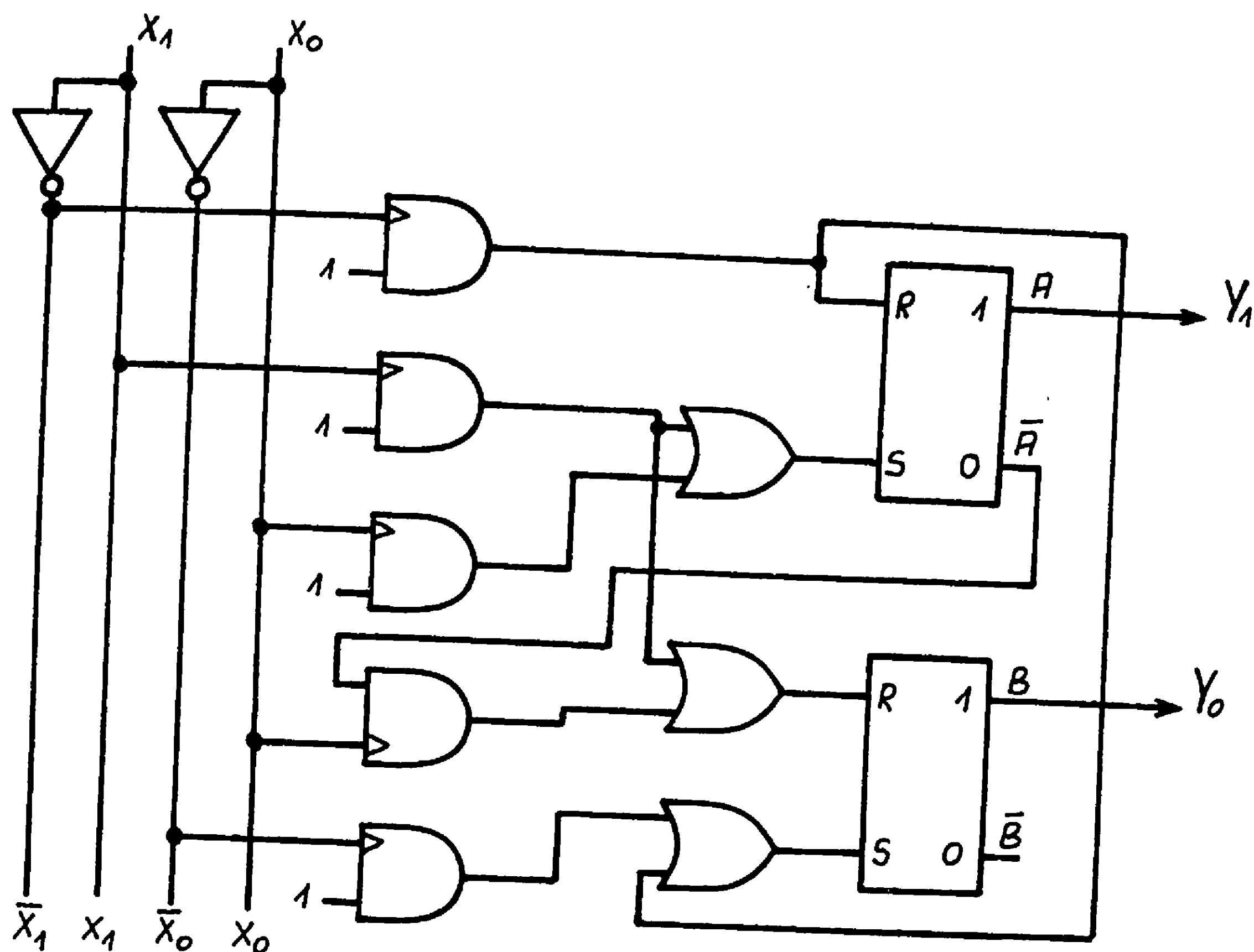
$$SA = dx_1 + dx_0.$$

W analogiczny sposób uzyska się wyrażenie sterujące dla przerzutnika B. I one będą identyczne z otrzymanymi uprzednio w oparciu o tabele wzbudzeń. Symbolom s_1 , s_0 przypisujemy nazwy wyjść przerzutników, przy czym $s_1 \rightarrow A$, $s_0 \rightarrow B$. Wyjścia y_1 i y_0 wyniosą wówczas:

$$y_1 = A,$$

$$y_0 = B.$$

Na rysunku 6.43 przedstawiono schemat logiczny układu.



Rys. 6.43. Schemat układu omówionego w przykładzie 6.8

a)

JK	00 10	01 11	11 01	10 00	10 11	00 01	01 00	11 10	
P_1	-	-	P_4	-	-	-	-	P_2	1
P_2	P_2	-	-	P_2	P_3	P_4	-	-	1
P_3	-	-	P_4	-	-	-	-	P_2	0
P_4	P_2	P_1	-	-	-	P_4	P_4	-	0

b)

JK	00 10	01 11	11 01	10 00	10 11	00 01	01 00	11 10	
$1(P_1, P_2)$	1	-	0	1	0	0	-	1	1
$0(P_3, P_4)$	1	1	0	-	-	0	0	1	0
	$K=0$	$K=1$	$K=1$	$K=0$	$J=1$	$J=0$	$J=0$	$J=1$	
	$d\bar{J}$	dJ	$d\bar{K}$	dK					

Rys. 6.44. Tablice impulsowe asynchronicznego przerzutnika typu JK (przykład 6.9), (a,b)

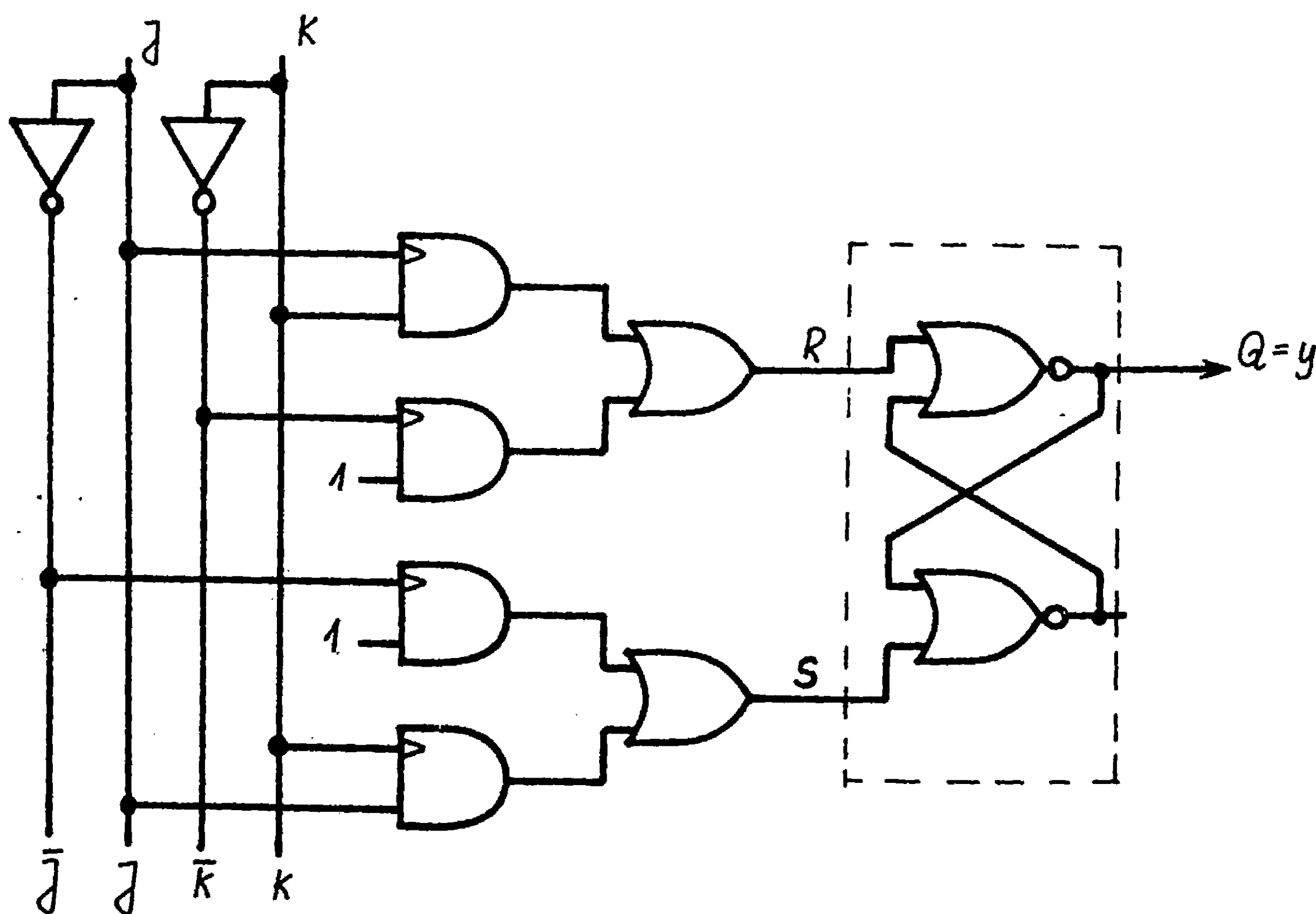
Przykład 6.9. Zaprojektować asynchroniczny, impulsowy przerzutnik typu JK.

Punktem wyjściowym niech będzie tabela pokazana na rysunku 6.30. Odpowiadająca tej tabeli tabela impulsowa została pokazana na rysun-

ku 6.44a. Litery wyjściowe stanów p_1 , p_2 oraz p_3 , p_4 są sobie nawzajem równe, można więc zastanowić się nad tym, czy nie mogą być zastąpione jednym stanem wspólnym. Stany następne w takim przypadku muszą być stanami nieokreślonymi albo muszą być podzbiorem zbioru reprezentowanego przez stan wspólny. Zgodnie z powyższymi uwagami, przekształcono tabelę pierwotną do postaci przedstawionej na rysunku 6.44b, przyjmując jako stan wspólny dla stanów p_1 , p_2 stan oznaczony symbolem 1, natomiast dla stanów p_3 , p_4 - stan oznaczony symbolem 0. Symbole te przyjęto celowo, aby wyjścia przerzutnika były od razu wyjściami układu. Stany następne otrzymane w wyniku przejść $0 \rightarrow 1$ oraz $1 \rightarrow 0$ pogrubiono. Analiza tabeli pozwala na określenie następujących impulsowych równań wejściowych asynchronicznego przerzutnika typu RS:

$$R = dJ K + d\bar{K},$$

$$S = d\bar{J} + dK J.$$



Rys. 6.45. Schemat asynchronicznego, impulsowego przerzutnika typu JK (przykład 6.9)

Schemat układu pokazano na rysunku 6.45. Przerzutnik jest zawsze zerowany na przednim zboczu impulsu K ($d\bar{K}$), natomiast ustawiany na przednim zboczu impulsu J ($d\bar{J}$). Warunkowe zerowanie występuje, gdy sygnał J zmienia stan z 1 na 0 (dJ) i jednocześnie $K = 1$. Warunkowe ustawianie wystąpi gdy sygnał J = 1, a sygnał K zmienia stan z 1 na 0. Mechanizmy te są intuicyjnie wyczuwalne i zgodne z zasadą pracy przerzutnika JK.

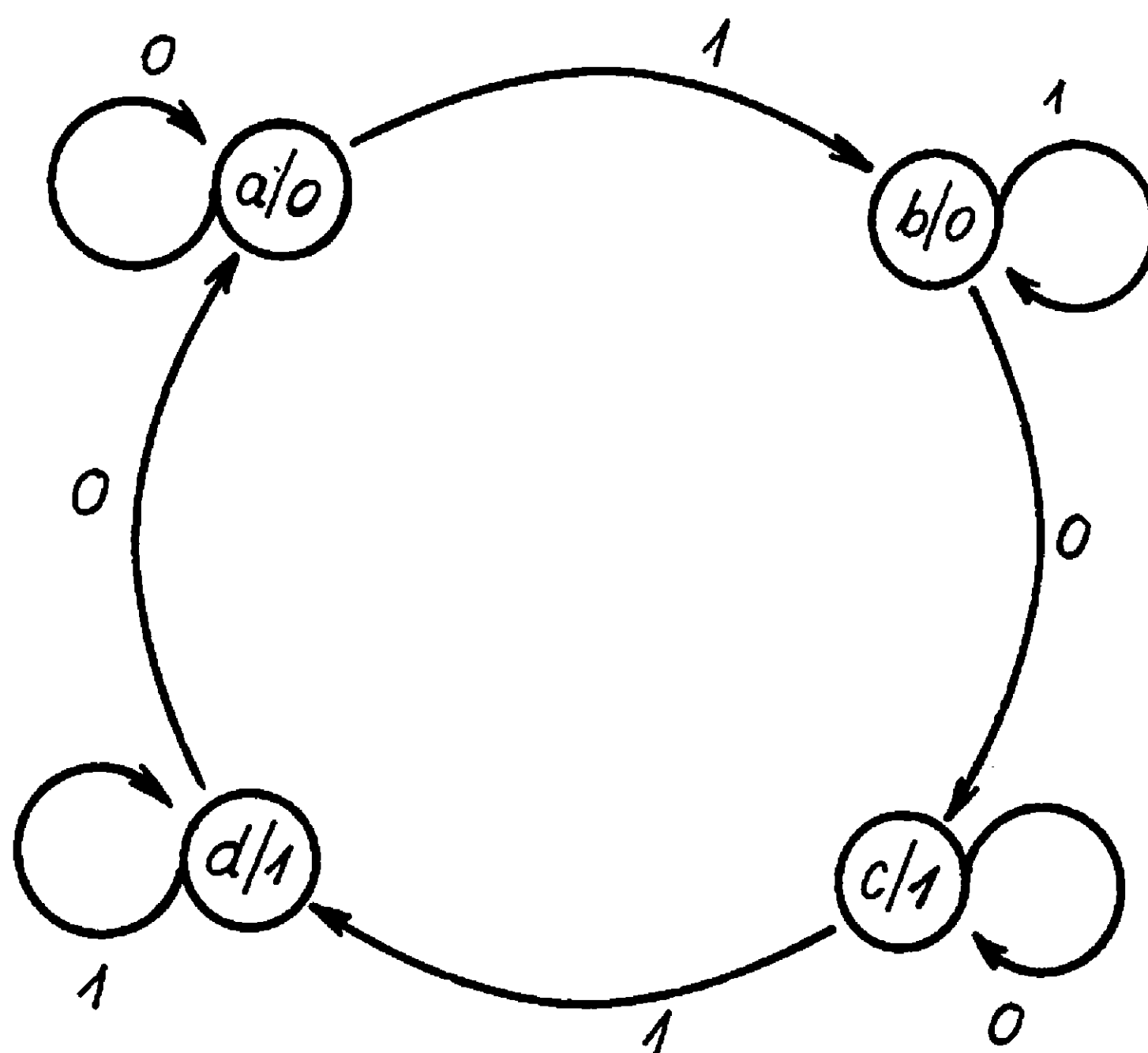
Przykład 6.10. Zaprojektować asynchroniczny impulsowy przerzutnik typu T.

Jako punkt wyjściowy przyjmijmy wariant przerzutnika opisany grafem pokazanym na rysunku 6.46. Grafiemu temu odpowiada tablica przejść - wyjść pokazana na rysunku 6.47a. Na rysunku tym pokazano także tablicę impulsową b), impulsową tablicę zredukowaną c) i jej zakodowany wariant d). Równania wyjściowe użytego do realizacji przerzutnika RS, mają postać

$$R = Q \, dT,$$

$$S = \bar{Q} \, dT.$$

Schemat układu pokazano na rysunku 6.48.



Rys. 6.46. Graf asynchronicznego przerzutnika typu T (przykład 6.10)

a)

	T	0	1
a	@	b	0
b	c	Ⓟ	0
c	Ⓒ	d	1
d	a	Ⓢ	1

b)

	T	$\frac{0}{1}$	$\frac{1}{0}$
a	b	-	0
b	-	c	0
c	d	-	1
d	-	a	1

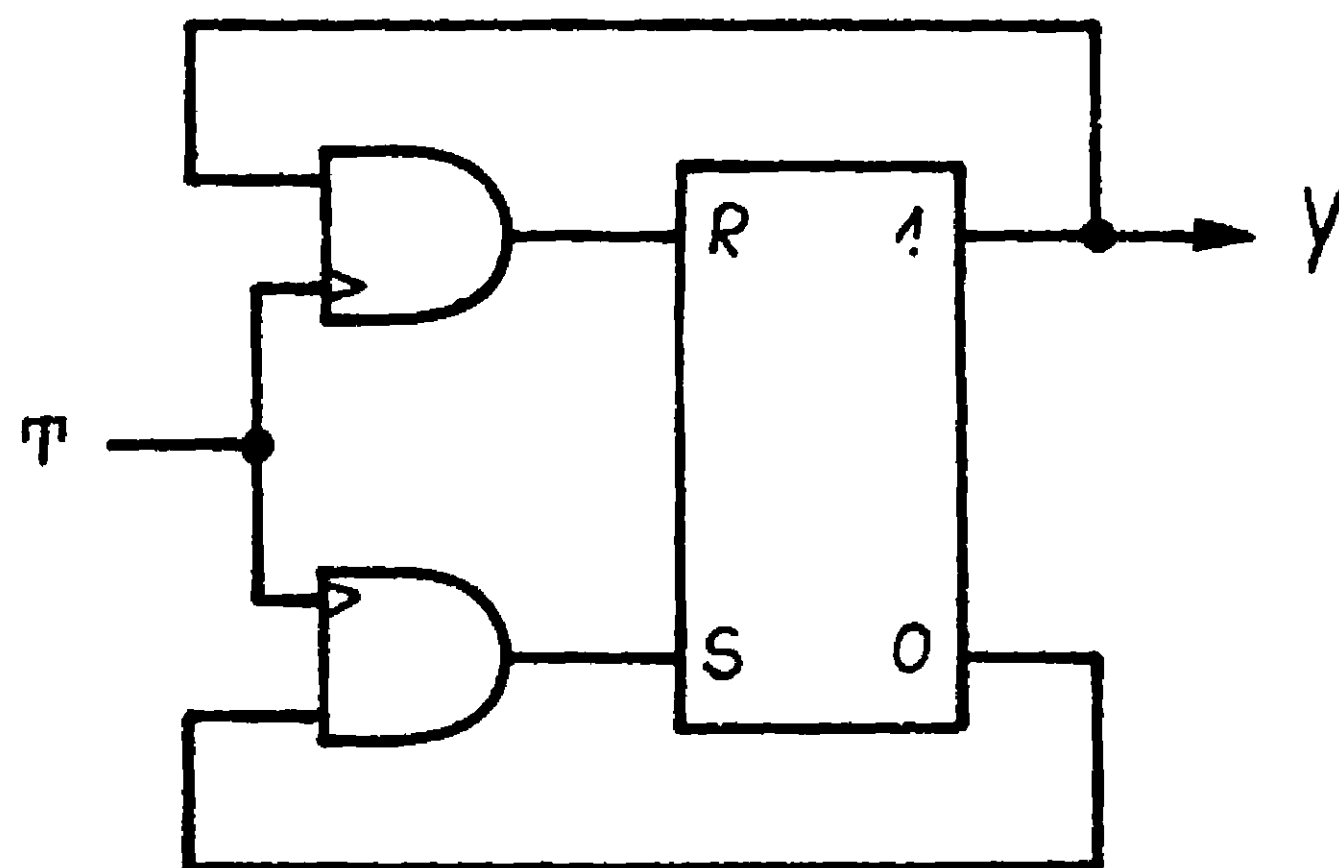
c)

	T	$\frac{0}{1}$	$\frac{1}{0}$
(a,b)	b	c	0
(c,d)	d	a	1

d)

	T	$\frac{0}{1}$	$\frac{1}{0}$
Q			
0	0	1	0
1	1	0	1
	$d\bar{T}$	dT	

Rys. 6.47. Tablica przejść, tablica impulsowa oraz zredukowane (a,b,c,d) tablice impulsowe asynchronicznego przerzutnika typu T (przykład 6.10) (c,d)



Rys. 6.48. Schemat impulsowego asynchronicznego przerzutnika typu T (przykład 6.10)

7. WSTĘP DO SYSTEMÓW CYFROWYCH

7.1. Uwagi ogólne

W 1919 roku dwaj badacze Eccles i Jordan opracowali pierwszy element pamięciowy zbudowany na lampach, nazwany następnie przerzutnikiem w układzie Eccles-Jordana. Współcześni docenili rangę wynalazku, rozpoczęły się intensywne prace nad rozwojem zarówno technologii lampowych układów przełączających, jak i ich zastosowań w przemyśle. Równocześnie podjęto intensywne badania nad teorią algorytmów i systemów cyfrowych.

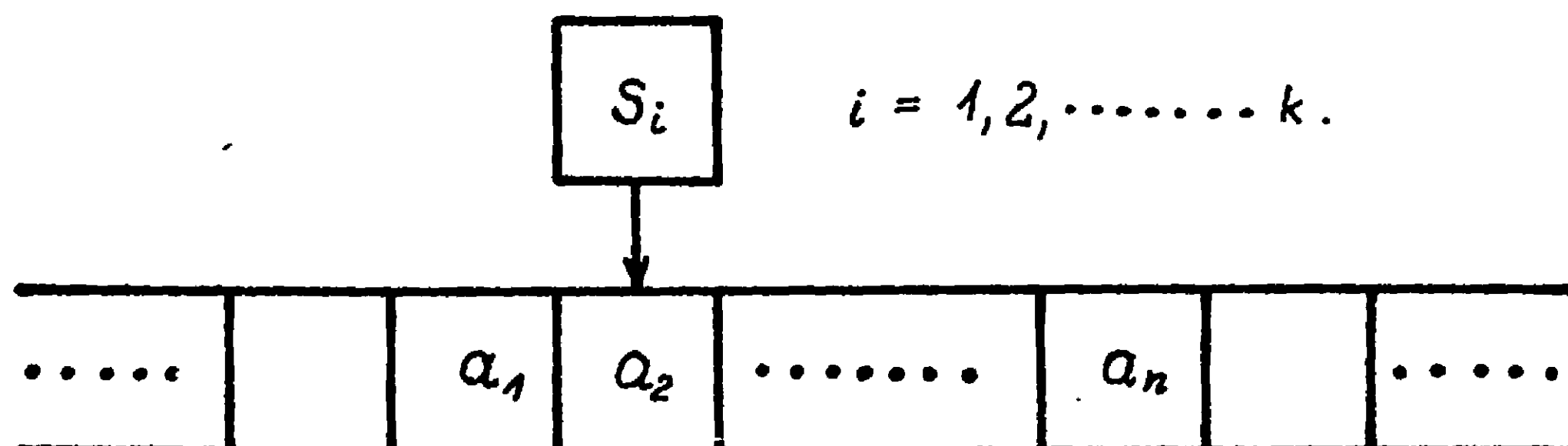
W 1936 roku matematyk angielski A. Turing oraz niezależnie od niego matematyk amerykański E. Post opublikowali prace, w których przedstawiono koncepcje abstrakcyjnych maszyn matematycznych realizujących określony schemat postępowania algorytmicznego przetwarzającego zadane zbiory symboli (alfabety). Symbole te ułożone w tzw. słowa wejściowe są przetwarzane wg określonego przepisu na słowa wyjściowe złożone z symboli należących do tego samego alfabetu.

Koncepcje Turinga i Posta odegrały istotną rolę w ukierunkowaniu prac nad konstrukcją pierwszej programowalnej maszyny cyfrowej ENIAC zbudowanej w 1946 roku na Uniwersytecie Stanowym w Pensylwanii (USA). Stały się one inspiracją do rozwijania dalszych prac badawczych w dziedzinie maszyn abstrakcyjnych.

7.2. Maszyna Turinga

Maszyna Turinga składa się z nieskończonej taśmy podzielonej na kratki (komórki) oraz z urządzenia sterującego, mającego znajdować się w jednym ze stanów: $s_1, \dots, s_k$ (rys. 7.1). Maszyna wykonuje ruch o jedną kratkę w lewo lub w prawo, odczytując symbol zapisany na taśmie i dokonując ewentualnej jego zmiany. W szczególnym przypadku symbol na taśmie może być wymazany. Oznacza to, że alfabet symboli zapisanych na taśmie zawiera element pusty, oznaczany często przez ϕ . Parę: aktual-

nie obserwowany symbol, stan urządzenia sterującego nazywamy sytuacją. Przykładowo na rysunku 7.1 dla $i = 1$ maszyna jest w sytuacji (a_2, s_1) . Wśród k stanów urządzenia sterującego znajduje się jeden stan bierny, powodujący zatrzymanie maszyny. Pozostałe stany powodują ruch, a to z kolei wywołuje:



Rys. 7.1. Schemat ideowy maszyny Turinga

- a) zmianę stanu urządzenia sterującego,
- b) odczytanie aktualnie obserwowanego symbolu i wpisanie na jego miejsce nowego; maszyna może także wpisać na taśmę ten sam symbol bądź też symbol obserwowany wymazać (pozostawić symbol lub wpisać symbol pusty $\emptyset$),
- c) zmianę obserwowanej kratki na kratkę znajdującą się na lewo (L) lub na prawo (P) od kratki obserwowanej, ewentualnie zatrzymanie maszyny dla stanu biernego (N).

Na skutek ruchu maszyna znajdzie się w nowej sytuacji (a_j, s_i) . Aby określić maszynę należy podać informacje:

- o stanie następnym urządzenia sterującego,
- o literze, która ma zostać wpisana w miejsce aktualnie obserwowanej kratki,
- o kierunku przejścia do następnej sąsiedniej kratki.

Informacje te mogą być podane w formie grafu lub tabeli. Na rysunku 7.2a przedstawiono przykładowo tabelę opisującą maszynę Turinga, której alfabet wejściowy zawiera cztery elementy

$$\{\emptyset, a, b, c\}.$$

Symbol $\emptyset$ oznacza element pusty. Stany urządzenia sterującego należą do zbioru $\{1, 2, 3\}$, przy czym stan 3 jest stanem biernym (stanem stopu). Na rysunku 7.2b pokazano kolejne etapy przetwarzania przez maszynę zapisanego na taśmie słowa a b c. Maszyna startuje od lewej pu-

stej kratki sąsiadującej z kratką, w której zapisano literę a, przy czym urządzenie sterujące znajduje się na początku w stanie 1. Aktualną sytuację maszyny opisuje para $(\phi, 1)$ lub w skrócie ϕ_1 . Indeks przy symbolu aktualnie obserwowanym określa stan urządzenia sterującego. W kolejnych sekwencjach symbole puste umieszczono tylko wtedy, gdy tworzą one aktualną sytuację. Zakładamy przy tym, że poza kratkami zawierającymi słowo a b c, taśma zawiera tylko symbole puste.

a)				b)		
	1	2	3		a b c	b a b ₁ a
ϕ	2 b,P	3	3	ϕ_1	a b c	b a ₁ b a
a	2 a,L	2 a,P	3	b	a ₂ b c	b ₂ a b a
b	1 b,L	2 b,P	3	b	a b ₂ c	ϕ_3 b a b a
c	3	1 a,L	3	b	a b c ₂	b a b a

Rys. 7.2. Tablica maszyny Turinga (a) oraz sekwencje zmian słowa początkowego wywołane pracą maszyny (b)

Zgodnie z tabelą stanem następnym sytuacji ϕ_1 jest stan urządzenia sterującegoznaczony numerem 2. Jednocześnie w aktualnie obserwowaną kratkę zostanie wpisany symbol b i maszyna przejdzie do obserwowania następnej prawej kratki. W kratce tej znajduje się symbol a, więc nową sytuację maszyny można symbolicznie zapisać jako a_2 . Słowo w tym cyklu będzie mieć postać b a₂ b c. Kolejne cykle są pokazane na rysunku 7.2b. Stan 3 jest stanem biernym - maszyna zatrzymuje się, a końcowe słowo ma postać b a b a.

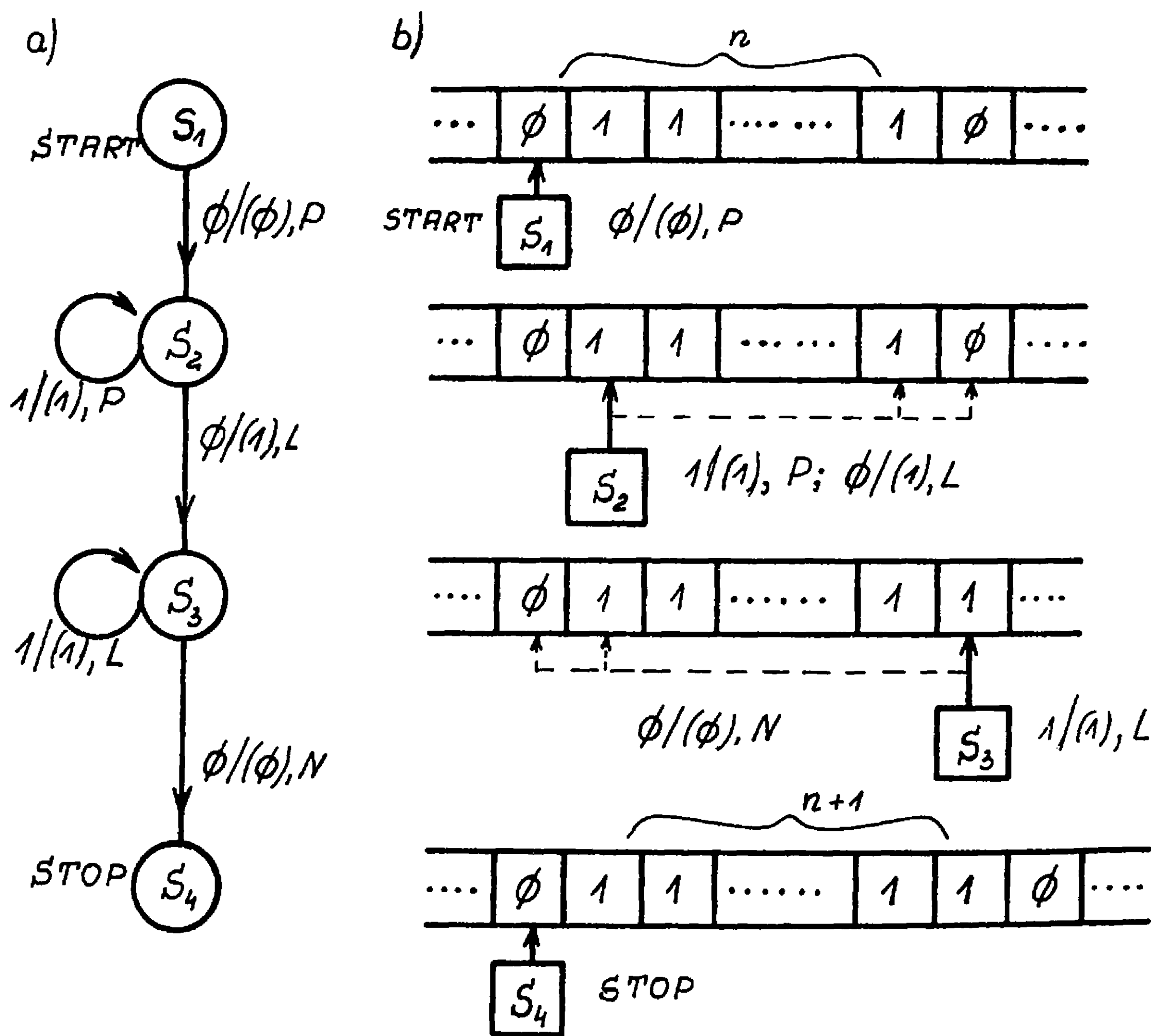
Urządzenie sterujące maszyny Turinga może być określone jako automat posiadający dwa wyjścia: jedno służy do wpisywania symboli na taśmę, drugie decyduje o ruchu względem taśmy. Stan każdego z wyjść zależy od aktualnie obserwowanego symbolu oraz od stanu aktualnego urządzenia sterującego. Jest to więc automat typu Mealy'ego. Wyjście zapisujące nazywa się często głowicą automatu.

Obok przetwarzania słów, maszyna Turinga może także dokonywać obliczeń. Ilustruje to przykład 7.1.

Przykład 7.1. Skonstruować maszynę Turinga liczącą funkcję

$$f(n) = n + 1,$$

przy czym n należy do zbioru liczb naturalnych. Załóżmy, że liczba n jest reprezentowana przez n -jedynek zapisanych kolejno na taśmie, a głowica przed rozpoczęciem obliczeń znajduje się z lewej strony nad pustą kratką sąsiadującą z pierwszą jedynką danych. Ponadto zakładamy, że głowica po dokonaniu obliczeń powróci do pozycji wyjściowej i tam się zatrzyma.



Rys. 7.3. Graf maszyny Turinga z przykładu 7.1(a) oraz sekwencje zmian zapisu na taśmie (b)

Do realizacji zadania wystarczy przyjąć alfabet maszyny zawierający tylko dwa elementy: $\{\emptyset, 1\}$. Na rysunku 7.3a przedstawiono graf maszyny. W kółka wpisano stan głowicy, a strzałki oznaczają kolejne przejścia. Obok strzałek przed kreską pionową podana jest informacja o aktualnie czytany symbolu, za kreską w nawiasie podano symbol litery, która ma być wpisana w miejsce aktualnie obserwowanej oraz informację dla drugiego wyjścia głowicy o tym, która kratka ma być w kolejnym ruchu obserwowana. Startujemy od stanu s_1 (rys. 7.3a), zgodnie z założeniem głowica w tym stanie obserwuje kratkę z pustym symbolem sąsiadującym z pierwszą jedyneką liczby n . Następnie w tę kratkę jest wpisany również symbol pusty, a maszyna wykonuje ruch przechodząc do obserwowania sąsiedniej kratki znajdującej się po prawej stronie i zawierającej symbol 1. Głowica znajdzie się wówczas w stanie s_2 . Stan ten powoduje kolejne przejście głowicy bez zmiany zawartości kratek taśmy aż do momentu natrafienia na symbol pusty. Na grafie obrazuje to pętla własna stanu s_2 . W momencie natrafienia na symbol pusty zostaje w miejsce tego symbolu wpisany symbol 1, a głowica wykonując ruch w lewo przechodzi do stanu s_3 . Stan s_3 powoduje kolejne przejścia głowicy w lewo bez zmiany zawartości kratek (pętla własna stanu s_3), aż do momentu natrafienia na pierwszy symbol pusty. W tym momencie maszyna zatrzymuje się, a głowica przechodzi do stanu stopu oznaczonego s_4 . W wyniku, na taśmie zostaje zapisana w postaci jedynekowej liczba $n + 1$. Sekwencje pracy maszyny obrazowo pokazano na rysunku 7.3b.

7.3. Maszyna Posta

Maszyna Posta podobnie jak maszyna Turinga składa się z nieskończonej długiej taśmy podzielonej na komórki, do których można wpisywać symbole należące do alfabetu danej maszyny. Alfabet zawiera również element pusty. Urządzenie sterujące może wykonywać następujące rozkazy:

- P - przejdź do obserwowania sąsiedniej kratki znajdującej się z prawej strony,
- L - jw. - z lewej strony,
- N - nie zmieniaj położenia głowicy,
- (x_i/r) - przejdź do rozkazu o numerze r , jeśli aktualnie odczytany symbol jest symbolem x_i ; jeśli nie - przejdź do następnego rozkazu; rozkaz ten jest rozkazem skoku warunkowego,

(x_i) - wpisz na taśmę w miejsce aktualnie obserwowanego symbolu, symbol x_i ,

S - zatrzymaj maszynę (stop).

Rozkazy powyższe występują w formie ponumerowanej, tworząc program działania maszyny. Maszyna jest konkretnie określona, jeśli zostanie podany program jej pracy. Łatwo zauważyć, że urządzenie sterujące jest automatem typu Moore'a.

Przykład 7.2. Zrealizować zadanie omówione w przykładzie 7.1, stosując maszynę Posta.

Program liczący funkcję $f(n) = n + 1$ będzie miał postać następującą:

0	P
1	(1/0)
2	(1)
3	L
4	(1/3)
5	S

Początek programu został oznaczony numerem 0, głowica maszyny znajduje się wówczas nad pierwszą pustą kratką sąsiadującą lewostronnie z ciągiem jedynek reprezentujących liczbę n . Rozkaz o numerze 0 powoduje przesunięcie głowicy w prawo o jedną kratkę i przejście do kolejnego rozkazu oznaczonego numerem 1. Jest to rozkaz skoku warunkowego, tzn. jeśli maszyna odczyta symbol 1 to przechodzi do wykonywania rozkazu oznaczonego numerem 0, a więc wystąpi kolejne (n -krotne) przesunięcie głowicy w prawo aż do momentu, gdy symbolem odczytanym będzie symbol pusty $\emptyset$. W tym momencie maszyna przejdzie do wykonywania rozkazu oznaczonego numerem 2, a więc wpisze w pustą kratkę symbol 1. Kolejny rozkaz (nr 3) spowoduje przesunięcie głowicy w lewo o jedną kratkę i przejście do rozkazu nr 4, którym jest następny rozkaz skoku warunkowego, przesuwający głowicę w lewo aż do momentu odczytania pierwszego z lewej strony symbolu pustego. Maszyna przechodzi wówczas do rozkazu nr 5, którym jest rozkaz stop, kończący realizację zadania.

Można wykazać, że każdy algorytm dający się zrealizować na maszynie Posta daje się również zrealizować na maszynie Turinga i odwrotnie. Maszyny te są więc sobie wzajemnie równoważne. Pewne problemy

wygodniej jest rozwiązywać wprowadzając pojęcie maszyny Turinga, inne natomiast operując maszyną Posta. Istnieje hipoteza, że każdy algorytm da się zrealizować za pomocą jednej z omówionych maszyn, jest to jednak tylko hipoteza, gdyż pojęcie algorytmu nie zostało dotąd jasno zdefiniowane.

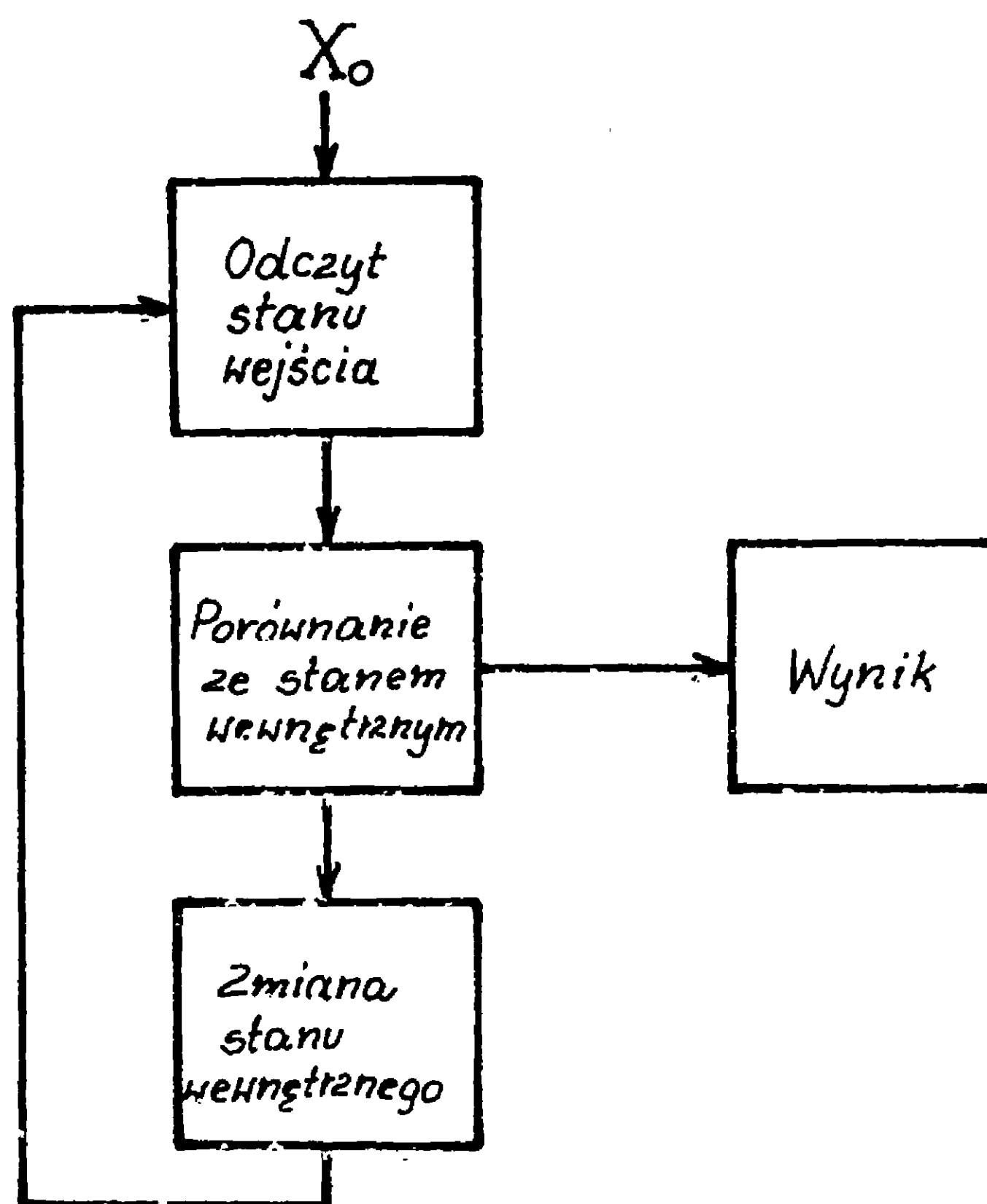
7.4. Maszyny uniwersalne

Koncepcje Turinga i Posta można rozwinąć i uogólnić na maszyny zawierające wiele taśm, co upraszcza program i zmniejsza liczbę ruchów potrzebnych do realizacji algorytmu. Zagadnienia te stały się przedmiotem badań teoretyków, natomiast działalność techniczną skoncentrowano na wykorzystaniu specyficznych własności elementów elektronowych, a w szczególności ich relatywnie bardzo dużej szybkości przełączania. Jest ona o wiele rzędów większa od szybkości przełączania układów mechanicznych czy też elektromechanicznych. W latach trzydziestych parametry techniczne lamp elektronowych osiągnęły już poziom rokujący powodzenie w realizacji systemu liczącego, użytecznego w zastosowaniach. Działania konstrukcyjne i badawcze były jednak niezwykle kosztowne i dopiero komasacja środków technicznych i finansowych wywołanych sytuacją polityczną na świecie (II Wojna Światowa) doprowadziła do sukcesu: dwaj badacze Eckert i Muchly skonstruowali na Uniwersytecie w Pensylwanii (USA) pierwszą elektronową maszynę cyfrową. Była to wykonana w 1946 r. maszyna ENIAC. Do jej konstrukcji użyto około 18000 lamp, przy czym 11000 z nich tworzyło pamięć operacyjną o pojemności 20 liczb dziesiętnych 10-cyfrowych. Maszyna pobierała moc 150 kW i była przeznaczona do rozwiązywania równań różniczkowych stosowanych w balistyce. Obecnie tego typu funkcje wykonują programowalne kalkulatory kieszonkowe, jednak wówczas w świecie techniki powstała nowa jakość: pierwsza generacja komputerów.

Działania wojenne niewątpliwie przyspieszyły tempo prac konstrukcyjnych związanych z budową elektronicznej maszyny liczącej, jednak przyczyny stymulujące prace nad konstrukcją były głębsze. Aby to uzmysłowić prześledźmy podstawowy cykl pracy komputera sekwencyjnego. Przebiega on wg następujących etapów (rys. 7.4):

- 1) odczytanie stanu wejścia X_0 ,
- 2) porównanie stanu wejścia ze stanem wewnętrznym,

- 3) wygenerowanie rezultatu porównania,
- 4) zmiana stanu wewnętrznego,
- 5) odczytanie następnego stanu wejścia.



Rys. 7.4. Podstawowy cykl pracy sekwencyjnej maszyny cyfrowej

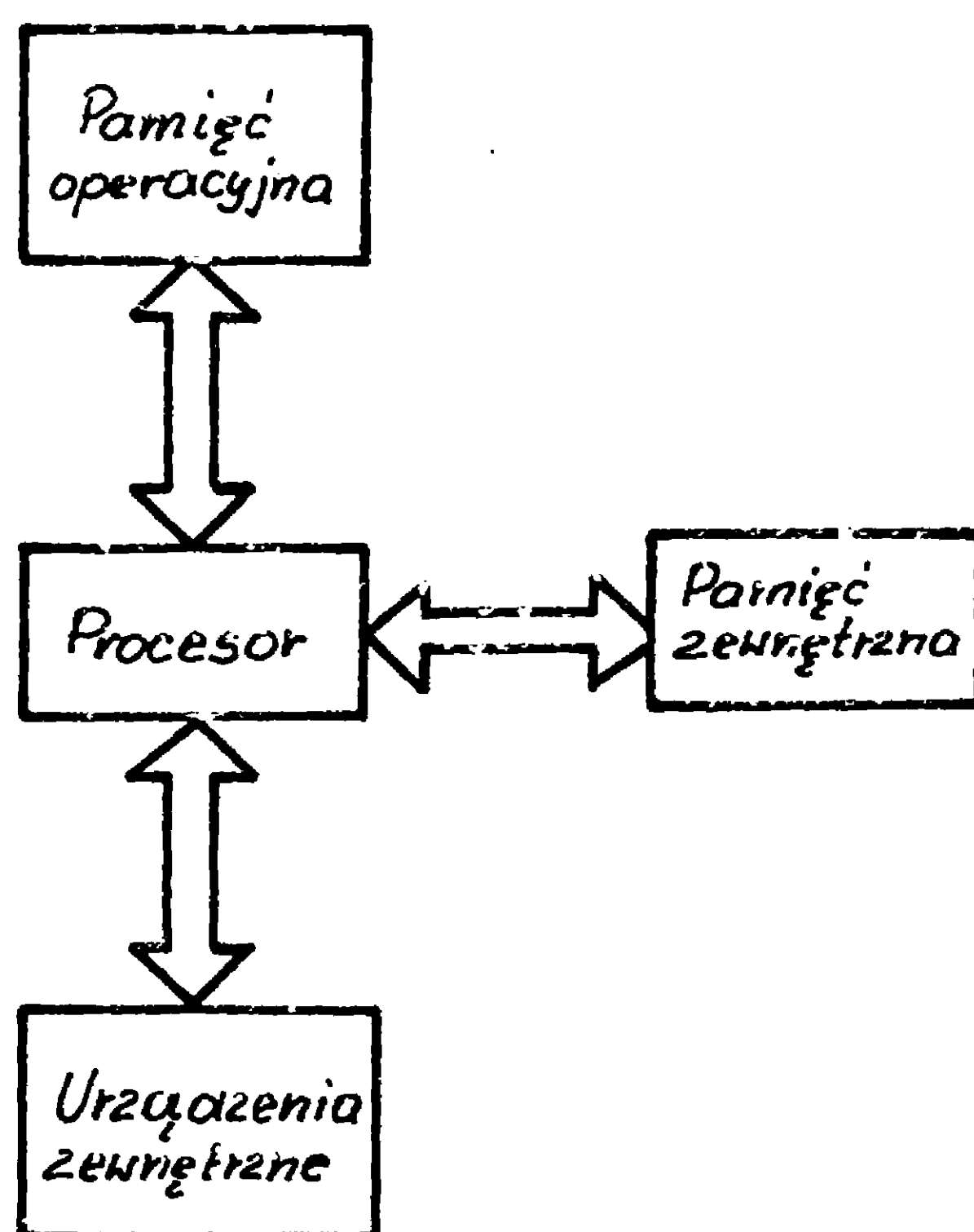
Schemat ten odpowiada hipotetycznym procesom fizycznym, w których zmniejszana jest tzw. entropia, współczynnik będący ogólnie miarą nieuporządkowania danego środowiska. Według tego schematu działa XIX-wieczny "Demon Maxwella", hipotetyczne urządzenie mogące przekazywać energię cieplną ze środowiska o temperaturze niższej do środowiska o temperaturze wyższej - wbrew II zasadzie termodynamiki. Schemat pracy urządzenia obejmuje dwa etapy:

- 1) odczytanie prędkości zbliżającej się cząstki,
- 2) porównanie tej prędkości ze średnią prędkością cząstek znajdujących się w danym środowisku,
- 3) podjęcie decyzji o zatrzymaniu cząstki, jeśli jej prędkość jest mniejsza lub równa prędkości średniej, w przeciwnym przypadku - wypuszczenia jej na zewnątrz,

4) ewentualna zmiana średniej prędkości środowiska, jeśli cząstka została usunięta,

5) odczytanie prędkości kolejno zbliżającej się cząstki itd.

Etapy te są analogiczne do wymienionych uprzednio etapów charakteryzujących podstawowy cykl pracy komputera. Tak więc odpowiednie zastosowanie komputera umożliwia zwiększenie stopnia uporządkowania (zmniejszenia entropii) systemu, w którym on funkcjonuje i właśnie ten fakt odegrał decydującą rolę w procesie rozwoju techniki cyfrowej.



Rys. 7.5. Blokowy schemat funkcjonalny komputera

Podstawowy cykl pracy maszyny cyfrowej może być realizowany przez system, którego schemat blokowy pokazano na rysunku 7.5. W skład systemu wchodzi: wewnętrzna pamięć operacyjna służąca do przechowywania danych i bieżącego programu, procesor, w którym następuje przetwarzanie i zapadają decyzje o przesyłaniu danych do innych bloków, urządzenia zewnętrzne służące do komunikowania się z maszyną, pamięć zewnętrzna służąca do przechowywania wyników i programów. Schemat ten jest określany jako tzw. struktura von Neumana. Według niej działają wszystkie nowoczesne komputery. Postęp techniczny wprowadził ogromne zmiany, ale tylko ilościowe - istota schematu pozostała bez zmian.

W początkowym okresie dominowała struktura pamięci operacyjnej typu Harvard - podziałem na pamięć dla danych i pamięć dla programu. Bardziej popularną okazała się jednak struktura typu Princeton, w której pamięć dla danych i rozkazów jest wspólna. Obecnie powszechnie stosuje się tę ostatnią.

Wynalazek tranzystora zastał systemową technikę cyfrową na poziomie maszyny ENIAC. Naturalną tego konsekwencją było zbudowanie analogicznych maszyn wykorzystujących w miejsce lamp tranzystory. Komputery te określono jako tzw. komputery drugiej generacji. Były one zdecydowanie bardziej niezawodne od poprzednich. Równolegle trwały intensywne prace nad rozwojem technologii półprzewodników. Chodziło z jednej strony o polepszenie parametrów wykonywanych elementów, z drugiej - o realizację w ramach tej samej struktury gotowych układów. Możliwość realizacji w tym samym krysztale półprzewodnika wielu elementów połączonych w układ, była dyskutowana już w początkach lat pięćdziesiątych. Rozpoczął się swoisty wyścig firm amerykańskich, a w wyniku kolejne sukcesy: w 1958 roku firma Texas Instruments opracowała pierwszy seryjny produkowany układ scalony - generator analogowy; w 1961 roku firma Fairchild Semiconductor wprowadziła na rynki światowe scalony przerzutnik; w 1963 roku firma RCA - pierwszy cyfrowy układ scalony typu MOS.

Układy scalone małej i średniej skali integracji dokonały dużego skoku ilościowego i jakościowego w dziedzinie komputerów. Zbudowano maszyny cyfrowe trzeciej generacji, m.in. popularny PDP-11 firmy Digital Equipment Corporation. Dalszy wzrost stopnia scalania, a w szczególności wprowadzenie na rynki systemów mikroprocesorowych doprowadziły do masowych zastosowań sprzętu o własnościach i cenach przekraczających wszelkie optymistyczne prognozy z końca lat sześćdziesiątych. Zażłamał się tradycyjny podział na generacje związany z techniką realizacyjną. Bardziej uzasadniony stał się podział zależny od architektury i możliwości programowych komputerów. W tym sensie mówi się o czwartej i piątej ich generacji. Dalszy postęp zależy od rozwoju technologii półprzewodników i materiałów przechowujących informację. Zagadnienia te stanowią przedmiot badań specjalistów technologów i informatyków, w związku z tym są tu jedynie sygnalizowane.



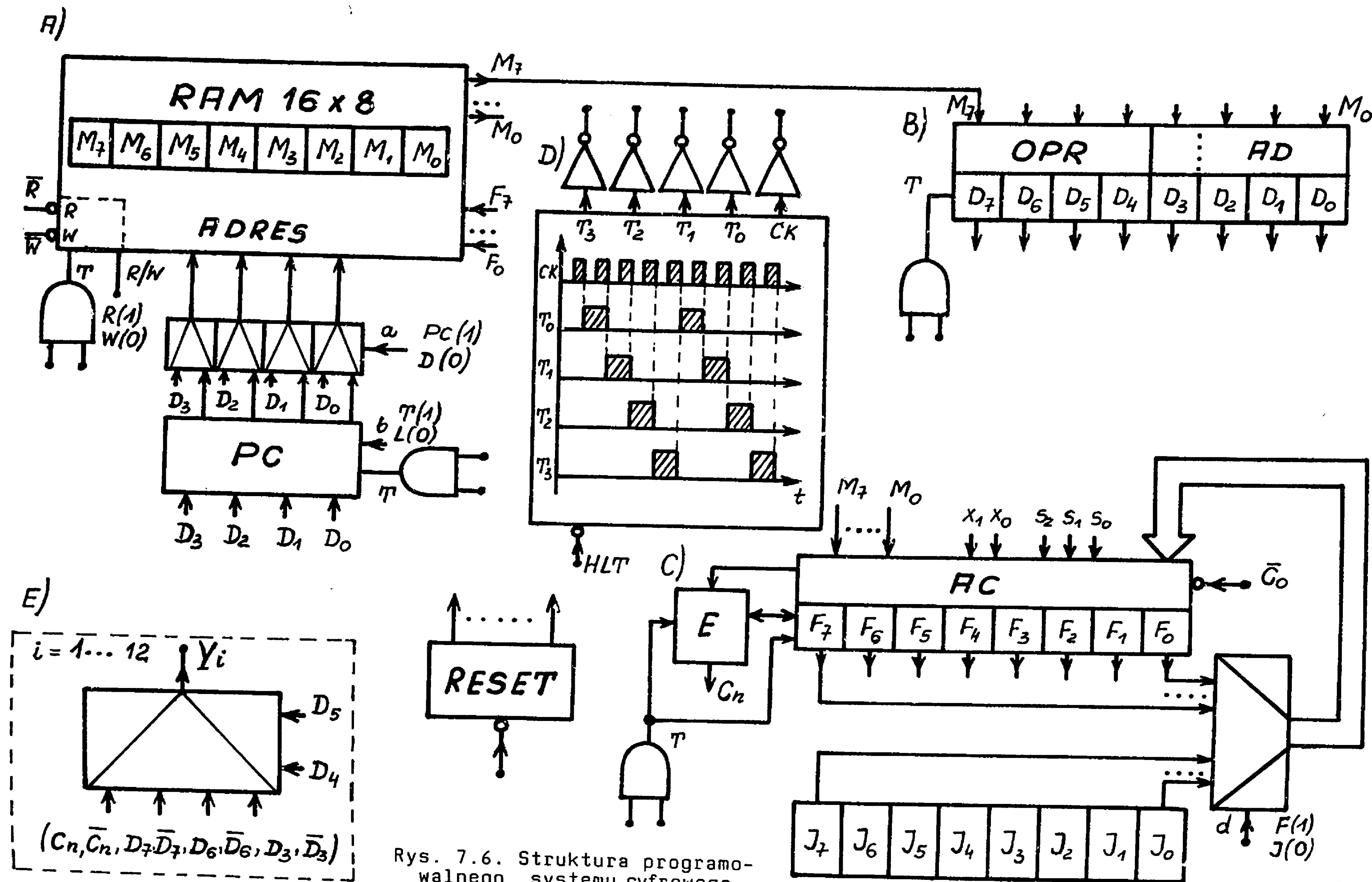
7.5. Laboratoryjny model programowalnego systemu cyfrowego

7.5.1. Struktura systemu

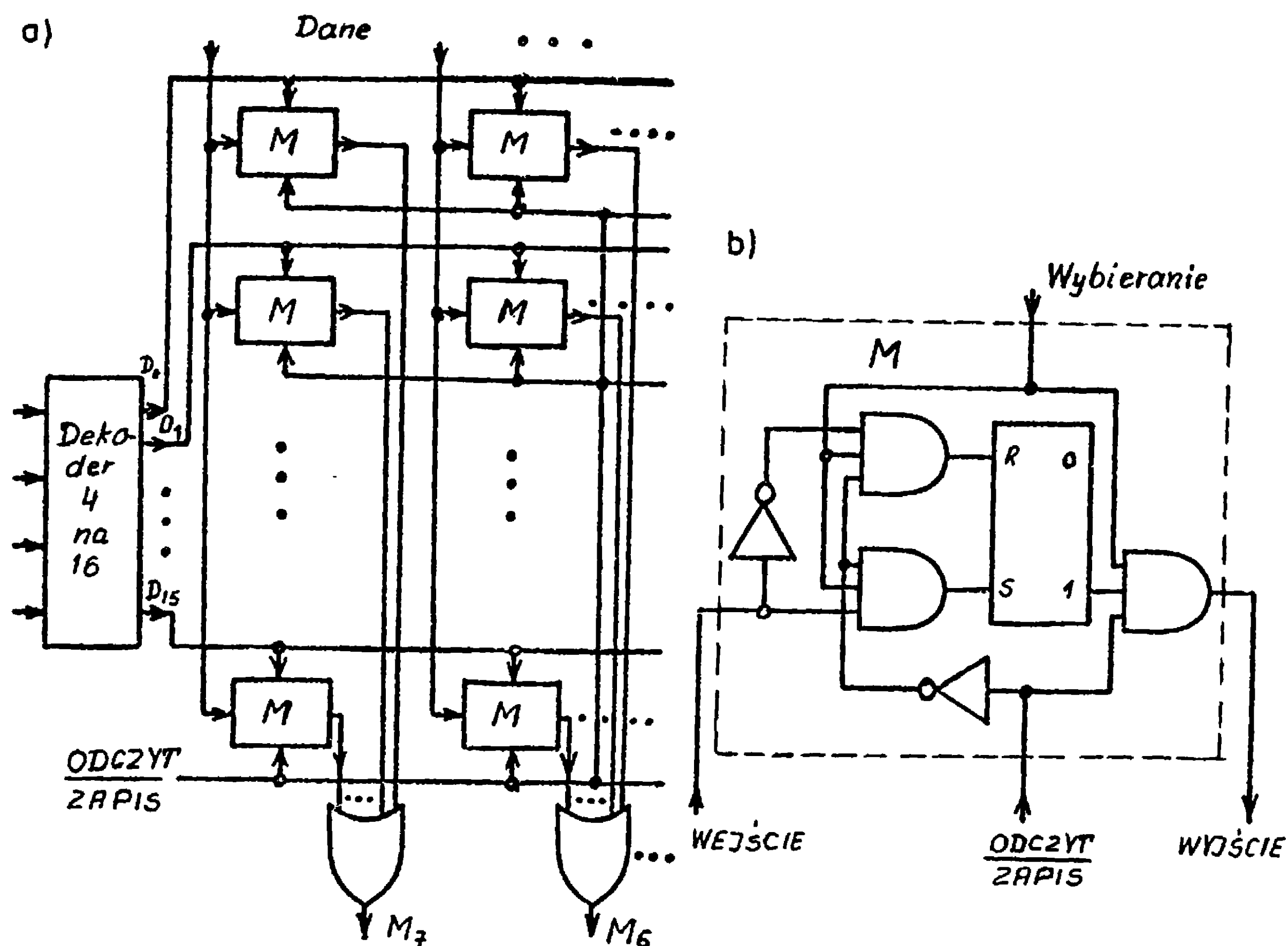
Modyfikacje maszyn Turinga i Posta, dokonane m.in. przez Rabina i Scotta, doprowadzają do pojęcia automatu skończonego, zdefiniowanego w rozdziale 4. Z kolei szczególnym przypadkiem automatów są układy logiczne sekwencyjne omówione w rozdziale 5 i 6, które w związku z tym mogą być uważane za przykłady elementarnych specjalizowanych maszyn pracujących wg algorytmu zadanego tabelą przejść - wyjść lub grafem. Możliwości programowe tych maszyn są niewielkie, gdyż jedynym oddziaływaniem zewnętrznym jakie można zrealizować jest wymuszenie stanów startowych.

Dla pełniejszego zrozumienia możliwości operacyjnych programowalnych systemów sekwencyjnych, należy rozpatrzeć bardziej złożony zestaw bloków logicznych, z których każdy reprezentuje sobą pewien autonomiczny sekwencyjny lub kombinacyjny układ logiczny. System taki został wykonany w Instytucie Elektroniki Politechniki Łódzkiej jako problemowe ćwiczenie laboratoryjne. Schemat systemu pokazano na rysunku 7.6; zawiera on omówione niżej zespoły.

A. Pamięć o dostępie swobodnym RAM (Random-Access Memory). Została utworzona z 16 rejestrów 8-bitowych, wybieranych czterobitowym adresem podawanym na wejście dekodera znajdującego się w bloku pamięci. Na rysunku 7.7a przedstawiono schemat bloku pamięci. Zastosowane w rejestrach przerzutniki są asynchronicznymi przerzutnikami typu RS z sygnałami podanymi na wejścia, uwarunkowanymi koniunkcją wybierania z dekodera z sygnałem (odczyt/zapis) (rys. 7.7b). Tak samo uwarunkowany jest sygnał wyjściowy przerzutnika. W konsekwencji jeśli sygnał na wejściu (odczyt/zapis) wynosi 0 oraz sygnał z dekodera wynosi 1, to do przerzutnika jest wpisany aktualny stan wejścia. Jeśli natomiast sygnał (odczyt/zapis) wynosi 1, to wejście jest zablokowane i następuje odczyt stanu przerzutnika poprzez bramkę iloczynu i jej wyjście. Te elementarne komórki połączone w blok tworzą pamięć o dostępie swobodnym. Na rysunku 7.7a oznaczono je symbolem M. Zależnie od kombinacji bitów reprezentującej adres i stan wejścia (odczyt/zapis) (lub R/W), do odpowiedniego rejestru pamięci są wpisywane dane z wejść $F_7 \dots F_0$ (rys. 7.6) lub też następuje odczyt na wyjściach bramek sumy $M_7 \dots M_0$.



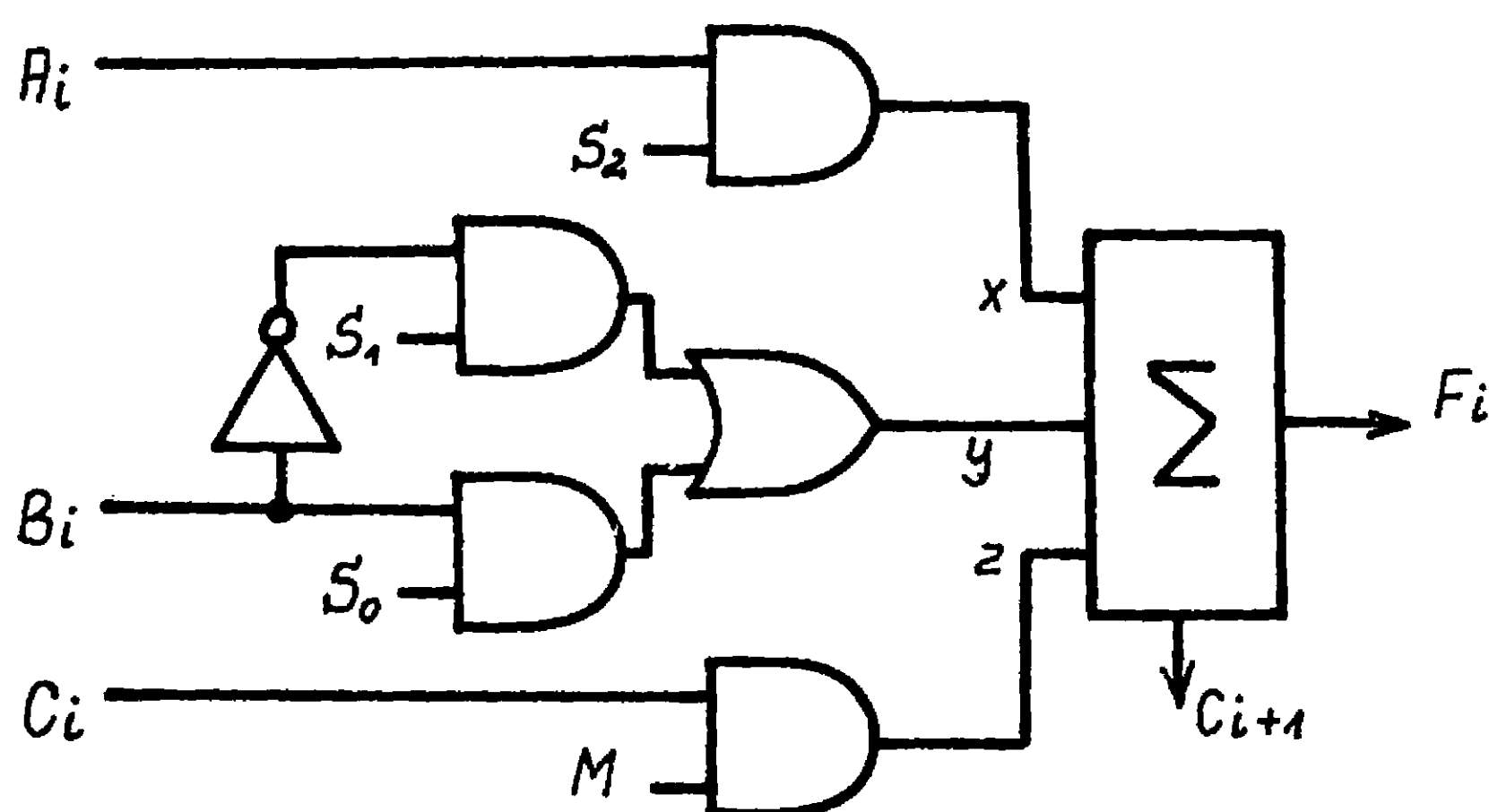
Rys. 7.6. Struktura programowalnego systemu cyfrowego



Rys. 7.7. Schemat logiczny pamięci RAM (a,b)

Moment zmiany stanu na wejściu (R/W) jest uwarunkowany sygnałem T podawanym z bramki iloczynu dołączonej do bloku pamięci. Istnieje także możliwość asynchronicznego ustawiania pamięci w stan czytania (Read) lub zapisu (Write) sygnałem logicznym 0. Zaznaczono to kółkami przy odpowiednich wejściach do bloku. Sygnały na wejścia adresowe, a więc na dekodery, są podawane z czterech dwuwejściowych multiplekserów (rozdzielaczy), których jedno wejście jest połączone z jednym z czterech najmłodszych bitów rejestru buforowego B, natomiast drugie - z jednym z wyjść synchronicznego licznika programu PC (Program Counter). Wybieranie wejścia licznika występuje dla adresu 1. Wejścia licznika programu są również połączone z czterema najmłodszymi bitami rejestru buforowego. Dla stanu wejścia $T = 1$, licznik dolicza do stanu wewnętrznego kolejne impulsy zegarowe. Jeśli na wejściu tym jest sygnał 0, w kolejnym impulsie zegarowym zostanie załadowana informacja z czterech najmłodszych bitów rejestru buforowego. Impuls synchronizacji jest podawany na wejście zegarowe poprzez dwuwejściową bramkę iloczynu.

B. Rejestr buforowy. Składa się on z 8 bitów, z których młodsze $D_0 \dots D_3$ tworzą część adresową, natomiast starsze $D_4 \dots D_7$, część operacyjną rozkazu odczytanego z pamięci. Wejścia rejestru są bezpośrednio połączone z wyjściami pamięci. Rejestr pracuje synchronicznie, przy czym impulsy zegarowe są podawane na dwuwejściową bramkę iloczynu. Wyjścia adresowe $D_0 \dots D_3$ są bezpośrednio połączone z wejściami licznika programu PC oraz z wejściami multipleksera (rozdzielacza) przekazującego adres do bloku pamięci. Sygnały na wejściach D_5, D_4 dochodzą bezpośrednio do wejść adresowych 12 multiplekserów o czterech wejściach informacyjnych, tworzących blok sterujący.



Rys. 7.8. Schemat sumatora jednobitowego z uwarunkowaniami

C. Jednostka arytmetyczno-logiczna (ALU - Arithmetic Logic Unit) wraz z rejestrem akumulatora. Podstawową strukturą ALU jest sumator z uwarunkowaniami, pokazany na rysunku 7.8. Symbole A_i, B_i, C_i reprezentują odpowiednio bit dodajnej, dodajnika i przeniesienia z niższej pozycji; F_i, C_{i+1} - są sumą i przeniesieniem na $i+1$ pozycję. Sygnały S_2, S_1, S_0, M warunkują wstęp do wejść sumatora poprzez bramki iloczynu. Dla $M = 0$ realizowane są wyłącznie funkcje logiczne, dla $M = 1$ logiczne i arytmetyczne (sumowanie). Stosując oznaczenia pokazane na rysunku 7.8, formuły opisujące poszczególne wyjścia można zapisać w postaci:

$$F_i = x \oplus y \oplus z = p \oplus z,$$

$$C_{i+1} = x \bar{y} + x z + y z = xy + (x \oplus y)z = xy + pz,$$

przy czym:

$$p = x \oplus y,$$

$$x = A_i S_2,$$

$$y = \bar{B}_i S_1 + B_i S_0,$$

$$z = C_i M.$$

Zbiór tych formuł pozwala określić operacje wykonywane przez wyjścia sumatora dla różnych kombinacji zmiennych sterujących S_2, S_1, S_0, M . Pełny zestaw funkcji zawiera tabela pokazana na rysunku 7.9.

$S_2 S_1 S_0$	F_i		C_{i+1}	
	$M=0$	$M=1$	$M=0$	$M=1$
0 0 0	0	C_i	0	0
0 0 1	B_i	$B_i \oplus C_i$	0	$B_i C_i$
0 1 0	$\bar{B}_i$	$\bar{B}_i \oplus C_i$	0	$\bar{B}_i C_i$
0 1 1	1	$\bar{C}_i$	0	C_i
1 0 0	A_i	$A_i \oplus C_i$	0	$A_i C_i$
1 0 1	$A_i \oplus B_i$	$A_i \oplus B_i \oplus C_i$	$A_i B_i$	$A_i B_i + A_i C_i + B_i C_i$
1 1 0	$A_i \odot B_i$	$A_i \oplus \bar{B}_i \oplus C_i$	$A_i \bar{B}_i$	$A_i \bar{B}_i + A_i C_i + \bar{B}_i C_i$
1 1 1	$\bar{A}_i$	$\bar{A}_i \oplus C_i$	A_i	$A_i + C_i$

Rys. 7.9. Zestaw funkcji realizowanych na wyjściu sumatora z uwarunkowaniami

Omawiany układ jest komórką iteracyjną, którą można łączyć w kaskadę celem wykonywania operacji na argumentach n -bitowych. Operacje logiczne dotyczą wówczas bitów z odpowiadających sobie pozycji, natomiast operacja dodawania arytmetycznego jest realizowana dla n -bitowej reprezentacji binarnej poszczególnych argumentów. W omawianym przypadku, sumowanie arytmetyczne będzie wykonywane, jeśli zmienne sterujące przyjmą wartości:

$$S_2 = 1, S_1 = 0, S_0 = 1, M = 1,$$

a na przykład dla:

$$S_2 = 1, S_1 = 1, S_0 = 1, M = 1$$

operacja wykonywana na bitach A_i, B_i ($i = 0, 1, \dots, n-1$) będzie operacją sumy logicznej.

Liczba operacji arytmetyczno-logicznych wykonywanych przez kaskadę jednobitowych sumatorów z uwarunkowaniami jest na ogół większa niż przewiduje to projektant systemu. Jednostka arytmetyczno-logiczna omawianego modelu programowalnego systemu cyfrowego wymaga tylko trzech sygnałów sterujących, potrzebnych do realizacji 8 operacji 8-bitowych. Operacje te przedstawiono w tabeli pokazanej na rysunku 7.10. Operację arytmetyczną dodawania opisano słownie dla odróżnienia jej od operacji logicznych, wykonywanych niezależnie na poszczególnych bitach.

S_2	S_1	S_0	Operacja $F(A, B)$
0	0	0	$F' = A$
0	0	1	$F' = A \text{ plus } B$
0	1	0	$F' = A \cdot B$
0	1	1	$F' = B$
1	0	0	$F' = \bar{A}$
1	0	1	$F' = A \oplus B$
1	1	0	$F' = A + B$
1	1	1	$F' = \bar{B}$

Rys. 7.10. Operacje wykonywane przez jednostkę ALU programowalnego systemu cyfrowego

Wyjścia jednostki arytmetyczno-logicznej są bezpośrednio połączone z wejściami 8-bitowego rejestru akumulującego i przechowującego rezultaty operacji. Rejestr ten jest uzupełniony dodatkowym przerzutnikiem E, przechowującym sygnał przeniesienia z jednostki ALU. Sterowanie rejestru odbywa się za pomocą dwóch sygnałów x_1, x_0 wg schematu:

$\langle x_1, x_0 \rangle = \langle 0, 0 \rangle$ - blokada zegara,

$\langle x_1, x_0 \rangle = \langle 1, 0 \rangle$ - przesuwanie w lewo,

$\langle x_1, x_0 \rangle = \langle 0, 1 \rangle$ - przesuwanie w prawo,

$\langle x_1, x_0 \rangle = \langle 1, 1 \rangle$ - wpisywanie równoległe.

W trakcie przesuwania w lewo informacja z bitu o najniższej wadze (F_7) jest wpisywana do przerzutnika E, natomiast informacja z przerzutnika przechodzi do bitu o najmniejszej wadze (F_0). Cyrkulacja informacji jest tu więc taka sama jak w liczniku pierścieniowym. Analogicznie przesuwanie w prawo powoduje przekazanie informacji z przerzutnika E do bitu F_7 , a z kolei do przerzutnika zostaje wpisana informacja z bitu F_0 .

Do wejść jednostki arytmetyczno-logicznej są bezpośrednio doprowadzane sygnały $M_7 \dots M_0$ z pamięci (argument B) oraz sygnały z wyjścia multipleksera rozdzielającego (argument A). Z kolei na wejścia tego multipleksera są podawane sygnały z wyjść rejestru akumulatora $F_7 \dots F_0$ oraz z wyjść rejestru wejściowego, którym w modelu są ręcznie ustawiane przełączniki $I_7 \dots I_0$. Jeśli stan wejścia adresowego wynosi 1, na 8 wyjściach multipleksera (rozdzielacza) pojawiają się sygnały $F_7 \dots F_0$, gdy 0 - sygnały z rejestru wejściowego a więc $I_7 \dots I_0$. Jednocześnie sygnały te pojawiają się na wejściu jednostki artmetyczno-logicznej jako argument A.

Taktowanie rejestru akumulatora i przerzutnika E jest realizowane przez podanie sygnału synchronizującego na dwuwejściową bramkę iloczynu.

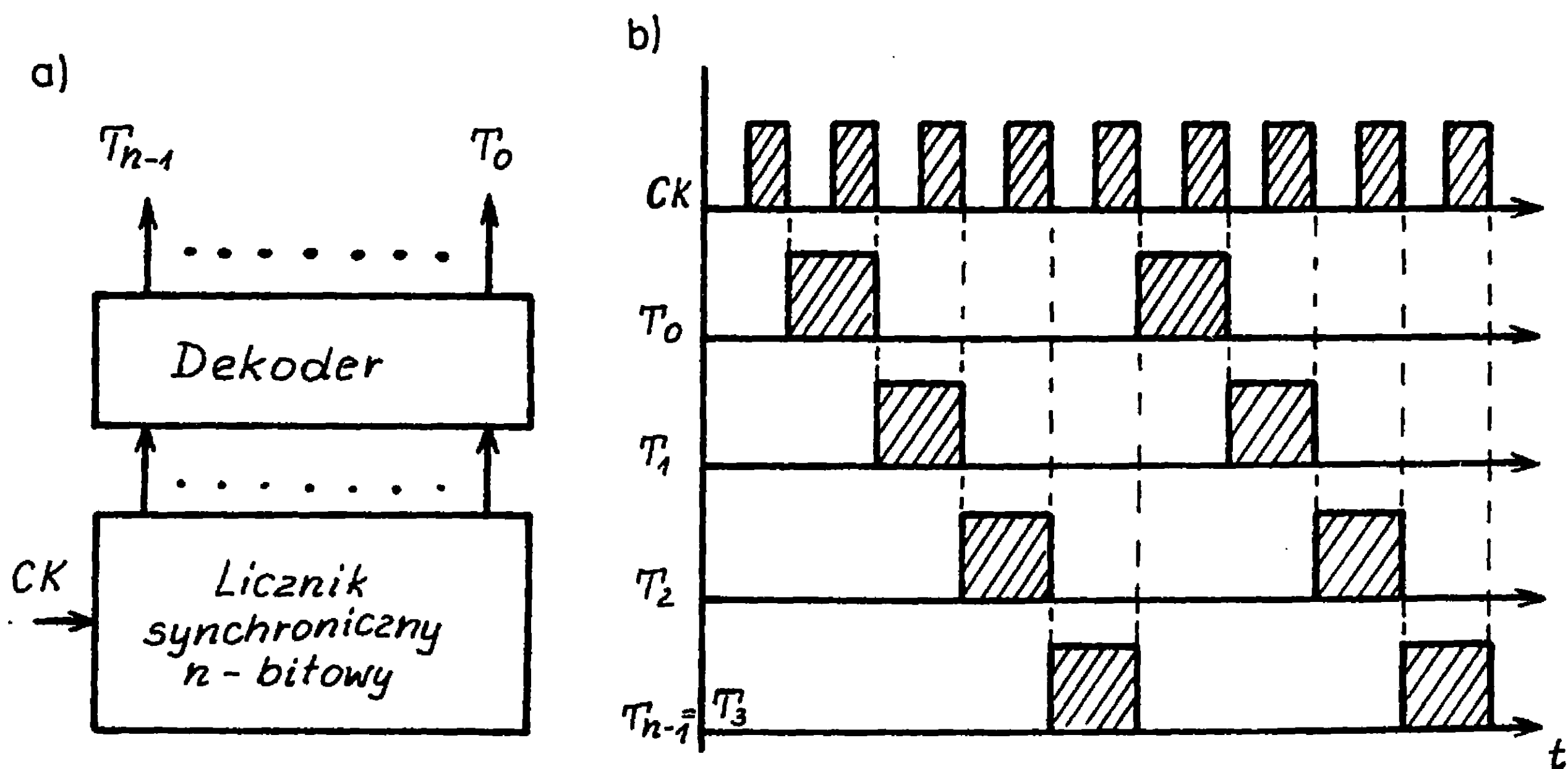
D. Generator impulsów taktujących. Został on zrealizowany w układzie licznika - dekodera, którego schemat blokowy pokazano na rysunku 7.11a. Wyjścia z 4-bitowego synchronicznego licznika binarnego są podawane na wejścia dekodera. W efekcie na wyjściach występują sygnały logiczne 1, dekodujące stan licznika. Przebiegi czasowe tych sygnałów pokazano na rysunku 7.11b. Sygnały z dekodera są dodatkowo podawane na wejścia inwerterów. Jeden cykl pracy układu zawiera cztery impulsy taktujące, a więc w procesie synchronizacji systemu można wykorzystać 8 zboczy: 4 - przednie i 4 - tylne. Wszystkie bloki systemu są wyzwalane zboczami impulsów taktujących. Praca układu może zostać zatrzymana przez podanie sygnału 0 na wejście HLT (rys. 7.6). Impulsy zegarowe podawane na licznik zostają wówczas zablokowane w wyniku koniunkcji z tym sygnałem. Możliwa jest praca krokowa zegara, sterowana przyciskiem kontaktronowym.



E. Układ sterowania. Podstawowym elementem układu jest blok 12 multiplekserów o czterech wejściach informacyjnych i dwóch wejściach adresowych. Wejścia adresowe są połączone na stałe z wyjściami D_5D_4 rejestru buforowego, przy czym bit D_5 reprezentuje wyższą wagę adresu. Do wejść informacyjnych można dołączyć następujące sygnały lub ich negacje: C_n (przeniesienie z wyjścia przerzutnika E), D_7 , D_6 , D_3 . Funkcje sterujące realizowane na wyjściu multiplekserów mają więc następującą postać ogólną:

$$Y_i = f_i(C_n, D_7, D_6, D_5, D_4, D_3), \quad (i = 1, \dots, 12).$$

Ich postaci szczególne zależą od realizowanego wariantu sterowania. Synteza właściwego wariantu jest głównym celem projektowym ćwiczenia.



Rys. 7.11. Schemat układu licznika - dekodera i przebiegi czasowe na wyjściach (a,b)

Z wariantem sterowania wiąże się określona lista rozkazów realizowanych przez system. Z kolei - analogicznie jak w maszynie Posta - w oparciu o taką listę można tworzyć programy przystosowane do realizacji konkretnych zadań. Oprogramowanie zrealizowanego wariantu systemu jest celem programowym ćwiczenia. Wprowadzenia programu do pamięci RAM dokonuje się poprzez ręcznie sterowany rejestr wejściowy złożony z przełączników $I_7, \dots, I_0$, sterujących wejściami multipleksera rozdzielającego sygnały z rejestru akumulatora i rejestru wejściowego.

Dodatkowo podając sygnał 0 na wejście RESET, można asynchronicznie wyzerować wszystkie rejestry wewnętrzne z wyjątkiem rejestrów tworzących pamięć RAM. W okresie ładowania programu do pamięci, wejścia sterujące poszczególnych bloków mogą być częściowo sterowane ręcznie.

Pełny wykaz wejść sterujących systemu obejmuje następujące pozycje:

- $\bar{R}$ - asynchroniczne ustawianie pamięci RAM w stanie odczytu (Read),
- $\bar{W}$ - asynchroniczne ustawianie pamięci RAM w stanie zapisu (Write),
- R/W - synchroniczne ustawianie pamięci w stanie zapis (1) lub odczyt (0),
- a - wejście adresowe multipleksera dekodera pamięci RAM,
- b - wejście rodzaju pracy licznika programu PC,
- $\bar{C}_0$ - wejście przeniesienia (Carry) na najbliższą pozycję jednostki arytmetyczno-logicznej,
- d - wejście adresowe multipleksera rozdzielającego sygnały z wyjścia akumulatora i wyjścia rejestru $I_7, \dots, I_0$,
- x_1, x_0 - wejścia sterujące rejestru akumulatora,
- S_2, S_1, S_0 - wejścia sterujące operacjami jednostki arytmetyczno-logicznej,
- HLT - zatrzymywanie pracy generatora impulsów taktujących,
- RESET - zerowanie rejestrów wewnętrznych systemu, bramki koniunkcji na wejściach taktujących rejestru buforowego, akumulatora, licznika programu i zapisu lub odczytu pamięci RAM.

Wejścia powyższe albo są sterowane z wyjść Y_i multiplekserów, albo służą do ręcznego wprowadzania programu, bądź też pozostają w zrealizowanym wariantcie sterowania niewykorzystane. Sygnał na wejściu niewykorzystanym jest odczytany przez system jako logiczne 1.

7.5.2. Przykład wariantu sterowania

Wszystkie multipleksery bloku sterującego mają wejścia adresowe połączone na stałe z wyjściami D_5, D_4 rejestru buforowego. Rozkazy można więc podzielić na cztery zasadnicze grupy, zależnie od sekwencji sterującej wejść adresowych. Jednocześnie bez ograniczeń, każdy multipleksers może zrealizować tylko funkcję trzech argumentów, z ograniczeniami - sześciu.

Wpisywanie programu poprzedza wyzerowanie rejestrów, w związku z tym wygodnie jest przyjąć jako kod rozkazu ładowania akumulatora stan bitów rejestru buforowego $\langle D_7 D_6 D_5 D_4 \rangle = \langle 0000 \rangle$. Ładowanie odbywa się wówczas z rejestru wejściowego $I_7 \dots I_0$, a kod 0000 zostaje wymuszony podaniem 0 na wejście RESET. Na wejścia sterujące jednostki arytmetyczno-logicznej należy, zgodnie z tabelą pokazaną na rysunku 7.10, podać sygnały: $\langle S_2 S_1 S_0 \rangle = \langle 000 \rangle$. Rejestr akumulatora winien być załadowany, a więc $\langle x_1 x_0 \rangle = \langle 1 1 \rangle$, multiplexer na wejściu adresowym d) musi mieć sygnał $d = 0$ (rys. 7.6), co łączy rejestr wejściowy z argumentem A jednostki arytmetyczno-logicznej. Do wejścia asynchronicznego $\bar{W}$ pamięci RAM inicjującego zapis należy przyłożyć sygnał $\bar{W} = 0$, natomiast do wejść sterujących licznika PC: $b = 1$. Stan 1 winien być również na wejściach $\bar{C}_0$ oraz HLT, uaktywnianych tylko sygnałem 0. W omawianym wariancie sterowania impulsy taktujące są uwarunkowane sygnałami koniunkcji tylko dla pamięci RAM oraz dla rejestru akumulatora. Podczas asynchronicznego zapisu, a więc gdy $\bar{W} = 0$ stan wejścia warunkującego takty jest nieistotny, natomiast dla rejestru akumulatora wymagana wartość sygnału wynosi 1. Pozostałe wejścia w rozpatrywanym wariancie są niewykorzystywane, co utożsamia się z występowaniem na nich stabilnej wartości 1.

Ładowanie programu jest realizowane po wyzerowaniu rejestru sygnałem RESET = 0. Kod rozkazu pobierania danych z wejścia będzie więc miał postać: $\langle D_7 D_6 D_5 D_4 D_3 \rangle = \langle 0 0 0 0 0 \rangle$. Przypiszmy mu nazwę mnemoniczną INP A. W grupie rozkazów o adresie multiplexerów układu sterującego $\langle D_5 D_4 \rangle = \langle 0 0 \rangle$, wygodnie jest umieścić te rozkazy, które nie wymagają wykonywania operacji przez jednostkę arytmetyczno-logiczną. Są nimi:

- INPA: pobieranie danych z wejścia (rozkaz wyżej omówiony),
- RLA: rotacja akumulatora w lewo,
- RRA: rotacja akumulatora w prawo,
- STA: zapamiętanie informacji z akumulatora w pamięci RAM.

Dodatkowo do tej grupy rozkazów można dołączyć rozkaz pobierania negacji danych z wejścia. Zmienne sterujące jednostki arytmetyczno-logicznej (rys. 7.10) winny wówczas przyjąć wartość $\langle S_2 S_1 S_0 \rangle = \langle 1 0 0 \rangle$. Kod rozkazu wystarczy, jeśli będzie się różnił tylko na pozycji D_3 , tzn. $\langle D_7 D_6 D_5 D_4 D_3 \rangle = \langle 0 0 0 0 1 \rangle$. Mnemonik określający ten rozkaz nie ulega zmianie, należy tylko w miejsce argumentu A wpisać symbol $\bar{A}$: INP $\bar{A}$.

W tabeli pokazanej na rysunku 7.12 przedstawiono funkcje sterujące związane z wymienionymi wyżej rozkazami oraz funkcje dotyczące rozkazów należących do grupy, dla której $\langle D_5 D_4 \rangle = \langle 1 \ 1 \rangle$. W grupie tej jest rozkaz HLT blokujący pracę zegara. Funkcji sterującej związanej z tym rozkazem nie można połączyć z innymi rozkazami, co zawęża ich liczbę w tej grupie do trzech. Są to rozkazy:

- HLT: zatrzymanie zegara,
- JNC: skocz, jeśli nie ma przeniesienia,
- IJNC: zwiększ akumulator o jeden i skocz, jeśli nie ma przeniesienia.

	INPA	INPĀ	RLA	RRA	STA	JNC	IJNC	HALT
S_2	0	1	X	X	X	0	0	X
S_1	0	0	X	X	X	0	0	X
S_0	0	0	X	X	X	0	0	X
x_0	1	1	0	1	0	1	1	X
x_1	1	1	1	0	0	1	1	X
d	0	X	X	X	X	1	1	X
$\bar{w}$	0	0	0	1	1	0	0	X
$T(RAM)$	X	X	X	0	1	X	X	X
b	1	1	1	1	1	C	C	X
$\bar{C}_0$	1	X	X	X	X	1	0	X
$T(AC)$	1	1	1	1	1	0	1	X
HLT	1	1	1	1	1	1	1	0
Kod	D_7	0	0	1	0	1	0	1
	D_6	0	0	0	1	1	0	0
	D_5	0	0	0	0	0	1	1
	D_4	0	0	0	0	0	1	1
	D_3	0	1	-	-	-	-	-

Rys. 7.12. Funkcje sterujące dla grup adresowych $\langle D_5 D_4 \rangle = \langle 0 \ 0 \rangle$ i $\langle D_5 D_4 \rangle = \langle 1 \ 1 \rangle$

Rozkazy operacji logicznych i arytmetycznych mogą zostać zrealizowane w grupach adresowych multiplekserów sterujących $\langle D_5 D_4 \rangle = \langle 0 \ 1 \rangle$ oraz $\langle D_5 D_4 \rangle = \langle 1 \ 0 \rangle$. Dla odróżnienia rozkazów, w których jest wskazany adres pamięci RAM od rozkazów bezadresowych można użyć bitu D_3 .

Będzie on sterował wejściem przeniesienia jednostki arytmetyczno-logicznej (ALU). Aby nie powodowało to fałszowania wyników w pozostałych operacjach z danej grupy, muszą do niej należeć tylko te, które nie uwzględniają stanu przeniesienia z przerzutnika E (rys. 7.6), a więc operacje logiczne. W konsekwencji dokonano następującego podziału rozkazów na grupy:

1) dla $\langle D_5 D_4 \rangle = \langle 1 \ 0 \rangle$:

- INC A: zwiększ zawartość akumulatora o jeden,
- RES C: skasuj przeniesienie,
- AND A, ... : wykonaj operację AND na bitach akumulatora i bitach słowa pamięci zawartych w rejestrze pod adresem ... ,
- OR A, ... : wykonaj operację OR na bitach akumulatora i bitach słowa pamięci zawartych w rejestrze pod adresem ... ,
- NEGA: zaneguj zawartość akumulatora,

2) dla $\langle D_5 D_4 \rangle = \langle 0 \ 1 \rangle$:

- LDA, ... : załaduj akumulator zawartością rejestru (słowa) pamięci o adresie ... ,
- LDĀ, ... : załaduj do akumulatora słowo zapisane w pamięci pod adresem ... i zaneguj wszystkie bity tego słowa,
- ADD A, ... : dodaj arytmetycznie do zawartości akumulatora słowo z pamięci znajdujące się w rejestrze pod adresem ... ,
- XOR A, ... : wykonaj operację EXCLUSIVE-OR na bitach akumulatora i bitach słowa pamięci zawartych w rejestrze pod adresem ...

W tabeli pokazanej na rysunku 7.13 przedstawiono funkcje sterujące związane z wymienionymi wyżej rozkazami.

Do synchronizacji pracy poszczególnych bloków trzeba wykorzystać impulsy taktujące lub ich negacje. Wyzerowanie systemu ustawia blok w takcie T_0 (rys. 7.6 i 7.11). Pobranie rozkazu z rejestru buforowego B wygodnie jest zrealizować w połowie taktu T_1 . Osiąga się to podając na bramkę iloczynu wejścia taktujące sygnał $T_1 \cdot (\overline{CK})$. W tym czasie pamięć musi być adresowana przez licznik rozkazów PC, a więc na wejście adresowe multipleksa sterującego dekodera adresów pamięci RAM należy podać również sygnał $a = T_1$. Wejście taktujące rejestru akumulatora łączymy z sygnałem $\overline{T}_2$. Dzięki temu ewentualne zmiany na wejściach $x_1 x_0$ nie wygenerują fałszywych impulsów taktujących. Należy zaznaczyć, że sygnał taktujący rejestr akumulatora jest utworzony z iloczynu $\overline{T}_2 \cdot [T(AC)]$ (rys. 7.12 i 7.13). Sygnał $T(AC)$ jest równy zero tylko dla

rozkazu: JNC - skocz, jeśli nie ma przeniesienia. Zapis do pamięci jest taktowany sygnałem T_2 (w koniunkcji z sygnałem $T(RAM)$), natomiast po dokonaniu zapisu, sygnał $\bar{T}_3$ przełącza wejście $\bar{R}$ pamięci w stan odczytu. Ten sam sygnał zwiększa również zawartości licznika programu PC.

		AND A_i (...)	INC A_i	RES C_i	OR A_i (...)	NEG A_i	LDA A_i (...)	ADDA A_i (...)	LDA A_i (...)	XOR A_i (...)
	S_2	0	0	0	1	1	0	0	1	1
	S_1	1	0	0	1	0	1	0	1	0
	S_0	0	0	0	0	0	1	1	1	1
	x_0	1	1	1	1	1	1	1	1	1
	x_1	1	1	1	1	1	1	1	1	1
	d	1	1	1	1	1	1	1	1	1
	$\bar{w}$	0	0	0	0	0	0	0	0	0
	$T(RAM)$	X	X	X	X	X	X	X	X	X
	b	1	1	1	1	1	1	1	1	1
	$\bar{C}_0$	X	0	1	X	X	X	1	X	X
	$T(AC)$	1	1	1	1	1	1	1	1	1
	HLT	1	1	1	1	1	1	1	1	1
Kod	D_7	0	1	1	0	1	0	1	0	1
	D_6	0	0	0	1	1	0	0	1	1
	D_5	1	1	1	1	1	0	0	0	0
	D_4	0	0	0	0	0	1	1	1	1
	D_3	—	0	1	—	—	—	—	—	—

Rys. 7.13. Funkcje sterujące dla grup adresowych
 $\langle D_5 D_4 \rangle = \langle 1 0 \rangle$ oraz $\langle D_5 D_4 \rangle = \langle 0 1 \rangle$

Na rysunku 7.14 pokazano przebiegi czasowe taktowania systemu, opisane przez następujący omówiony wyżej schemat sterowania odpowiednich jego wejść (rys. 7.6).

$\overline{CK}$ - wejście zegarowe rejestru buforowego,

T_1 - wejście zegarowe rejestru buforowego, (koniunkcja z $\overline{CK}$),

T_1 - wejście adresowe (a) multipleksera adresów pamięci RAM,

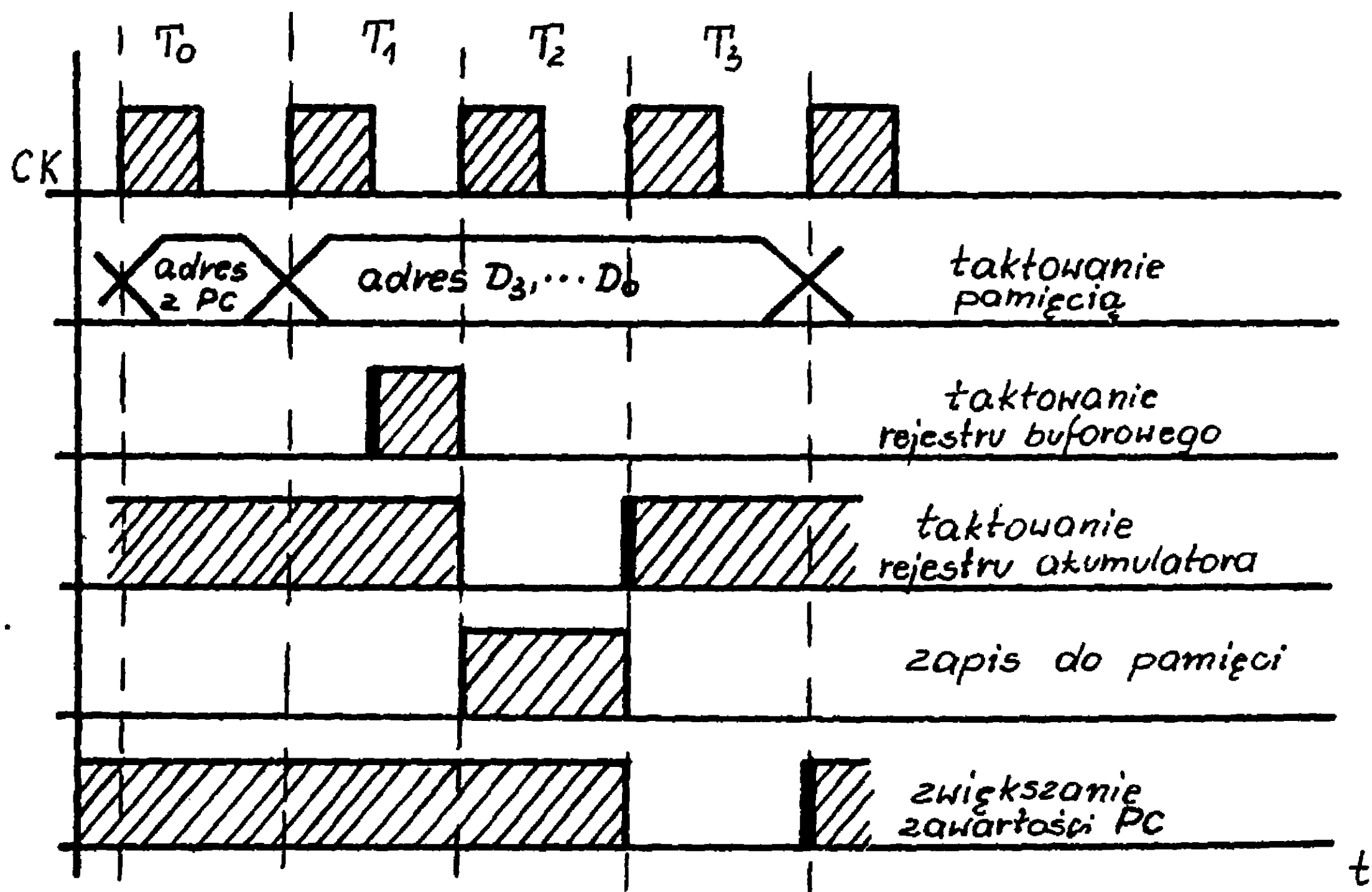
$\overline{T}_2$ - wejście zegarowe rejestru akumulatora,

T_2 - wejście zegarowe zapisu do pamięci,

$\overline{T}_3$ - asynchroniczne wejście ustawiania odczytu z pamięci,

$\overline{T}_3$ - zwiększanie zawartości licznika rozkazów.

Grubymi liniami zaznaczono zbocza, na których występuje uaktywnienie danego rejestru.



Rys. 7.14. Przebiegi czasowe taktowania systemu

Wyjście multiplekserów bloku sterującego muszą być połączone z odpowiednimi wejściami sterującymi poszczególnych bloków. Każda z tych funkcji jest zależna od bitów D_5 i D_4 , które są na stałe połączone z wejściami adresowymi wszystkich multiplekserów. Na wejścia informacyjne należy więc podać takie sygnały, które pozwolą odróżnić kody poszczególnych grup adresowych. Przykładowo, dla wejścia sterującego S_2 jednostki arytmetyczno-logicznej, otrzymamy następujący zestaw kodów (rys. 7.12 i rys. 7.13):

D_7	D_6	D_5	D_4	D_3	S_2	
0	0	0	0	0	0	
0	0	0	0	1	1	$0 \rightarrow D_3$
0	0	1	0		0	
1	0	1	0	0	0	
1	0	1	0	1	0	$2 \rightarrow D_6$
0	1	1	0		1	
1	1	1	0		1	
0	0	0	1		0	
1	0	0	1		0	$1 \rightarrow D_6$
0	1	0	1		1	
1	1	0	1		1	
0	0	1	1		0	
1	0	1	1		0	$3 \rightarrow 0$

Dla odróżnienia sygnałów pochodzących od rozkazów w ramach grupy $\langle D_5 D_4 \rangle = \langle 0 0 \rangle$, do wejścia informacyjnego o tym adresie wystarczy podać sygnał D_3 . Z kolei sygnały sterujące wejścia S_2 pochodzące od rozkazów grupy $\langle D_5 D_4 \rangle = \langle 0 1 \rangle$ oraz grupy $\langle D_5 D_4 \rangle = \langle 1 0 \rangle$ będą jednoznacznie rozróżnialne, gdy wejścia informacyjne multiplexerów oznaczone numerami 1 i 2 zostaną połączone z sygnałem D_6 . W grupie $\langle D_5 D_4 \rangle = \langle 1 1 \rangle$ wejście S_2 winno byćysterowane sygnałem logicznym 0. Taki też sygnał należy podać na 3 wejście informacyjne rozpa-trywanego multipleksera.

Z powyższego przykładu wynika, że kody rozkazów nie mogą być wybierane dowolnie. Konieczne jest takie ich zestawienie w ramach grup adresowych, by określone wejścia sterujące mogły je rozróżnić. Muszą więc być one dostosowane do możliwości realizacyjnych multiplexerów bloku sterującego. Gdyby w grupie o adresie $\langle D_5 D_4 \rangle = \langle 1 0 \rangle$, wejście było równe 1 jeszcze np. dla kodu $\langle D_7 D_6 D_5 D_4 \rangle = \langle 0 0 1 0 \rangle$, to trzeba by zmienić koncepcję sterowania, gdyż multiplekser nie byłby w stanie odróżnić tej sekwencji bitów rejestru buforowego. A więc kody rozkazów w ramach tej samej grupy, dające na danym wejściu sterującym ten sam sygnał mogą się różnić tylko na jednej pozycji (D_6 lub D_7).

Bit D_3 odgrywa rolę pomocniczą i może być wykorzystany tylko w rozkazach bezadresowych. Kody rozkazów w rozpatrywanym wariantcie zostały tak dobrane, by sprostać tym ograniczeniom.

W analogiczny sposób zostały określone sygnały podawane na wejścia informacyjne pozostałych multiplexerów. Pełny zestaw funkcji sterujących wg kolejności sterowania wymienionej w tabelach pokazanych na rysunkach 7.12 i 7.13 będzie następujący:

- wejście S_2 - sterowanie jednostki arytmetyczno-logicznej

wejścia: 0 - D_3 $S_2 = Y_1 = D_3(\bar{D}_5\bar{D}_4) + D_6(\bar{D}_5D_4) + D_6(D_5\bar{D}_4)$

1 - D_6

2 - D_6

3 - 0,

- wejście S_1 - sterowanie jednostki arytmetyczno-logicznej

wejścia: 0 - 0 $S_1 = Y_2 = \bar{D}_7(\bar{D}_5D_4) + \bar{D}_7(D_5\bar{D}_4)$

1 - $\bar{D}_7$

2 - $\bar{D}_7$

3 - 0,

- wejście S_0 - sterowanie jednostki arytmetyczno-logicznej

wejścia: 0 - 0 $S_0 = Y_3 = \bar{D}_5D_4$

1 - 1

2 - 0

3 - 0,

- wejście x_0 - sterowanie rejestru akumulatora:

wejścia: 0 - $\bar{D}_7$ $x_0 = Y_4 = \bar{D}_7(\bar{D}_5\bar{D}_4) + \bar{D}_5D_4 + D_5\bar{D}_4 + D_5D_4$

1 - 1

2 - 1

3 - 1,

- wejście x_1 - sterowanie rejestru akumulatora:

wejścia: 0 - $\bar{D}_6$ $x_1 = Y_5 = \bar{D}_6\bar{D}_5\bar{D}_4 + \bar{D}_5D_4 + D_5\bar{D}_4 + D_5D_4$

1 - 1

2 - 1

3 - 1,

- wejście d - sterowanie multiplexera rozdzielającego argument A jednostki arytmetyczno-logicznej:

wejścia: 0 - D_6 $d = Y_6 = D_6(\bar{D}_5\bar{D}_4) + \bar{D}_5D_4 + D_5\bar{D}_4 + D_5D_4$

1 - 1

2 - 1

3 - 1,

- wejście $\bar{W}$ - asynchroniczny zapis do pamięci RAM:

wejścia: 0 - D_6 $\bar{W} = Y_7 = D_6(\bar{D}_5\bar{D}_4)$

1 - 0

2 - 0

3 - 0,

- wejście $T(RAM)$ - koniunkcyjne uwarunkowanie taktowania pamięci - wystarczy połączyć bezpośrednio z bitem D_7 , tzn.

$$[T(RAM)] = D_7,$$

- wejście b - sterowane pracą licznika programu PC:

wejścia: 0 - 1 $b = Y_8 = \bar{D}_5\bar{D}_4 + \bar{D}_5D_4 + D_5\bar{D}_4 + C_n(D_5D_4)$

1 - 1

2 - 1

3 - C_n ,

- wejście $\bar{C}_0$ - sterowanie pierwszej pozycji przeniesienia jednostki arytmetyczno-logicznej:

wejścia: 0 - 1 $\bar{C}_0 = Y_9 = \bar{D}_5\bar{D}_4 + \bar{D}_5D_4 + D_3(D_5\bar{D}_4) + \bar{D}_7(D_5D_4)$

1 - 1

2 - D_3

3 - $\bar{D}_7$,

- wejście $T(AC)$ - koniunkcyjne uwarunkowanie taktowania rejestru akumulatora:

wejścia: 0 - 1 $T(AC) = Y_{10} = \bar{D}_5\bar{D}_4 + \bar{D}_5D_4 + D_5\bar{D}_4 + D_7(D_5D_4)$

1 - 1

2 - 1

3 - D_7 ,

- wejście HLT - zatrzymanie pracy zegara:

wejścia: 0 - 1 $HLT = Y_{11} = \bar{D}_5\bar{D}_4 + \bar{D}_5D_4 + D_5\bar{D}_4 + \bar{D}_6(D_5D_4)$

1 - 1

2 - 1

3 - $\bar{D}_6$.

7.5.3. Przykłady oprogramowania

Wprowadzanie programu do pamięci RAM systemu wymaga wykonania następujących ręcznych manipulacji:

1) przełączenia wyzwalania impulsów zegarowych na pracę ręczną krokową,

2) przełączenia wejścia a multipleksera sterującego dekoderek pamięci i ustawienia stanu $a = 1$,

3) połączenia wejścia taktującego rejestru buforowego ze stanem 0 logiczne,

4) połączenia wejścia asynchronicznego ustawiania zapisu $\bar{W}$ do pamięci RAM z wyjściem taktującym $\bar{T}_2$,

5) wyzerowanie rejestrów podaniem sygnału 0 na wejście RESET.

Czynności 1-4 są wykonywane jednocześnie jednym przyciskiem typu IZOSTAT. Po wpisaniu programu tym samym przyciskiem zostaje przywrócona pierwotna konfiguracja i należy ponownie wyzerować układ.

Pełna lista rozkazów omówionego wariantu sterowania systemu, ze stawiona na podstawie tabel pokazanych na rysunkach 7.12 i 7.13, obejmuje 17 pozycji:

Mnemonic K O D

	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
INP A	0	0	0	0	0	X	X	X	
INP $\bar{A}$	0	0	0	0	1	X	X	X	
LDA, ...	0	0	0	1					adres
LD $\bar{A}$, ...	0	1	0	1					adres
STA, ...	1	1	0	0					adres
RLA	1	0	0	0	X	X	X	X	
RRA	0	1	0	0	X	X	X	X	
NEG A	1	1	1	0	X	X	X	X	
INC A	1	0	1	0	0	X	X	X	
RES C	1	0	1	0	1	X	X	X	
AND A, ...	0	0	1	0					adres
OR A, ...	0	1	1	0					adres
XOR A, ...	1	1	0	1					adres
ADD A, ...	1	0	0	1					adres
JNC (N)	0	0	1	1					adres

N - adres skoku do określonego rejestru pamięci RAM

IJNC (N)	1	0	1	1	adres			
HLT	1	1	1	1	X	X	X	X
	0	1	1	1	X	X	X	X

Rozkaz HLT w omawianym wariantcie może mieć dwa kody. W oparciu o przedstawioną listę zrealizowano szereg wymienionych niżej procedur programowych ilustrujących możliwości kryjące się w niezwykle skromnej architekturze systemu, posiadającego ponadto 16 rejestrów 8-bitowych, pamięci operacyjnej RAM. Programy te powinny pozwolić wyrobić sobie pogląd na ogromne możliwości operacyjne wielkich systemów cyfrowych tworzących nowoczesną technikę komputerową.

A . Dodawanie liczb czterobitowych

Adres	Mnemonik	K 0 0			
0	INP A	0 0 0 0	0 0 0 0		
1	AND A, (1 1)	0 0 1 0	1 0 1 1		
2	STA, (1 5)	1 1 0 0	1 1 1 1		
3	INP A	0 0 0 0	0 0 0 0		
4	RR A	0 1 0 0	0 0 0 0	I argument: $I_3, \dots, I_0$	
5	RR A	0 1 0 0	0 0 0 0	II argument: $I_7, \dots, I_4$	
6	RR A	0 1 0 0	0 0 0 0		
7	RR A	0 1 0 0	0 0 0 0		
8	AND A, (1 1)	0 0 1 0	1 0 1 1		
9	ADD A, (1 5)	1 0 0 1	1 1 1 1		
10	HLT	1 1 1 1	1 1 1 1		
11	Maska	0 0 0 0	1 1 1 1		

Opis programu. Wymienione wyżej rozkazy znajdują się w rejestrach pamięci RAM pod zaznaczonymi adresami. Program zajmuje łącznie ze słowem maskującym (1 1) dwanaście rejestrów. Po wstępnym wyzerowaniu systemu, w takcie T_0 do dekodera pamięci jest doprowadzony adres 0 0 0 0. Pod tym adresem znajduje się rozkaz INP A powodujący wczytanie do rejestru akumulatora z rejestru wejściowego $I_7, \dots, I_0$ pierwszego argumentu. Następnie, jest wykonywana operacja AND na bitach rejestru akumulatora i bitach rejestru pamięci znajdującej się pod adresem 11. W wyniku zostają wyzerowane cztery starsze bity wczytanego argumentu.

Operację taką nazywa się operacją maskowania. Rezultat zostaje zapamiętany w rejestrze pod adresem 15. Następuje pobranie kolejnego argumentu z rejestru wejściowego i następnie czterokrotne przesunięcie rotacyjne w prawo (rozkazy 4÷7) zawartości rejestru akumulatora, co przesuwca czterobitowy argument na pozycje zajmowane przez najstarsze bity. Po kolejnej operacji maskowania następuje dodawanie arytmetyczne obu argumentów i przesłanie wyniku do rejestru pamięci o adresie 15. Rozkaz znajdujący się w rejestrze o adresie 10 zatrzymuje zegar - obliczenie zostało wykonane.

Praca programu jest synchronizowana zgodnie z przebiegami taktującymi pokazanymi na rysunku 7.14. Z uwagi na brak rozkazu skoku kolejny adres rejestru pamięci jest wybierany poprzez multiplexer z wyjść licznika programu PC, który pracuje w sekwencji binarnej zmieniając zawartość każdorazowo na tylnym zboczu impulsu T_3 , a więc na samym końcu czterotaktowego cyklu zegarowego. W ten sposób w taktach T_0 z pamięci są pobierane kolejne rozkazy (instrukcje) realizowanego programu, od 0 do 10.

Skoki pośrednie dla rozkazów adresowych są sterowane sygnałem na wejściu a multiplexera dekodera adresu pamięci RAM. Jeśli $a = 0$ do dekodera jest przekazywany adres zawarty w czterech najmłodszych bitach $D_3, \dots, D_0$ rejestru buforowego, co powoduje - zależnie od stanu wejść sterujących pamięci - zapis lub odczyt pod wskazanym adresem odpowiedniej sekwencji binarnej. Przed końcem impulsu taktującego T_3 następuje powrót do wartości $a = 1$, co narzuca z powrotem adres zawarty w liczniku programu, zwiększany o jeden na tylnym zboczu impulsu T_3 .

B. Odejmowanie liczb czterobitowych

Adres	Mnemonic	K o d		
0	INP $\bar{A}$	0 0 0 0	1 1 1 1	
1	OR A, (11)	0 1 1 0	1 0 1 1	
2	INC A	1 0 1 0	0 0 0 0	Odjemna: $I_7, \dots, I_4$
3	STA, (15)	1 1 0 0	1 1 1 1	Odjemnik: $I_3, \dots, I_0$
4	INP A	0 0 0 0	0 0 0 0	
5÷8	RR A	0 1 0 0	0 0 0 0	
9	AND A, (0)	0 0 1 0	0 0 0 0	
10	ADD A, (15)	1 0 0 1	1 1 1 1	
11	HLT	1 1 1 1	0 0 0 0	

Opis programu. Operacje 0,1,2,3 obliczają uzupełnienie dwójkowe wczytanego argumentu, maskują cztery starsze bity (rozkaż nr 1), wykorzystując do tego celu zawartość rejestru o adresie 11, gdzie jest umieszczony kod instrukcji HLT. Wynik maskowania zostaje zapamiętany w rejestrze o adresie 15. Dalszy przebieg programu - analogiczny jak przy dodawaniu. Do maskowania odjemnika wykorzystano kod rozkazu INP $\bar{A}$. C. Mnożenie liczb czterobitowych

Adres	Mnemonic	K o d		
0	STA (15)	1 1 0 0	1 1 1 1	
1	INP A	0 0 0 0	0 0 0 0	
2	ADD A (12)	1 0 0 1	1 1 0 0	
3	JNC (11)	0 0 1 1	1 0 1 1	
4	STA (14)	1 1 0 0	1 1 1 0	I argument: $I_3, \dots, I_0$
5	AND A (13)	0 0 1 0	1 1 0 1	II argument: $I_7, \dots, I_4$
6	ADD A (15)	1 0 0 1	1 1 1 1	
7	STA (15)	1 1 0 0	1 1 1 1	
8	LDA (14)	0 0 0 1	1 1 1 0	
9	RES C	1 0 1 0	1 0 0 0	
10	JNC 2	0 0 1 1	0 0 1 0	
11	LDA (15)	0 0 0 1	1 1 1 1	
12	HLT	1 1 1 1	0 0 0 0	
13	Maska	0 0 0 0	1 1 1 1	

Opis programu. Mnożenie zostało tu sprowadzone do dodawania w pętli. Oba argumenty czterobitowe umieszczono razem, przy czym starsze cztery bity są traktowane jako wskaźnik pętli, w której młodsze bity są wielokrotnie dodawane. Rozkaż nr 0 zapamiętuje wyzerowaną wartość akumulatora w rejestrze pamięci pod adresem 15, - inaczej mówiąc, zeruje ten rejestr. Następnie są pobierane argumenty z wejścia $I_7, \dots, I_0$. Do czterech starszych bitów następuje dodawanie liczby (-1) (w uzupełnieniu dwójkowym), co wygenerowuje przeniesienie C_n . Jeśli $C_n = 0$, to pętla winna być zakończona. Program wykonuje skok do rozkazu, którego kod znajduje się w rejestrze o adresie 11. Akumulator zostaje wówczas załadowany rezultatem obliczeń umieszczonym sukcesywnie w trakcie realizacji pętli, w rejestrze o adresie 15. Potem program jest zatrzymany rozkazem HLT.

kodu 1 z 8. Tak otrzymany adres jest właśnie adresem indeksowym. Jeśli np. $\langle I_2 I_1 I_0 \rangle = \langle 0 \ 1 \ 0 \rangle$, to adres indeksowy wynosi $7 + 2 = 9$. Pod tym adresem jest 1 dla bitu o wadze 2. Ten rezultat zostanie zapamiętany w rejestrze pamięci pod adresem 4 i w momencie, gdy licznik programu wywoła rozkaz nr 4, wynik 0 0 0 0 0 0 1 0 pojawi się w rejestrze akumulatora.



1472/15

LITERATURA

- [1] Dzikowski M., Mroczek H.: Teoria automatów, materiały pomocnicze (skrypt). RWN PŁ, Łódź 1977.
- [2] Majewski W.: Układy logiczne. WNT, Warszawa 1976.
- [3] Mostowski A.W., Pawlak Z.: Logika dla inżynierów. PWN, Warszawa 1970.
- [4] Perkowski M., Rydzewski A., Misiurewicz P.: Teoria układów logicznych (skrypt). PW, Warszawa 1984.
- [5] П о с п е л о в Д.А.: Логические методы анализа и синтеза схем. Энергия, Москва 1974.
- [6] Traczyk W.: Układy cyfrowe, podstawy teoretyczne i metody syntezy. WNT, Warszawa 1982.