МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ДНІПРОВСЬКА ПОЛІТЕХНІКА»

Факультет інформаційних технологій
Кафедра інформаційних технологій та
комп’ютерної інженерії

В.І. Олевський, Ю.Б. Олевська, Н.О. Соколова

Комп’ютерна логіка

Конспект лекцій

Дніпро
НТУ «ДП»
2024

Олевський В.І. Конспект лекцій з дисципліни «Комп’ютерна логіка» для здобувачів ступеня бакалавра спеціальності 123 Комп’ютерна інженерія / В.І. Олевський, Ю.Б. Олевська, Н.О. Соколова ; М-во освіти і науки України, Нац. техн. ун-т «Дніпровська політехніка». – Електрон. дані. – Дніпро : НТУ «ДП», 2024. – 384 с.

Автори:
В.І. Олевський, д-р. техн. наук, проф. каф. ІТКІ
Ю.Б. Олевська, канд. фіз.-мат. наук, доц. каф. ПМ
Н.О. Соколова, канд. техн. наук, доц. каф. ІТКІ

Погоджено науково-методичною комісією спеціальності 123 Комп’ютерна інженерія (протокол № 2 від 02.02.2024) за поданням кафедри інформаційних технологій та комп’ютерної інженерії (протокол № 11 від 01.02.2024).

Розглянуто прикладні питання теорії цифрових автоматів, методи синтезу логічних схем в заданому елементом базисі та їх дослідження. Розкрито теми «Мінімізація логічних функцій», «Абстрактна теорія цифрових автоматів», «Структурний синтез цифрових автоматів», «Комп’ютерна арифметика», розглянуто приклади по кожній з тем, надано перелік використаних і рекомендованих джерел.

Відповідальний за випуск професор кафедри ІТКІ В.І. Олевський, д-р. техн. наук, проф.

ЗМІСТ

- **Лекція 7.** Структурний синтез цифрових автоматів. Типові комбінаційні схеми комп'ютерів. Суматори. Дешифратори та шифратори. Мультиплексори та демультиплексори. Регістри. Двійковий лічильник.
- **Лекція 8.** Етапи синтезу автоматів. Структурний синтез цифрових автоматів. Канонічний метод структурного синтезу.
- **Лекція 9.** Комп′ютерна арифметика. Загальні поняття про системи числення. Переведення чисел з одних систем числення в інші. Подання чисел в засобах обчислювальної техніки.
- **Лекція 10.** Комп′ютерна арифметика. Способи кодування двійкових чисел. Алгоритми виконання арифметичних операцій у цілочисельних операційних пристроях, частина I. Алгоритми виконання операцій додавання і віднімання. Алгоритми множення кодів чисел з фіксованою комою. Множення чисел із знаком. Множене довільного знаку, множник додатний.
- **Лекція 11.** Алгоритми виконання арифметичних операцій у цілочисельних операційних пристроях, частина II. Прискорення цілочисельного множення. Алгоритми ділення кодів чисел з фіксованою комою. Алгоритм ділення з нерухомим дільником і відновленням залишку та без відновлення залишку. Ділення чисел із знаком. Пристрій ділення. Прискорення цілочисельного ділення. Виконання арифметичних операцій з плаваючою комою. Додавання, віднімання, множення, ділення.

ВСТУП

Мета дисципліни – формування компетентностей, що пов’язані з використанням інформаційних технологій піл час аналізу і синтезу цифрових автоматів комп'ютерних систем та мереж відповідно до освітньо-професійної програми.

ОЧІКУВАНІ ДИСЦИПЛІНАРНІ РЕЗУЛЬТАТИ НАВЧАННЯ

- ❖Вміти розв’язувати задачі аналізу та синтезу цифрових автоматів та систем обчислень у комп’ютерах.
- ❖Вміти ідентифікувати, класифікувати, та описувати роботу комп’ютерів та цифрових автоматів загального виду.

Лекція 1

КОМП’ЮТЕРНА ЛОГІКА ТА ТЕОРІЯ ЦИФРОВИХ АВТОМАТІВ

ОСНОВНІ ПОНЯТТЯ АЛГЕБРИ ЛОГІКИ

КОМП'ЮТЕРНА ЛОГІКА ТА ТЕОРІЯ ЦИФРОВИХ АВТОМАТІВ

Комп'ютерна логіка – це навчальна дисципліна, що охоплює логічні, математичні та технічні основи, базові принципи, поняття та моделі обчислювальних та управляючих цифрових систем.

Фундаментальною логічною моделлю комп'ютерної системи є поняття автомату.

1. Автомат – це **пристрій (система)**, який без участі людини здійснює відбір, опрацювання, передавання та зберігання інформації (даних, повідомлень, сигналів) відповідно до закладеногоу нього алгоритму.

2. Автомат – це **математична модель реальної технічної системи** (наприклад, комп'ютерачи певного його вузла, елемента). У цьому випадку автомат розглядається як деяка «чорна скринька», що має

скінченну кількість входів та виходів, а також деяку множину внутрішніх станів.

Теорія цифрових автоматів вивчає математичні моделі перетворювачів (автоматів) дискретної інформації, яка подана у цифровій формі. Такими перетворювачами є як реальні пристрої (обчислювальні машини, живі організми), так й абстрактні системи (наприклад, *формальна система* – це сукупність абстрактних об'єктів, не пов'язаних із зовнішнім світом, в яких подано правила оперування множиною символів у строго синтаксичному трактуванні без урахування смислового навантаження, тобто семантики).

Практично, *автомат* – це деякий пристрій, призначений для перетворення інформації. Автомат можна вважати **інформаційною системою**. Термін «інформація» має багато означень.

У широкому сенсі *інформація* – відображення реального світу.

У вузькому сенсі: *інформація* – будь-які відомості, що є об'єктом відбору, передавання,перетвореннята зберігання.

Важливе питання теорії інформації – **встановлення міри, кількості та якості інформації**. Інформаційні міри, переважно, розглядаються у трьох аспектах: **структурному, статистичному та семантичному**.

У ***структурному*** *аспекті* розглядається будова масивів інформації та їх зміна простим підрахунком інформаційних елементів або комбінаторним методом.

Структурний підхід застосовується для оцінювання можливостей інформаційних систем незалежно від умов їх використання.

При ***статистичному*** *підход і*використовується поняття ентропії як міри використовується поняття ентропії як міри невизначеності, що враховує ймовірність появи того чи іншого повідомлення.

*Семантичний підхід*даєзмогу виділити корисність або цінність інформаційного повідомлення.

Інформація надходить у систему передавання даних у формі **повідомлень**. Під *повідомленням* розуміють сукупність знаків або первинних сигналів, що містять інформацію.

Дискретні *повідомлення* формуються в результаті послідовного видавання джерелом повідомлень окремих елементів – *знаків, символів, букв*. Множину різних знаків називають ***алфавітом*** **джерела повідомлення**, а число знаків – ***об'ємом алфавіту***. Якщо повідомленняє дискретними, то пристрій обробки такої інформації називають дискретним пристроєм або ***дискретним автоматом***.

Неперервні повідомлення не поділені на елементи. Вони описуються **функціями часу**, що означені на континуальних множинах.

Для передавання повідомлення каналом зв'язку йому ставлять у відповідність певний **сигнал**. Під *сигналом* розуміють фізичний процес, що відображає, переносить повідомлення.

Перетворення повідомлення в сигнал, зручний для передавання каналом зв'язку, називають ***кодуванням***. Операцію відновлення повідомлення по прийнятому сигналу називають ***декодуванням***.

Прийнято ототожнювати букви довільних алфавітів з цифрами. Закодовані таким чином сигнали називають ***цифровими сигналами***. Тому дискретні автомати називають ***цифровими автоматами***.

На практиці вдаються до операції представлення початкових знаків в іншому алфавіті з меншим числом знаків. Пристрій, що виконує таку операцію, називають таким, що кодує, або ***кодером*.** Кожному знаку відповідає деяка послідовність символів, яку називають ***кодовою комбінацією***. Кількість символів у кодовій комбінації називають її ***значущістю***, кількість ненульових символів – ***вагою***.

Зіставлення символів зі знаками початкового алфавіту - "декодування". Технічна реалізація цієї операції здійснюється декодуючим пристроєм, або ***декодером***.

Найтісніше теорія автоматів пов'язана з ***теорією алгоритмів***.

Автомат називають ***абстрактним***, коли абстрагуються від реальних фізичних вхідних та вихідних сигналів, розглядаючи їх просто як букви деякого алфавіту.

Скінченний автомат – математична абстракція, що дає змогу описувати шляхи зміни стану об'єкта залежно від його **поточного стану та вхідних даних**, за умови, що загальна можлива кількість станів та множина вхідних сигналів **скінченні**. Скінченний автомат є частковим випадком абстрактного автомата.

Ще однією особливістю цифрових автоматів є властивість ***стрибкоподібного переходу*** автомата зодного стану в інший. Такий перехід можна вважати миттєвим, хоча має місце скінченна тривалість перехідних процесів. Інше припущення, яке стосується роботи цифрового автомата, полягає в тому, що після переходу автомата в довільний стан перехід унаступний стан може бути здійсненимне раніше, ніж через деякий фіксований проміжок часу $\Delta t > 0$, якийназивають ***інтервалом дискретності автомата***. Таке припущення даєзмогу розглядати функціонування цифрового автомата в *дискретному часі*. При побудові автоматів з дискретним автоматним часом розрізняють синхронні та асинхронні автомати.

У *синхронних автоматах* моменти часу, в які необхідна зміна стану автомата, визначаються спеціальним пристроєм – **генератором синхронізуючих імпульсів** з . рівними часовими проміжками.

В *асинхронних автоматах* моменти переходів з одного стану в інший заздалегідь не визначаються та можуть здійснюватися через нерівні між собою проміжки часу.

Історично першими дискретними схемами автоматики були схеми на основі електромеханічних контактних апаратів (реле, контакторів, магнітних пускачів та ін.). Потім почали використовувати безконтактні релейні елементи, які отримали назву **логічних елементів**. У цих елементах використовували транзистори, діоди, резистори, магнітні осердя з обмотками тощо.

Розроблення дискретних схем промислової автоматики починається з визначення **алгебричних виразів (логічних формул)**, що описують роботу схеми (логічного синтезу). В результаті логічного синтезу за формулюванням умов роботи схеми мають бути визначені логічні формули, що описують її роботу, яка задовольняє усі задані умови автоматизації, особливості технологічного процесу та керованого об’єкта. Логічні формули є основою для побудови схем з жорсткою логікою або для складання програм роботи логічних програмованих контролерів у схемах з гнучкою логікою.

Математичною основою аналізу і синтезу дискретних схем автоматики є двозначна **алгебра логіки**.

ОСНОВНІ ПОНЯТТЯ АЛГЕБРИ ЛОГІКИ

В алгебрі логіки, основними поняттями є *логічна* (*булева*) *змінна* та *логічна функція* (*функція алгебри логіки*).

Логічна* (*булева*) *змінна –величина *x*, яка може приймати тільки два значення 0 та 1. Значення 0 та 1 називають *булевими константами*.

Логічна функція (*функція алгебри логіки*) – функція $f(x_1,\ldots,x_n)$, область значень якої є двоелементною множиною $\{0,1\}$ та яка залежать від булевих змінних $x_1,\ldots,x_n$, що набувають значення із цієї ж двоелементної множини. Логічні функції також називають *функціями перемикання* або ***перемикальними функціями***.

Логічні функції можуть бути задані трьома способами:

1. За допомогою *таблиці істинності* – таблиці, в якій кожній інтерпретації (набору аргументів) функції поставлено у відповідність її значення.

2. *Порядковим номером* логічної функції. Кожній логічній функції ставиться у відповідність її порядковий номер у вигляді натурального числа, двійковий код якого зображує стовпчик значень функції у таблиці істинності.

3. *Аналітично* (у вигляді формули). *Формула* – це вираз, що містить булеві функції та їхні суперпозиції.

Верхньому рядку таблиці істинності ставиться у відповідність номер 0, потім йде інтерпретація 1 тощо. У найнижчому рядку – $2^n - 1$, де *n* – кількість змінних, від яких залежить булева функція.

Формули, що зображують одну й ту ж функцію, називають *еквівалентними* або *рівносильними* (позначається знаком «=»).

Окрім *еквівалентності* «=» в алгебрі логіки означено три операції:

- диз'юнкція (логічне додавання, функція АБО), яка позначається знаком «+» або «$\cup$»,
- кон'юнкція (логічне множення, функція І), яка позначається крапкою, яку можна опускати, а також знаками «$\cap$» або «&»,
- інверсія (заперечення, функція НІ), що позначається рискою над логічними змінними або над константами 0 та 1.

Операція еквівалентності задовольняє такі властивості:

- $x=x$ – рефлексивність;

- якщо $y=x$, то $x=y$ – симетричність;

- якщо $y=x$ та $z=y$, то $z=x$ – транзитивність.

Алгебра логіки визначається такою системою аксіом

$$\begin{cases} x=0, & якщо \quad x\neq 1, \\ x=1, & якщо \quad x\neq 0. \end{cases} \qquad \begin{cases} 0+0=0, \\ 1\cdot 1=1. \end{cases} \qquad \begin{cases} 1+1=1, \\ 0\cdot 0=0. \end{cases}$$

$$\begin{cases} 0+1=1+0=1, \\ 1\cdot 0=0\cdot 1=0. \end{cases} \qquad \begin{cases} \overline{0}=1, \\ \overline{1}=0. \end{cases}$$

Для алгебри логіки особливе значення мають логічні функції від двох змінних. Задання логічних функцій лише для двох змінних дає змогу визначити усі логічні функції для будь-якої кількості аргументів.

Аналітичний запис	Назва	x_1x_2				Функція
		00	01	10	11	
$y_0 = 0$	константа 0	0	0	0	0	y_0
$y_1 = x_1 \cdot x_2$	кон'юнкція	0	0	0	1	y_1
$y_2 = x_1 \Delta x_2 = x_1 \cdot \bar{x}_2$	заборона за x_2	0	0	1	0	y_2
$y_3 = x_1 = x_1\bar{x}_2 + x_1x_2$	змінна x_1	0	0	1	1	y_3
$y_4 = x_2 \Delta x_1 = \bar{x}_1 \cdot x_2$	заборона за x_1	0	1	0	0	y_4
$y_5 = x_2 = \bar{x}_1x_2 + x_1x_2$	змінна x_2	0	1	0	1	y_5
$y_6 = x_1 \oplus x_2$	сума за модулем 2 (нееквівалентність)	0	1	1	0	y_6
$y_7 = x_1 + x_2$	диз'юнкція	0	1	1	1	y_7
$y_8 = x_1 \downarrow x_2 = \overline{x_1 + x_2}$	стрілка Пірса (заперечення диз'юнкції)	1	0	0	0	y_8
$y_9 = x_1 \equiv x_2$	еквівалентність	1	0	0	1	y_9
$y_{10} = \bar{x}_2 = \bar{x}_1\bar{x}_2 + x_1\bar{x}_2$	інверсія x_2	1	0	1	0	y_{10}
$y_{11} = x_2 \rightarrow x_1 = x_1 \cdot \bar{x}_2$	імплікація від x_2 до x_1	1	0	1	1	y_{11}
$y_{12} = \bar{x}_1 = \bar{x}_1\bar{x}_2 + \bar{x}_1x_2$	інверсія x_1	1	1	0	0	y_{12}
$y_{13} = x_1 \rightarrow x_2 = \bar{x}_1 \cdot x_2$	імплікація від x_1 до x_2	1	1	0	1	y_{13}
$y_{14} = x_1 / x_2 = \overline{x_1x_2}$	штрих Шеффера (заперечення кон'юнкції)	1	1	1	0	y_{14}
$y_{15} = 1$	константа 1	1	1	1	1	y_{15}

Конституентою одиниці називається логічна функція *n* змінних, яка дорівнює одиниці тільки для якого-небудь одного набору аргументів і дорівнює нулеві для решти 2^n–1 наборів. Конституенту одиниці називають також **мінтермом**.

Конституентою нуля називається логічна функція *n* змінних, яка дорівнює нулеві тільки для якого-небудь одного набору аргументів і одиниці для решти 2^n–1 наборів. Конституенту нуля називають також **макстермом**.

Частинними випадками конституент є логічні функції – кон'юнкція та диз'юнкція.

Назва закону	Формули
Закони ідемпотентності	$x + x = x$ $x \cdot x = x$
Закони комутативності	$x + y = y + x$ $x \cdot y = y \cdot x$
Закони асоціативності	$(x + y) + z = x + (y + z)$ $(x \cdot y) \cdot z = x \cdot (y \cdot z)$
Закон дистрибутивності для кон'юнкції відносно диз'юнкції	$x \cdot (y + z) = x \cdot y + x \cdot z$
Закон дистрибутивності для диз'юнкції відносно кон'юнкції	$x + y \cdot z = (x + y) \cdot (x + z)$
Закон виключення третього	$x + \bar{x} = 1$
Закон протиріччя	$x \cdot \bar{x} = 0$
Співвідношення для констант	$0 + x = x$ $1 \cdot x = x$ $1 + x = 1$ $0 \cdot x = 0$
Закони подвійності (теореми де Моргана)	$\overline{x + y} = \bar{x} \cdot \bar{y}$ $\overline{x \cdot y} = \bar{x} + \bar{y}$
Закон подвійного заперечення	$\overline{(\bar{x})} = \bar{\bar{x}} = x$
Закони поглинання	$x + x \cdot y = x$ $x \cdot (x + y) = x$
Закон склеювання	$x \cdot y + x \cdot \bar{y} = x$ $(x + y) \cdot (x + \bar{y}) = x$

Ці закони дають змогу перетворювати функції до вигляду, що не містить однієї з операцій – АБО або І. Це буває потрібним для побудови схем на логічних елементах, що реалізують операції АБО–НІ або І–НІ. Наприклад, функцію

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

можна перетворити до вигляду, що не містить операції АБО, якщо взяти подвійну інверсію, а потім розкрити одну з інверсій за правилом де Моргана, тобто

$$f = \overline{\overline{a\bar{b} + \bar{b}\,\bar{c} + cd}} = \overline{\overline{a\bar{b}} \cdot \overline{\bar{b}\,\bar{c}} \cdot \overline{cd}}\,.$$

Користуючись отриманою формулою, можна побудувати схему на елементах І–НІ.

Аналогічним способом можна усунути операцію І. Наприклад,

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

За цією формулою можна побудувати схему на елементах АБО–НІ.

Функції кон'юнкція, диз'юнкція, інверсія (І, АБО, НІ) – основні й найбільш уживані логічні функції.

Узагальнення законів де Моргана, запропоноване Шенноном

$$\bar{f}(a, b, c, \ldots, w, \cdot, +) = f(\bar{a}, \bar{b}, \bar{c}, \ldots, \bar{w}, +, \cdot),$$

тобто для отримання інверсії функції потрібно замінити всі аргументи на їх інверсії, всі операції І на операції АБО, а операції АБО на операції І.

Нормальні та довершені нормальні форми логічних функцій

Елементарна кон'юнкція (диз'юнкція) – кон'юнкція (диз'юнкція) кількох аргументів, кожний з яких може одноразово входити до неї зі знаком інверсії або без нього (*abc, abc, ad , ac, abd* — елементарні кон'юнкції; *a+b+c, a+c, a+d+b, a+b* — елементарні диз'юнкції).

Диз'юнкція кількох елементарних кон'юнкцій називається ***диз'юнктивною нормальною формою*** **(ДНФ).**

Аналогічно, кон'юнкція кількох елементарних диз'юнкцій називається ***кон'юнктивною нормальною формою*** **(КНФ)** (*ab+abc+bcd* – ДНФ, (*a+b*)(*a+b+c*) – КНФ. Кількість аргументів в елементарній кон'юнкції (диз'юнкції) називається її довжиною і визначає її **ранг**. *abc* – кон'юнкція третього рангу, *a+b+c+d* – диз'юнкція четвертого рангу.

Якщо кожний член диз'юнктивної нормальної форми містить усі *n* аргументів, то форма називається ***довершеною*** **(ДДНФ)**.

Отже, ДДНФ – це диз'юнкція конституент одиниці. Оскільки конституента одиниці повністю визначається єдиним набором аргументів, що перетворює її в одиницю, то кожний кон'юнктивний член ДДНФ перетворюється в одиницю за деякого єдиного набору аргументів, а загальна кількість кон'юнктивних членів у ДДНФ дорівнює кількості наборів, що перетворюють функцію в одиницю.

Лекція 2

МІНІМІЗАЦІЯ ЛОГІЧНИХ ФУНКЦІЙ

МІНІМІЗАЦІЯ ЛОГІЧНИХ ФУНКЦІЙ

Нормальні та довершені нормальні форми логічних функцій

Елементарна кон'юнкція (диз'юнкція) – кон'юнкція (диз'юнкція) кількох аргументів, кожний з яких може одноразово входити до неї зі знаком інверсії або без нього (*abc, abc, ad , ac, abd* — елементарні кон'юнкції; *a+b+c*, *a+c*, *a+d+b*, *a+b* — елементарні диз'юнкції).

Диз'юнкція кількох елементарних кон'юнкцій називається ***диз'юнктивною нормальною формою*** **(ДНФ).**

Аналогічно, кон'юнкція кількох елементарних диз'юнкцій називається ***кон'юнктивною нормальною формою*** **(КНФ)** (*ab+abc+bcd* – ДНФ, (*a+b*)(*a+b+c*) – КНФ. Кількість аргументів в елементарній кон'юнкції

(диз'юнкції) називається її довжиною і визначає її **ранг**. *abc* – кон'юнкція третього рангу, *a*+*b*+*c*+*d* – диз'юнкція четвертого рангу.

Якщо кожний член диз'юнктивної нормальної форми містить усі *n* аргументів, то форма називається ***довершеною*** **(ДДНФ)**.

Отже, ДДНФ – це диз'юнкція конституент одиниці. Оскільки конституента одиниці повністю визначається єдиним набором аргументів, що перетворює її в одиницю, то кожний кон'юнктивний член ДДНФ перетворюється в одиницю за деякого єдиного набору аргументів, а загальна кількість кон'юнктивних членів у ДДНФ дорівнює кількості наборів, що перетворюють функцію в одиницю.

Логічна функція відображає роботу реального технічного пристрою. Складність функції деякою мірою визначає складність пристрою або складність програми для логічного програмованого контролера. Рівносильним логічним функціям відповідають схеми, що виконують однакові завдання. Отже, однакові завдання можна виконувати за схемами різної складності. Тому виникає прагнення у результаті рівносильних перетворень так видозмінити функцію, щоб отримати найпростішу схему. Іноді визначальними є надійність схеми, вартість, маса або інші показники.

Загалом завдання мінімізації можна сформулювати так. Відомі:

1) логічна функція або функції, що реалізуються;

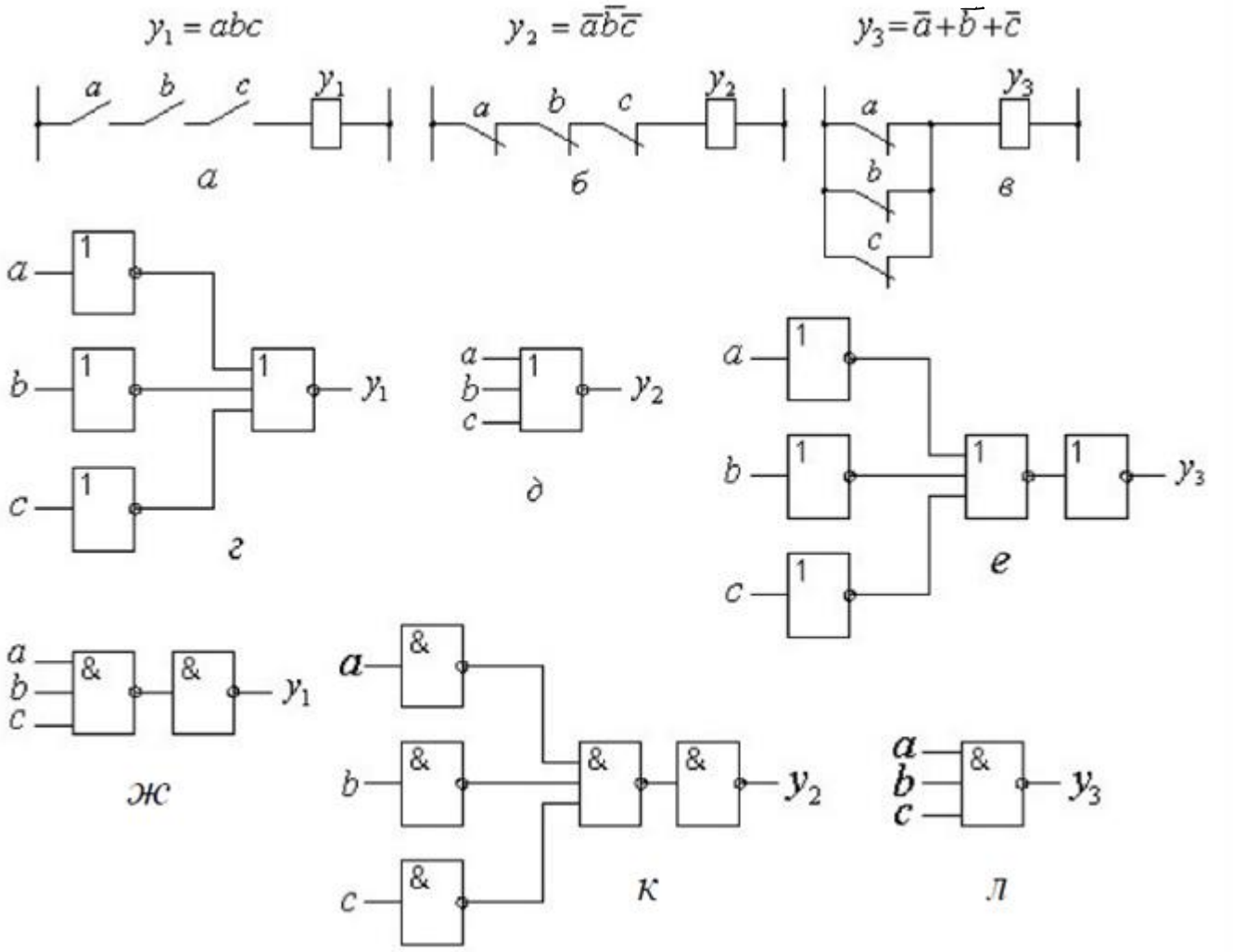
2) серія мікросхем, на яких передбачається побудувати пристрій, що реалізує задані функції;

3) характеристика серії – набір функцій, що реалізуються елементами серії, навантажувальна здатність елементів, кількість входів, правила використання, вартість;

4) критерії мінімізації: мінімум вартості, мінімум маси, максимальна швидкодія, мінімальна витрата елементів тощо.

Потрібно знайти алгебричний вираз функції, що задовольняє заданий критерій мінімізації.

У загальному випадку проблему мінімізації натепер не розв’язано, тобто немає загальних методів, які дозволяють стандартними способами визначити вигляд формул, що задовольняють той або інший критерій мінімізації. Найбільш повно розроблено способи мінімізації, що забезпечують отримання формул мінімальної довжини, тобто формул, до складу яких входить мінімальна кількість букв. Таким формулам відповідають релейно-контактні схеми з мінімальною кількістю контактів. У цьому разі завдання мінімізації розв’язують так: спочатку одним з методів відшукують мінімальну диз’юнктивну або кон’юнктивну нормальні форми функції, а потім зменшують кількість букв у формулі за допомогою дужкових форм.



Схеми на елементах

АБО-НІ

$$y_1 = \overline{\overline{abc}} = \overline{\bar{a} + \bar{b} + \bar{c}};$$

$$y_2 = \overline{\overline{\bar{a}\,\bar{b}\,\bar{c}}} = \overline{a + b + c};$$

$$y_3 = \overline{\overline{\bar{a} + \bar{b} + c}},$$

на елементах І–НІ

$$y_1 = \overline{\overline{abc}};$$

$$y_2 = \overline{\overline{a\,\bar{b}\,\bar{c}}};$$

$$y_3 = \overline{\overline{\bar{a} + \bar{b} + c}} = \overline{abc}.$$

Мінімізація безпосереднім застосуванням законів алгебри логіки

$f = abd + a\bar{b} + a\bar{d}$. $\bar{b} + bd = \bar{b} + d$

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

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

$$af(a,\bar{a},b,c,\ldots,w) = af(1,0,b,c,\ldots,w);$$

$$\bar{a}f(a,\bar{a},b,c,\ldots,w) = \bar{a}f(0,1,b,c,\ldots,w);$$

$$a + f(a,\bar{a},b,c,\ldots,w) = a + f(0,1,b,c,\ldots,w);$$

$$\bar{a} + f(a,\bar{a},b,c,\ldots,w) = \bar{a} + f(1,0,b,c,\ldots,w).$$

2. $f = ad + ab + \bar{a}c + \bar{b}cd$.

За законом узагальненого склеювання

$ad + \bar{a}c = ad + \bar{a}c + cd$, $f = ad + ab + \bar{a}c + cd(1 + \bar{b}) = ad + ab + \bar{a}c + cd$.

$ad + \bar{a}c + cd = ad + \bar{a}c$. $f = ad + ab + \bar{a}c$.

Методи Квайна і Мак-Класкі

Зі збільшенням кількості аргументів функцій використовують алгоритмічні методи. Якщо функцію подано в нормальній диз'юнктивній формі, то методом її спрощення є використання законів поглинання і склеювання.

Склеювання	$x \cdot y + x \cdot \overline{y} = x$	$(x + y)(x + \overline{y}) = x$
Поглинання	$x + x \cdot y = x$	$x \cdot (x + y) = x$

Послідовно здійснюючи всі можливі склеювання та поглинання, можна отримати **тупикову** ДНФ функції, що не містить жодної пари елементарних кон'юнкцій, до яких можна застосувати закони склеювання та поглинання.

Логічні функції можуть мати кілька тупикових форм. Мінімальна форма функції є однією з тупикових, тому її можна вибрати, якщо знайти всі тупикові форми вихідної функції.

Одним з методів, який дозволяє визначити всі тупикові форми функції, є метод Квайна. Мінімізувати функцію за методом Квайна можна в кілька етапів.

1. Функцію, подану у довільній формі, розгортають у ДДНФ.

2. Відшукують усі первинні імпліканти логічної функції.

Логічна функція φ називається **імплікантою** логічної функції f у тому разі, якщо вона дорівнює нулеві за будь-якого набору аргументів, для якого функція f також дорівнює нулеві. Набори, перетворюючи функцію f в одиницю, можуть перетворювати імпліканту φ як в одиницю, так і в нуль.

Набори, що перетворюють імпліканту φ в одиницю, мають перетворювати в одиницю також і функцію f. Якщо для деяких наборів аргументів імпліканти φ функція f перетворюється в одиницю, то вважають, то імпліканта φ покриває функцію f на цих наборах.

Будь-яка елементарна кон'юнкція, що входить до складу ДНФ функції, або група елементарних кон'юнкцій, з'єднаних знаком диз'юнкції, є імплікантою вихідної ДНФ функції.

Первинною імплікантою функції називається імпліканта у вигляді елементарної кон'юнкції деяких аргументів, жодна частина якої вже не є імплікантою.

Для відшукування первинних імплікант виписують у стовпець усі конституенти одиниці, які входять до складу ДДНФ функції. Потім кожну конституенту одиниці порівнюють з рештою конституент. Якщо будь-які дві конституенти одиниці відрізняються одна від одної тільки однією змінною, то виписують їх загальну частину і проти кожної з цих конституент проставляють будь–який знак. Заміна двох конституент одиниці їх загальною частиною еквівалентна застосуванню закону склеювання. Конституенти, які беруть участь хоча б в одному склеюванні, позначають, але не вилучають з подальших порівнянь. Усі непозначені елементарні кон’юнкції являють собою первинні імпліканти. Первинною імплікантою може бути також конституента одиниці, до якої не могла бути застосованою операція склеювання з будь-якої іншою конституентою.

Будь-яку елементарну кон’юнкцію, як позначену, так і не позначену, можна включити в остаточний вираз вихідної функції. Більш того, виявляється, що диз’юнкція усіх непозначених елементарних кон’юнкцій (усіх первинних імплікант) являє собою вихідну функцію. Такий висновок випливає з того, що після виконання другого етапу щоразу з двох позначених елементарних кон’юнкцій виходила одна, еквівалентна їм. Отже, непозначені первинні імпліканти після закінчення другого етапу еквівалентні усім вихідним конституентам одиниці. Проте диз’юнкція усіх первинних імплікант у загальному випадку має збитковість, тобто її склад містить більше елементарних кон’юнкцій, ніж це треба для отримання заданої функції. Усі подальші етапи мінімізації – це вибір тих первинних імплікант із знайдених на другому етапі, диз’юнкція яких дає мінімальну форму функції.

3. Складають таблицю А, кількість рядків якої дорівнює кількості первинних імплікант, а кількість стовпців – кількості конституент одиниці у ДДНФ вихідної функції. Кожний рядок таблиці озаглавлюють первинною імплікантою, а кожний стовпець – конституентою одиниці. Ставлять хрестики у тих місцях таблиці, де первинна імпліканта, що стоїть ліворуч, є складовою частиною конституенти одиниці стовпця.

4. Аналізують стовпці таблиці А. Якщо у будь-якому стовпці є тільки один хрестик, то відповідна йому первинна імпліканта обов’язково ввійде до остаточного виразу функції. Усі хрестики в рядках таких імплікант обводять кружками. Після цього складають нову таблицю Б, аналогічну таблиці А, але яка не містить тих первинних імплікантів і конституент одиниці в рядках і стовпцях яких обведені хрестики.

5. Вилучаючи з таблиці Б стовпці, вилучені на п’ятому етапі, та рядки, у яких взагалі немає хрестиків, складають нову таблицю В. Вилучення рядків, які не мають хрестиків, можливе тому, що всі конституенти одиниці, які можуть бути покритими первинною імплікантою, вже покриваються іншими первинними імплікантами, які мають обов’язково увійти в остаточний вираз функції.

6. Аналізуючи таблицю В, обирають мінімальну групу (або групи) первинних імплікант, яка має хоча б по одному хрестику у кожному стовпці. Диз’юнкція такої групи первинних імплікант і первинних імплікант, знайдених на третьому етапі, дає мінімальну форму функції f .

$$f = \bar{a}\bar{b}c\bar{d} + \bar{a}\bar{b}\bar{c}\bar{d} + \bar{a}bc\bar{d} + \bar{a}bcd + \bar{a}b\bar{c}d + abcd + ab\bar{c}d + a\bar{b}\bar{c}d + a\bar{b}\bar{c}\bar{d}$$

Функцію f подано в ДДНФ, тому мінімізацію починаємо з другого етапу

1) $\bar{a}\bar{b}c\bar{d}$ ∨
2) $\bar{a}\bar{b}\bar{c}\bar{d}$ ∨
3) $\bar{a}bc\bar{d}$ ∨
4) $\bar{a}bcd$ ∨
5) $\bar{a}b\bar{c}d$ ∨
6) $abcd$ ∨
7) $ab\bar{c}d$ ∨
8) $a\bar{b}\bar{c}d$ ∨
9) $a\bar{b}\bar{c}\bar{d}$ ∨

1) $\bar{a}\bar{b}\,\bar{d}$
2) $\bar{a}c\bar{d}$
3) $\bar{b}\bar{c}\bar{d}$
4) $\bar{a}bc$
5) $\bar{a}bd$ ∨
6) bcd ∨
7) $b\bar{c}d$ ∨
8) abd ∨
9) $a\bar{c}d$
10) $a\bar{b}\bar{c}$

1) bd

Первинні імпліканти	Конституенти одиниці								
	$\bar{a}\bar{b}cd$	$\bar{a}\bar{b}\bar{c}d$	$\bar{a}bcd$	$\bar{a}bcd$	$\bar{a}b\bar{c}d$	$abcd$	$ab\bar{c}d$	$a\bar{b}\bar{c}d$	$a\bar{b}\bar{c}\bar{d}$
$\bar{a}\bar{b}\,\bar{d}$	+	+							
$\bar{a}c\bar{d}$	+		+						
$\bar{b}\bar{c}\bar{d}$		+							+
$\bar{a}bc$			+	+					
$a\bar{c}d$							+	+	
$a\bar{b}\bar{c}$								+	+
bd				⊕	⊕	⊕	⊕		

Первинні імпліканти	Конституенти одиниці				
	$\bar{a}\bar{b}cd$	$\bar{a}\bar{b}\bar{c}d$	$\bar{a}bcd$	$a\bar{b}\bar{c}d$	$a\bar{b}\bar{c}\bar{d}$
$\bar{a}\bar{b}\,\bar{d}$	+	+			
$\bar{a}c\bar{d}$	+		+		
$\bar{b}\bar{c}\bar{d}$		+			+
$\bar{a}bc$			+		
$a\bar{c}d$				+	
$a\bar{b}\bar{c}$				+	+

$$f_1 = bd + \bar{a}\bar{b}\,\bar{d} + \bar{a}bc + a\bar{b}\,\bar{c};$$
$$f_2 = bd + \bar{a}c\bar{d} + \bar{b}\,\bar{c}\,\bar{d} + a\bar{b}\bar{c};$$
$$f_3 = bd + \bar{a}c\bar{d} + \bar{b}\,\bar{c}\,\bar{d} + a\bar{c}d;$$
$$f_4 = bd + \bar{a}\bar{b}\,\bar{d} + \bar{a}c\bar{d} + a\bar{b}\,\bar{c}.$$

Для мінімізації функцій методом Квайна найбільш трудомісткий процес, який може бути причиною помилок, є вичерпне порівняння усіх пар кон'юнкцій для склеювання. Порівняння елементарних кон'юнкцій виконувати набагато простіше, якщо їх подати у вигляді двійкових чисел (двійкових еквівалентів), причому процедуру порівняння можна виконувати таким чином, що кількість потрібних порівнянь суттєво скорочується. Метод Квайна з таким удосконаленням називають методом Мак-Класкі.

Мінімізація функцій алгебри логіки методом Квайна-Мак-Класкі.

Даний метод базується на задаванні функцій елементарних змінних, що входять в ДДНФ у виді двійкових чисел, які називаються **номерами** відповідних наборів. Крім номера кожному добуткові присвоюється визначений **індекс**, під яким розуміють кількість одиниць у двійковому поданні даного набору. Наприклад:

Набір $\overline{X1} \cdot X2 \cdot \overline{X3}$, *номер* 010 (2), *індекс* 1(I)

Набір $X1 \cdot X2 \cdot \overline{X3}$, *номер* 110 (6), *індекс* 2(II)

В результаті реалізації даного методу функція алгебри логіки розкладається на прості імпліканти. Під простою імплікантою функції розуміють будь-який елементарний добуток, що приймає одиничне значення на всіх наборах аргументів, що і вхідна ФАЛ і при виключенні з якого хоча б одного аргументу, вже не виконуватиметься дана умова.

Алгоритм Квайна-Мак-Класкі формулюється наступним чином: для того, щоб два числа m та n були номерами двох наборів, що склеюються між собою, необхідно і достатньо, щоб індекси даних чисел відрізнялись на одиницю, самі числа відрізнялись на степінь числа два і число з більшим індексом було більше за число з меншим індексом.

Реалізацію алгоритму розглянемо на прикладі мінімізації фунукії:

$$Y = \overline{X1} \cdot \overline{X2} \cdot \overline{X3} \cdot X4 \vee \overline{X1} \cdot X2 \cdot \overline{X3} \cdot X4 \vee X1 \cdot \overline{X2} \cdot X3 \cdot \overline{X4} \vee \overline{X1} \cdot X2 \cdot X3 \cdot X4 \vee X1 \cdot \overline{X2} \cdot X3 \cdot X4 \vee \overline{X1} \cdot \overline{X2} \cdot X3 \cdot X4$$

На першому етапі мінімізації визначаємо номери та індекси кожного набору, записуючи функцію у виді:

$$f = 0001 \vee 0101 \vee 1010 \vee 0111 \vee 1011 \vee 0011$$

	1_{I}	5_{II}	10_{II}	7_{III}	11_{III}	3_{II}

Групуємо набори і розміщуємо їх в порядку зростання індексів.

Таблиця 1

Індекси	Номери	Результат склеювання	
I	0001 (1)	00-1 (1, 3) 0-01 (1, 5)	0--1 (1, 3, 5, 7) 0--1 (1, 5, 3, 7)
II	0011 (3) 0101 (5) 1010 (10)	0-11 (3, 7) -011 (3, 11) 01-1 (5, 7)	
III	0111 (7) 1011 (11)	101- (10, 11)	

На наступному етапі поводимо склеювання різних наборів, керуючись наведеним формулюванням алгоритму. При склеюванні розряди чисел, що не співпадають відмічаються почерками. Наприклад, склеювання чисел 0001 та 0011 дає число 00-1. Результат склеювання виписується в наступний стовпець таблиці, який так само розділюється на стрічки з індексами, які відрізняються на одиницю. Після склеювання всіх груп першого стовпця таблиці переходять до другого, переписуючи результат склеювання в третій стовпець. При об’єднанні наборів третього і наступних стовпців таблиці можна склеювати тільки сила, що містять прочерки в однойменних розрядах. Склеювання продовжується до тих пір, поки неможливо буде створити новий стовпець.

Після завершення склеювання приступають до побудови імплікантної таблиці, записуючи в неї в якості простих імплікант набори, що містяться в останньому стовпці табл.1. В якості простих імплікант в табл.2 також вписуються набори з тих стовпців табл.1, котрі не приймали участі у склеюванні. Якщо імпліканта, яка міститься в *i*-й лінійці таблиці складає деяку частину конституанти *i*-го стовпця на перетині *i*-тої лінійки та *i*-го стовпця ставиться символ “*”. З метою отримання мінімальної форми ФАЛ із табл.2 необхідно вибрати мінімальну кількість лінійок, щоб для кожного стовпця серед вибраних лінійок знайшлася хоча б одна, що містить в цьому стовпці символ “*”.

Таблиця 5 Імплікантна таблиця мінімізованої ФАЛ

Набори	1	5	10	7	11	3
Імпліканти	$\overline{X1}\cdot\overline{X2}\cdot\overline{X3}\cdot X4$	$\overline{X1}\cdot X2\cdot\overline{X3}\cdot X4$	$X1\cdot\overline{X2}\cdot\overline{X3}\cdot X4$	$\overline{X1}\cdot X2\cdot X3\cdot X4$	$X1\cdot\overline{X2}\cdot X3\cdot X4$	$\overline{X1}\cdot\overline{X2}\cdot X3\cdot X4$
$\overline{X1}\cdot X4$	*	*		*		*
$\overline{X2}\cdot X3\cdot X4$					*	*
$X1\cdot\overline{X2}\cdot X3$			*		*	

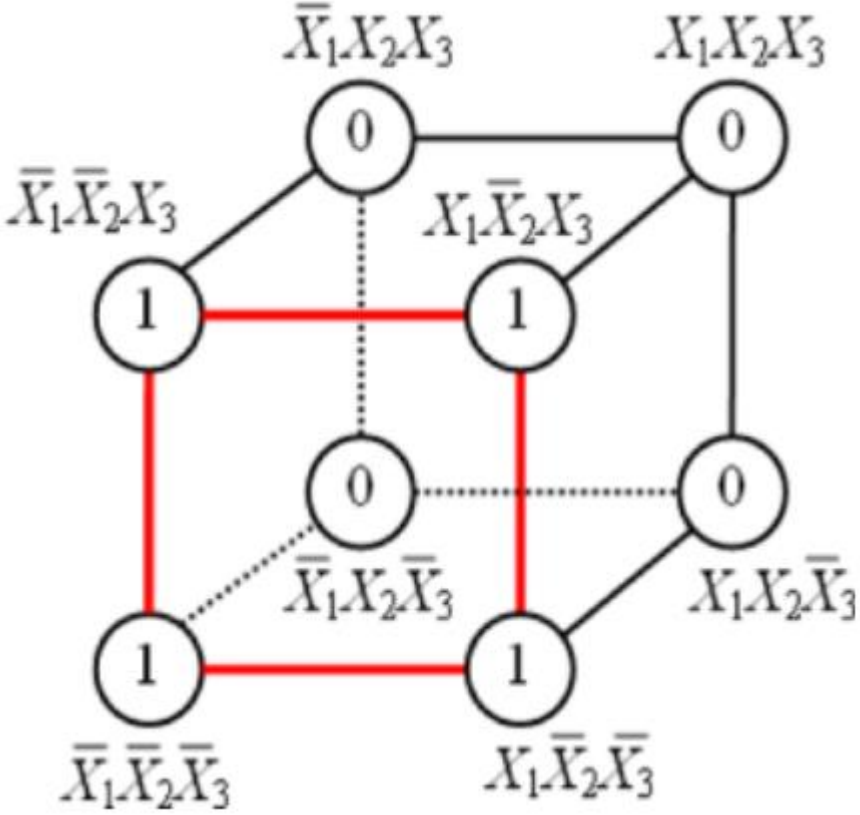
Отримана після мінімізації ФАЛ (приклад мінімізованої ФАЛ подано для розглянутого варіанту) записується в наступному виді:

$$Y = \overline{X1}\cdot X4 \vee X1\cdot\overline{X2}\cdot X3.$$

Застосування карт Карно (Вейча) для мінімізації логічних функцій

Карта Карно – один з графічних способів подання логічних функцій. Для функції n змінних вона складається з 2^n клітинок, причому кожна клітинка відповідає певному набору змінних.

X_1	X_2	X_3	$f(X_1, X_2, X_3)$
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0



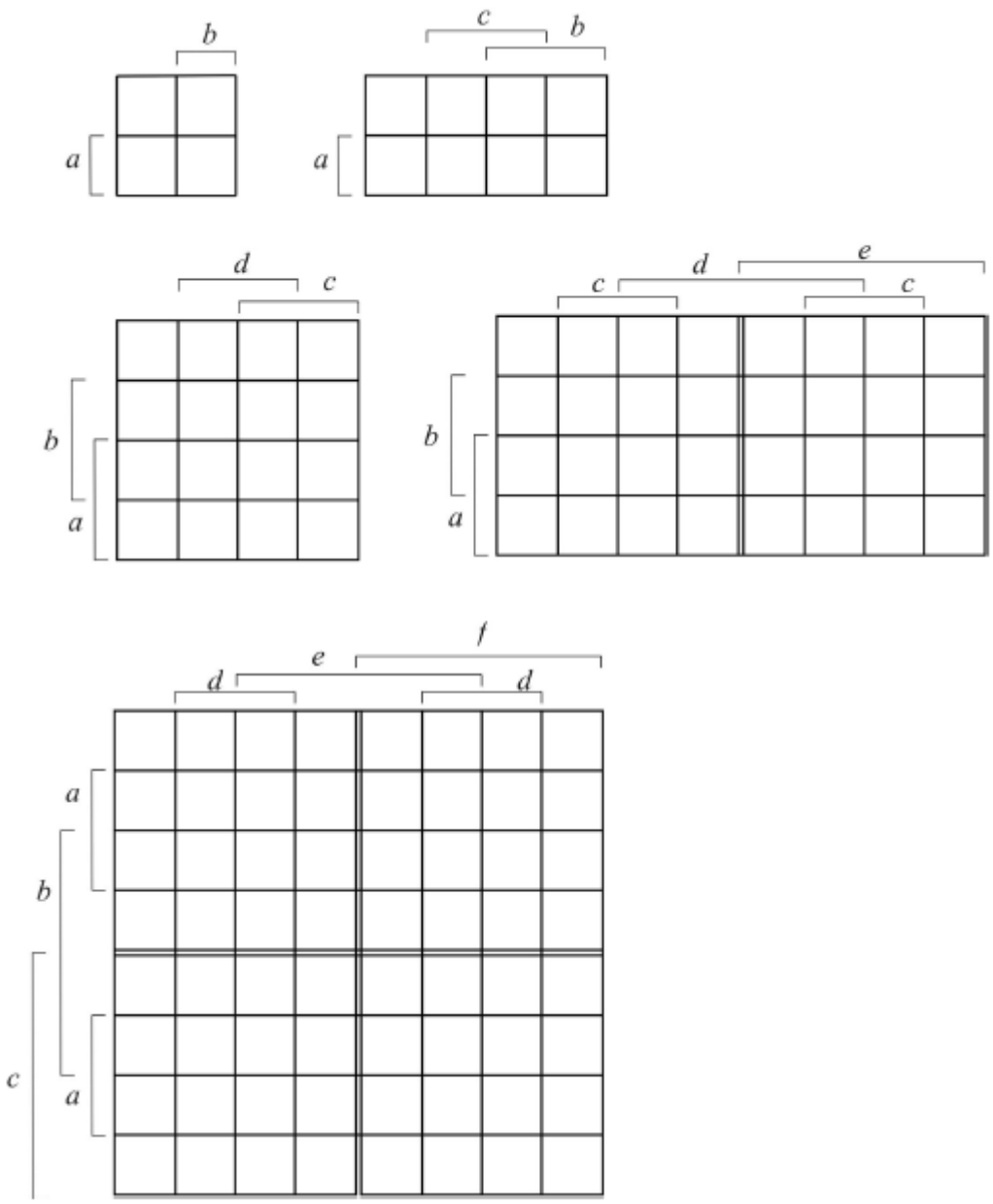
Призначення

Карта Карно для отримання мінімального виду булевої функції зменшує потребу таких обчислень завдяки:

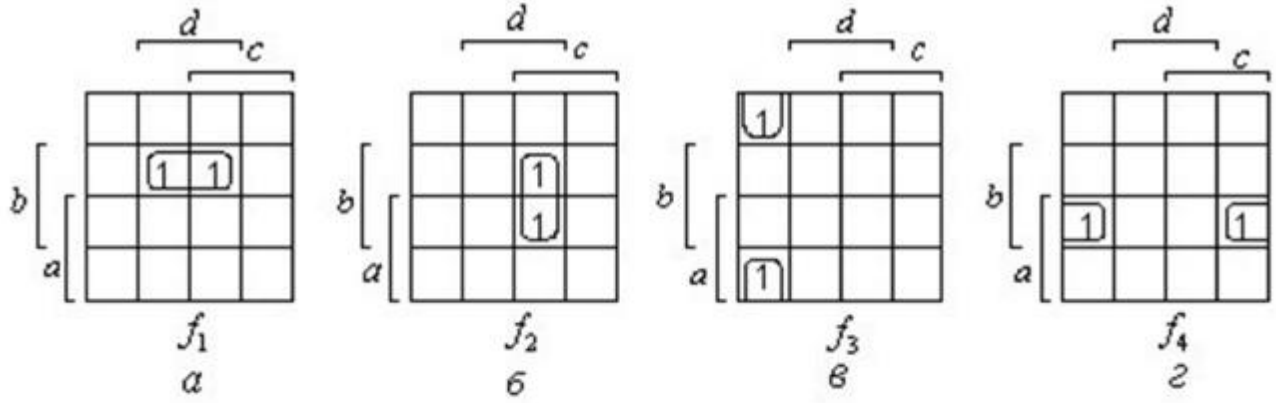
- Використанню можливості людського розуму по розпізнаванню шаблонів для визначення які терми мають бути поєднані для отримання найпростішого виразу.
- Дозволяє швидко визначити та видалити потенційні стани гонитв, які неминучі в булевих рівняннях.
- Забезпечує найкращу допомогу в спрощенні до шістьох змінних, однак з більшою кількістю змінних стає складно розрізнити оптимальні шаблони.
- Допомагає в навчанні про булеві функції та їх мінімізацію.

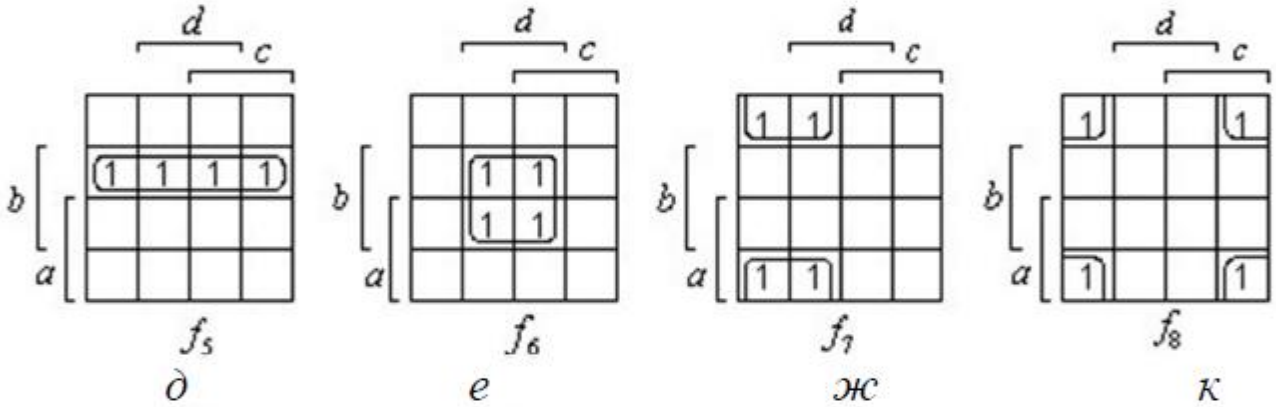
Проблеми

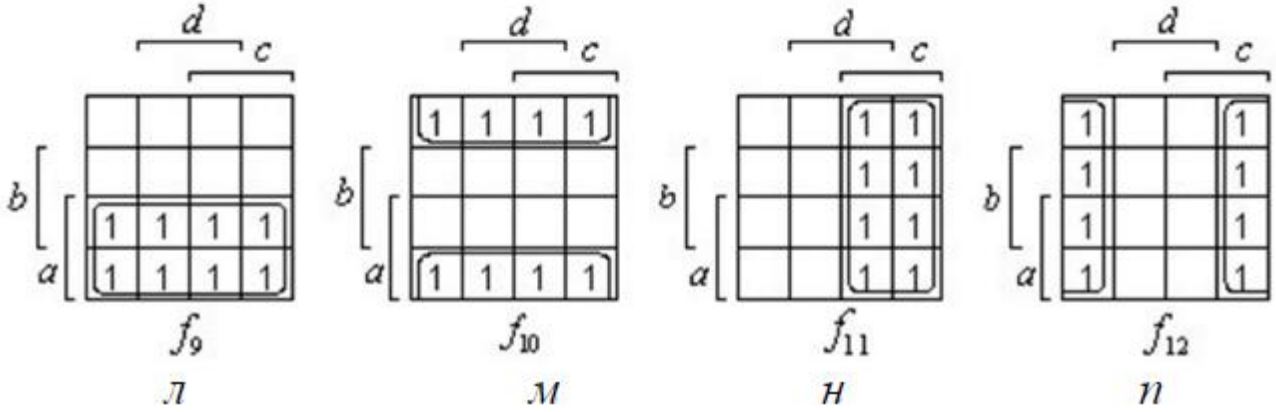
Карта Карно зазвичай становиться важкою для розпізнання при збільшені кількості змінних. Загальне правило таке, карта Карно добре працює до чотирьох-п'яти змінних, і не має використовуватись з більше ніж шістьома змінними. Для виразів з більшою кількістю змінних може бути використаний Метод Куайна — Мак-Класкі. Сьогодні здебільшого для процесу мінімізації використовуються комп'ютери, для яких евристичний алгоритм еспресо став стандартною програмою мінімізації.



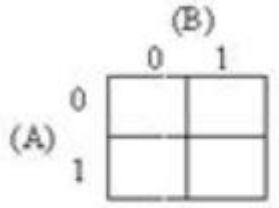
Значення вхідної змінної стосується усіх клітинок у рядку або стовпці і дорівнює одиниці, якщо проти рядка або стовпця є дужка з позначенням цієї змінної. Для решти рядків і стовпців значення змінної дорівнює нулю. Дужки розміщують так, щоб кожна клітинка карти відповідала єдиному набору змінних. Змінні впорядковуються згідно з кодом Грея

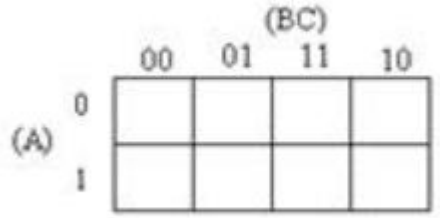


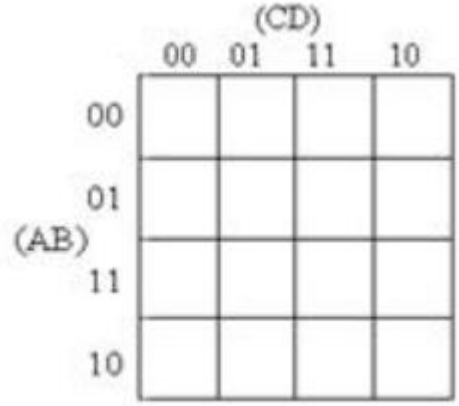




	AB	$A\overline{B}$	$\overline{A}\overline{B}$	$\overline{A}B$
CD				
$C\overline{D}$				
$\overline{C}\overline{D}$				
$\overline{C}D$				







Карти Карно (Вейча)
для функцій 2-х,
3-х і 4-х змінних.

Процес мінімізації можна поділити на три етапи.

Перший етап - заповнення карт Карно (Вейча). У відповідні клітини записують значення функції, що відповідає даному набору.

Другий етап - наведення контурів. На карті наводять контури, що об'єднують «1»; за цим необхідно дотримуватися таких правил:

- 1) контур повинен бути прямокутним і в ньому повинні бути тільки одиниці;
- 2) кількість клітин у контурі повинна бути цілим степенем двійки, тобто 1,2,4,8,16,…;
- 3) одні й ті ж клітини з одиницями можуть входити в декілька контурів;
- 4) крайні праворуч та ліворуч стовпчики вважаються сусідніми, те ж саме й для верхнього та нижнього рядків;
- 5) контури повинні бути якомога більшими, а їх кількість якомога меншою;
- 6) необхідно об'єднати всі одиниці, до яких підходять сформульовані правила.

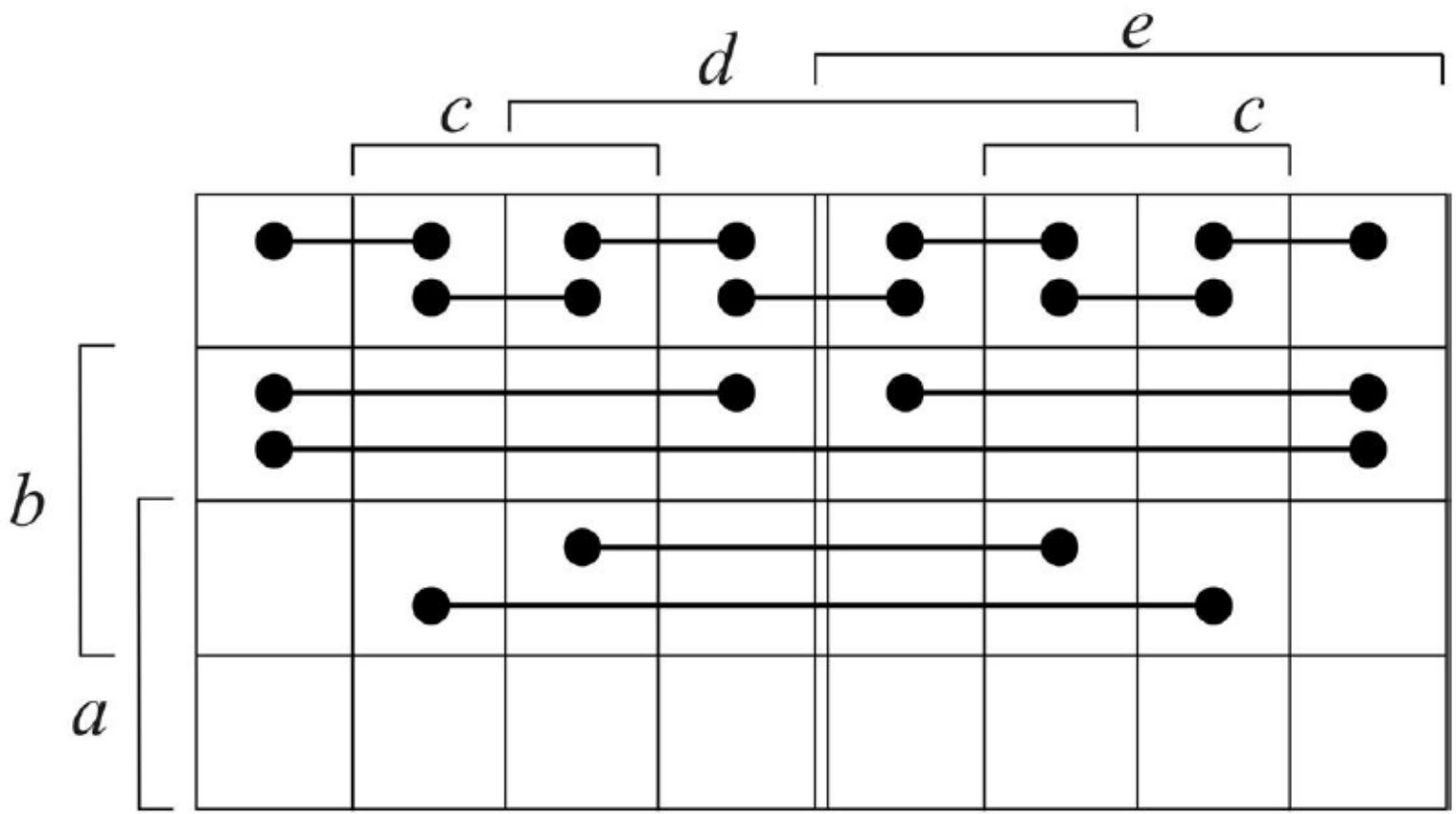
Третій етап - запис мінімізованої логічної функції у вигляді МДНФ. Для запису МДНФ функції необхідно дотримуватися таких правил:

- 1) функцію записують у вигляді диз'юнкції, кожен член якої відповідає одному контуру;
- 2) кожен член диз'юнкції є кон'юнкцією змінних, загальних для даного контуру;
- 3) якщо всі клітини карти Карно (Вейча) охоплені одним контуром, логічна функція тотожно дорівнює одиниці при всіх значеннях наборів.

Якщо в контур об’єднано дві клітинки, то вираз для контуру містить на одну змінну менше порівняно з конституентою одиниці, якщо чотири клітинки – на дві, вісім – на три. Взагалі, якщо контур містить 2^n клітинок, то у вираз для контуру входить на n змінних менше.

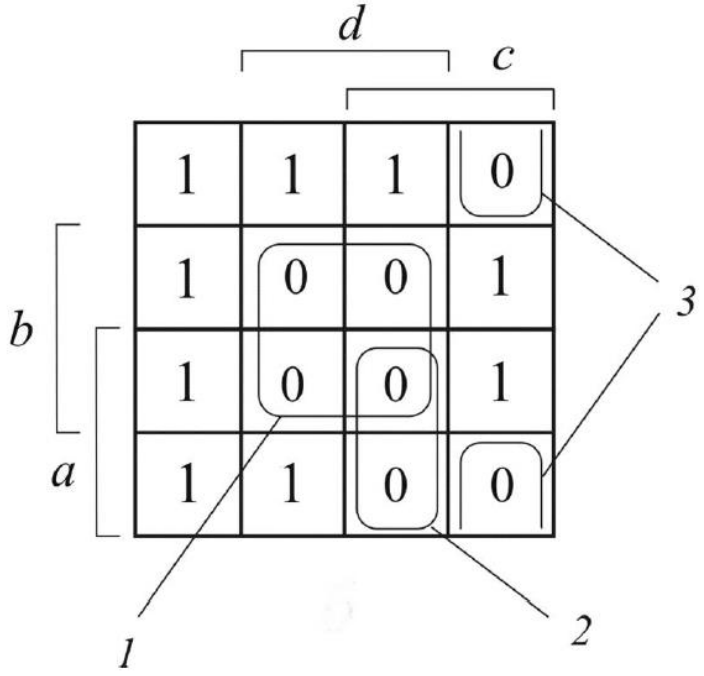
Для мінімізації необхідно побудувати карту для відповідної кількості змінних і нанести на неї задану функцію, об’єднати сусідні клітинки з одиницями у контури, записати вирази для контурів і скласти їх диз’юнкцію. Сусідні клітинки спочатку об’єднують у пари, потім у четвірки із сусідніх пар, тобто пар, що відрізняються тільки однією змінною, після цього сусідні четвірки об’єднують у вісімки тощо. Чим більше клітинок об’єднано у контур, тим простіший вираз, що відповідає контуру, тому слід прагнути того, щоб кожний контур мав якомога більше сусідніх клітинок. При цьому деякі контури можуть частково перекриватися, тобто ті ж самі клітинки можуть одночасно входити у кілька контурів. У контур можна об’єднувати не будь-яку парну кількість клітинок, а тільки 2^n клітинок, тобто 2, 4, 8, 16 тощо.

Сусідніми є не тільки клітинки, що розміщені поряд в одному рядку або стовпці, але й клітинки на протилежних кінцях одного рядка або стовпця. Для чотирьох змінних знаходити їх просто, складніше для п’ятьох, шістьох й більшої кількості змінних. Карту **п’ятьох** змінних можна подати у вигляді **двох карт чотирьох змінних**, карту шістьох змінних – у вигляді чотирьох карт чотирьох змінних тощо. Карти **чотирьох** змінних, що відрізняються значенням тільки однієї змінної, також можна назвати сусідніми. Тоді сусідні карти будуть розміщені поряд або на протилежних кінцях одного рядка або стовпця з карт чотирьох змінних і, отже, сусідніми будуть клітинки, що є сусідніми у тій ж самій карті чотирьох змінних, а також клітинки у сусідніх картах, які розміщуються симетрично відносно ліній, що ділять карту великої кількості змінних на карти чотирьох змінних.



Крім того, необхідно уникати створення зайвих контурів, тобто контурів, усі клітинки яких уже належать до інших контурів. Для цього об’єднання слід починати з тих одиниць, які можуть увійти тільки в один контур.

У контури можна об’єднувати клітинки не тільки з одиницями, але й з нулями. При цьому всі правила об’єднування залишаються попередніми, а функція записується у вигляді кон’юнкції елементарних диз’юнкцій, що відповідають контурам з нулями. Вираз для контуру з нулями записують у вигляді диз’юнкції інверсій координат контуру.

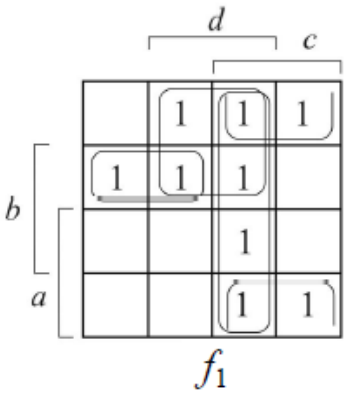


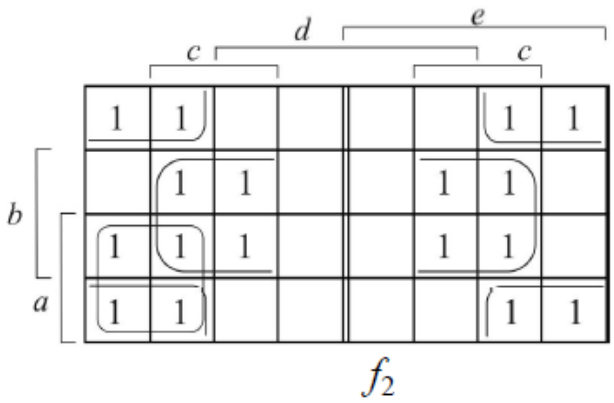
1 - $\overline{b} + \overline{d}$

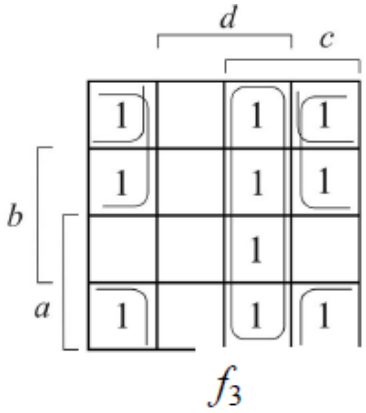
2 - $\overline{a} + \overline{c} + \overline{d}$

3 - $b + \overline{c} + d$

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







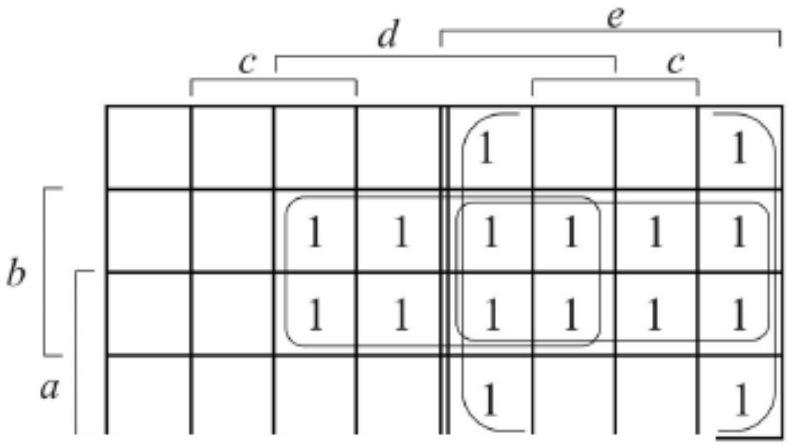
$$f_1 = \bar{a}d + cd + \bar{b}c + \bar{a}b\bar{c};$$

$$f_2 = bc + \bar{b}\,\bar{d} + a\bar{d}\,\bar{e};$$

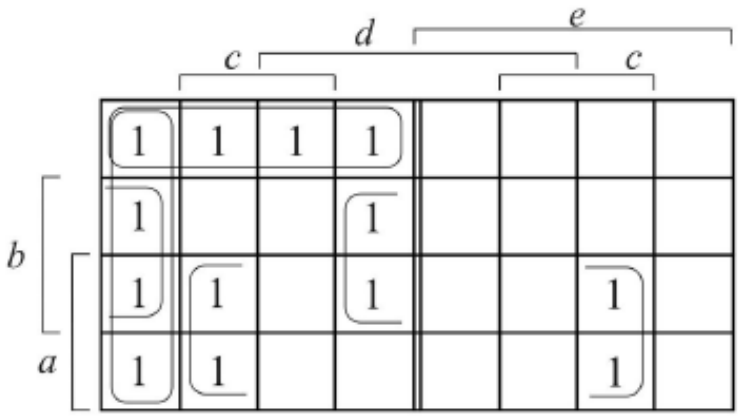
$$f_3 = cd + \bar{b}\,\bar{d} + \bar{a}\,\bar{d};$$

$$f_4 = bd + be + \bar{c}e;$$

$$f_5 = \bar{c}\,\bar{d}\,\bar{e} + \bar{a}\,\bar{b}\,\bar{e} + b\bar{c}\,\bar{e} + ac\bar{d}.$$



f_4



f_5

Мінімізація частково визначених двійкових функцій

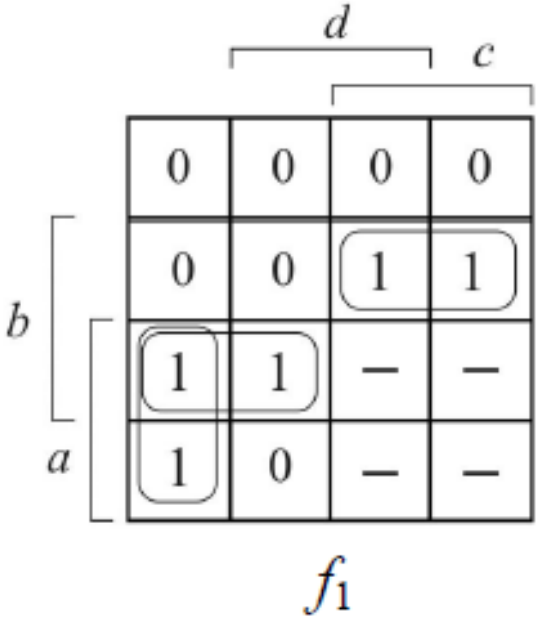
У розглянутих прикладах функції були повністю визначеними, тобто кожному наборові вхідних змінних відповідало цілком визначене значення функції: 0 або 1. Проте під час роботи багатьох дискретних систем керування можуть бути не всі набори вхідних змінних. Наприклад, одночасно не може бути сигналів про верхній та нижній рівні рідини, про наявність механізму у крайньому лівому і крайньому правому положеннях тощо. Набори вхідних змінних, яких за заданих умов роботи ніколи не може бути, називаються *невикористаними станами входів.* У клітинках карти Карно, що відповідають таким станам, проставляють риски. Функції, що описують схеми з невикористаними станами входів, називають *неповністю визначеними*.

Метод Вейча –Карно

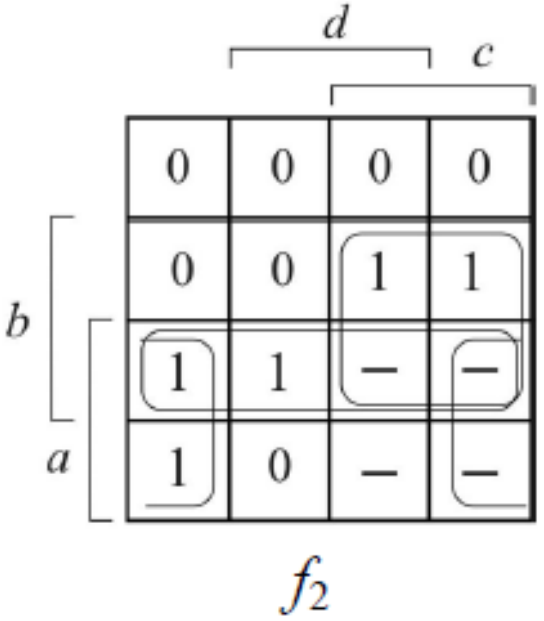
Для спрощення функції у контури можна об’єднувати клітинки не тільки з одиницями, але й із рисками.

$$f_1 = a\bar{d}\,\bar{c} + ab\bar{c} + \bar{a}bc$$

$$f_2 = ab + a\bar{d} + bc$$



f_1



f_2

Функції f_1 і f_2, точно кажучи, не рівносильні. Наприклад, якщо a=1, b=0,c=1,d=0, то f_1=0, а f_2=1. Проте, якщо вважати, що a і c одночасно не дорівнюватимуть одиниці, то схеми, працюватимуть однаково.

Метод Квайна – Мак-Класкі

Нехай дано частково визначену функцію: $f = f(x_1, x_2, ..., x_n)$. Позначимо через j_1 : $j_1 = j_1(x_1, x_2, ..., x_n)$, функцію, яку задано на всіх «байдужих» наборах одиницями; а через j_0 : $j_0 = j_0(x_1, x_2, ..., x_n)$, функцію, яку задано на всіх «байдужих» наборах нулями. Задача оптимального визначення даної функції f зводиться до вибору із скороченого покриття для функції j_1 мінімальної кількості кубів максимальної розмірності, сукупність яких покривала б усі вершини функції j_0.

Ця сукупність якраз і утворює мінімальне покриття частково визначеної функції f. При цьому воно може покривати й деякі 0-куби, відповідні «байдужим» наборам, що свідчить про те, що функцію задано на цих наборах одиничними значеннями.

Приклад. Мінімізувати $f(x_1, x_2, x_3, x_4) = \bigvee_1 (0, 1^*, 2^*, 4^*, 6, 7^*, 8^*, 9, 11^*, 13^*, 14, 15^*)$.

Номери наборів, на яких функцію недовизначено, записано зі знаком «*».

Розв'язання. Складемо таблицю істинності f, j_1 і j_0.

№	x_1	x_2	x_3	x_4	f	j_1	j_0
0	0	0	0	0	1	1	1
1	0	0	0	1	*	1	0
2	0	0	1	0	*	1	0
3	0	0	1	1	0	0	0
4	0	1	0	0	*	1	0
5	0	1	0	1	0	0	0
6	0	1	1	0	1	1	1
7	0	1	1	1	*	1	0

№	x_1	x_2	x_3	x_4	f	j_1	j_0
8	1	0	0	0	*	1	0
9	1	0	0	1	1	1	1
10	1	0	1	0	0	0	0
11	1	0	1	1	*	1	0
12	1	1	0	0	0	0	0
13	1	1	0	1	*	1	0
14	1	1	1	0	1	1	1
15	1	1	1	1	*	1	0

Крок 1. Об'єднаємо 0-куби в групи за кількістю одиниць у кожному двійковому наборі, тобто

$$K_0^0 = \{0 \quad 0 \quad 0 \quad 0\}; K_1^0 = \begin{matrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{matrix}; K_2^0 = \begin{matrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{matrix};$$

$$K_3^0 = \begin{matrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{matrix}; K_4^0 = \{1 \quad 1 \quad 1 \quad 1\}$$

,

Крок 2. Пошук первинних імплікант, а саме:

а) Порівнюємо K_0^0 й K_1^0

(набори, які склеюються, відзначимо символом «*»);

0 0 0 0 *	0 1 0 0 *
	0 0 0 1 *
	0 0 1 0 *
	1 0 0 0 *

На підставі порівняння будуємо 1-куби, у яких поглинуту координату замінюємо символом «–», а саме:

0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
0 1 0 0	0 0 0 1	0 0 1 0	1 0 0 0
0 - 0 0	0 0 0 -	0 0 - 0	- 0 0 0

б) Порівнюємо K_1^0 й K_2^0:

0	0	0	1 *						
0	0	1	0 *						
0	1	0	0 *		0	1	1	0 *	
1	0	0	0 *		1	0	0	1 *	

На базі порівняння будуємо такі 1-куби:

0	0	1	0		0	1	0	0		0	0	0	1		1	0	0	0
0	1	1	0		0	1	1	0		1	0	0	1		1	0	0	1
0	-	1	0		0	1	-	0		-	0	0	1		1	0	0	-

в) Порівнюємо K_2^0 й K_3^0:

					0	1	1	1 *	
					1	0	1	1 *	
0	1	1	0 *		1	1	0	1 *	
1	0	0	1 *		1	1	1	0 *	

На підставі порівняння будуємо 1-куби:

0 1 1 0	0 1 1 0	1 0 0 1	1 0 0 1
0 1 1 1	1 1 1 0	1 0 1 1	1 1 0 1
0 1 1 -	- 1 1 0	1 0 - 1	1 - 0 1

г) Порівнюємо K_3^0 й K_4^0:

0 1 1 1 *	
1 0 1 1 *	
1 1 0 1 *	
1 1 1 0 *	1 1 1 1 *

На основі порівняння будуємо 1-куби:

0 1 1 1	1 0 1 1	1 1 0 1	1 1 1 0
1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1
- 1 1 1	1 - 1 1	1 1 - 1	1 1 1 -

Як бачимо, первинних імплікант четвертого рангу функції j_1 немає.

Розіб'ємо всі 1-куби на групи відповідно до числа одиниць, враховуючи положення незалежної змінної «–», а саме:

$$K_1^1 = \begin{matrix} 0\ 0\ 0\ - \\ 1\ 0\ 0\ - \\ 0\ 1\ 1\ - \\ 1\ 1\ 1\ - \end{matrix}, \quad K_2^1 = \begin{matrix} 0\ 0\ -\ 0 \\ 0\ 1\ -\ 0 \\ 1\ 0\ -\ 1 \\ 1\ 1\ -\ 1 \end{matrix}, \quad K_3^1 = \begin{matrix} 0\ -\ 0\ 0 \\ 0\ -\ 1\ 0 \\ 1\ -\ 0\ 1 \\ 1\ -\ 1\ 1 \end{matrix}, \quad K_4^1 = \begin{matrix} -\ 0\ 0\ 0 \\ -\ 0\ 0\ 1 \\ -\ 1\ 1\ 0 \\ -\ 1\ 1\ 1 \end{matrix}.$$

Порівняння груп, що мають різну кількість одиниць, усередині кубів K_1^1, K_2^1, K_3^1 і K_4^1 зумовлює такий результат:

0 0 0 -	0 1 1 -	0 0 - 0	1 0 - 1
1 0 0 -	1 1 1 -	0 1 - 0	1 1 - 1
———	———	———	———
- 0 0 -	- 1 1 -	0 - - 0	1 - - 1

0 - 0 0	1 - 0 1	- 0 0 0	- 1 1 1
1 - 1 0	1 - 1 1	- 0 0 1	- 1 1 0
0 - - 0	1 - - 1	- 0 0 -	- 1 1 -

Отже, первинних імплікант функції j_1 третього рангу немає.

Отже, за результатами склеювання отримано такі первинні імпліканти:

$$\{ -\ 0\ 0\ -\ \},\ \{\ 0\ -\ -\ 0\ \},\ \{\ -\ 1\ 1\ -\ \},\ \{1\ -\ -\ 1\ \}.$$

Крок 3. Розміщення позначок.

Первинні імпліканти j_1	Вихідні терми (**лише з 1**)			
	0 1 1 0	1 1 1 0	0 0 0 0	1 0 0 1
–0 0 –			V	V
– 1 1 –	V	V		
1 – – 1				V
0 – – 0	V		V	

Крок 4. Пошук істотних імплікант.

Істотною тут виявляється первинна імпліканта { – 1 1 – }. Із таблиці викреслюємо відповідні їй стовпці.

Крок 5. Вибір мінімального покриття.

Мінімальне покриття термів, що залишилися, здійснює первинна імпліканта { – 0 0 – }.

Таким чином, тупикова форма заданої частково визначеної функції має такий вигляд:

$$f\left(x_1, x_2, x_3, x_4\right) = x_2 x_3 + \overline{x}_2 \overline{x}_3 .$$

Мінімізація систем булевих функцій

На практиці часто необхідно реалізовувати сукупності булевих функцій. Якщо зробити мінімізацію булевих функцій, які входять в систему незалежно одна від одної, то загальна схема буде складатися з ізольованих підсхем. Для того, щоб спростити отриману схему, використовують метод мінімізації булевих функцій, який грунтується на методі Квайна.

Нехай задано систему повністю визначених функцій, які подані в ДНФ:

$$\begin{cases} f_1(x_1, x_2, x_3) = x_1\overline{x_3} + x_1\overline{x_2} + \overline{x_1}x_3; \\ f_1(x_1, x_2, x_3) = x_1\overline{x_2} + \overline{x_1}x_3 + \overline{x_1}x_2; \\ f_1(x_1, x_2, x_3) = x_1x_2 + \overline{x_1}\,\overline{x_2}\,\overline{x_3}. \end{cases}$$

Всі різні елементарні кон’юнкції системи об’єднуємо в множину А, яку назвемо повною множиною елементарних кон’юнкцій системи функцій.

$$A = \left\{ x_1 \overline{x_3}; x_1 \overline{x_2}; \overline{x_1} x_3; \overline{x_1} x_2; x_1 x_2; \overline{x_1}\,\overline{x_2}\,\overline{x_3} \right\}$$

Сума рангів (число букв) елементарних кон’юнкцій множини А є зручним критерієм для оцінювання складності заданої системи.

Означення. Система ДНФ булевих функцій називається мінімальною, якщо її повна множина елементарних кон’юнкцій має мінімальну кількість букв, а кожна ДНФ булевої функції системи містить мінімальне число елементарних кон’юнкцій найбільшого рангу.

Алгоритм мінімізації систем булевих функцій

1. Побудувати повну множину А елементарних кон'юнкцій системи, яку мінімізуємо, враховуючи, що спочатку кожна з функцій системи подана в ДДНФ. Кожній конституенті одиниці множини А присвоїти ознаку, що містить номери функцій системи, в яку входить розглядувана конституента.
2. Виконати мінімізацію ДДНФ функції ϕ, конституентами якої є всі елементи множини А. При цьому:

а) при склеюванні двох конституент одиниці кожній одержаній елементарній кон'юнкції присвоїти ознаку, що складається з номерів функцій, загальних для двох склеюваних конституент одиниці;

б) якщо ознаки не мають спільних номерів, то склеювання не відбувається;

в) поглинання відбувається тільки для елементарних кон’юнкцій з однаковими ознаками. Одержані в результаті склеювання і поглинання кон’юнкції називаються простими імплікантами системи функцій.

3. Побудувати таблицю імплікант функції ϕ, аналогічно до методу Квайна, тільки для кожної конституенти одиниці виділяється стільки стовпців, скільки різних номерів функцій має її ознака.

Приклад. Система булевих функцій задана таблицею істинності.

Таблиця істинності

x_1	x_2	x_3	f_1	f_2
0	0	0	1	1
0	0	1	0	0
0	1	0	0	1
0	1	1	0	1
1	0	0	0	0
1	0	1	1	1
1	1	0	1	0
1	1	1	1	0

Подамо кожну з функцій в ДДНФ:

$$f_1 = \overline{x_1}\,\overline{x_2}\,\overline{x_3} + x_1\overline{x_2}x_3 + x_1x_2\overline{x_3} + x_1x_2x_3;$$

$$f_2 = \overline{x_1}\,\overline{x_2}\,\overline{x_3} + \overline{x_1}x_2\overline{x_3} + \overline{x_1}x_2x_3 + x_1\overline{x_2}x_3.$$

1. Побудуємо повну множину елементарних кон'юнкцій системи, приписуючи кожній конституенті ознаку входження до функцій f_1 і f_2 :

$$A = \left\{\overline{x_1}\,\overline{x_2}\,\overline{x_3}(1,2); \overline{x_1}x_2\overline{x_3}(2); \overline{x_1}x_2x_3(2); x_1\overline{x_2}x_3(1,2); x_1x_2\overline{x_3}(1); x_1x_2x_3(1)\right\}.$$

2. Будуємо ДДНФ функції ϕ :

$$\phi = \overline{x_1}\,\overline{x_2}\,\overline{x_3}(1,2) + \overline{x_1}x_2\overline{x_3}(2) + \overline{x_1}x_2x_3(2) + x_1\overline{x_2}x_3(1,2) + x_1x_2\overline{x_3}(1) + x_1x_2x_3(1).$$

1 2 3 4 5 6

Пронумеруємо конституенти для зручності склеювання.

1-2 : $\overline{x_1}\,\overline{x_3}(2) = \overline{x_1}\,\overline{x_2}\,\overline{x_3}(1,2) + \overline{x_1}\,x_2\,\overline{x_3}(2)$

2-3 : $\overline{x_1}\,x_2(2) = \overline{x_1}\,x_2\,\overline{x_3}(2) + \overline{x_1}\,x_2\,x_3(2)$

4-6 : $x_1 x_3(1) = x_1\,\overline{x_2}\,x_3(1,2) + x_1 x_2 x_3(1)$

5-6 : $x_1 x_2(1) = x_1 x_2\,\overline{x_3}(1) + x_1 x_2 x_3(1)$

Після проведення поглинань ($\overline{x_1}\,\overline{x_3} + \overline{x_1}\,x_2\,\overline{x_3} = \overline{x_1}\,\overline{x_3}(1+\overline{x_2}) = \overline{x_1}\,\overline{x_3}$), з урахуванням ознаки, маємо:

$$\phi = \overline{x_1}\,\overline{x_3}(2) + x_1 x_3(1) + \overline{x_1}\,x_2(2) + x_1 x_2(1) + x_1\,\overline{x_2}\,x_3(1,2) + \overline{x_1}\,\overline{x_2}\,\overline{x_3}(1,2)\,.$$

Таким чином отримано прості імпліканти вихідної системи функцій

3. Будуємо імплікантну матрицю. Стовпці – конституенти одиниці з ДДНФ функції ϕ. Для кожної конституенти виділяємо стільки стовпців, скільки різних номерів функцій мають ознаку конституенти. Рядки матриці – прості імпліканти системи. Заповнення матриці аналогічно до методу Квайна. Отримане ядро покриває всі конституенти одиниці функції ϕ

	Конституенти одиниці функції ϕ							
	$\overline{x_1}\,\overline{x_2}\,\overline{x_3}$		$\overline{x_1}\,x_2\,\overline{x_3}$	$\overline{x_1}\,x_2\,x_3$	$x_1\,\overline{x_2}\,x_3$		$x_1\,x_2\,x_3$	$x_1\,x_2\,\overline{x_3}$
	1	2	2	2	1	2	1	1
$\overline{x_1}\,\overline{x_3}(2)$		v	v					
$x_1 x_3(1)$					v		v	
$\overline{x_1}\,x_2(2)$			v	v				
$x_1 x_2(1)$							v	v
$x_1\,\overline{x_2}\,x_3(1,2)$					v	v		

$\overline{x_1}\,\overline{x_2}\,\overline{x_3}(1,2)$	v	v						

$$\phi = \overline{x_1}\,\overline{x_2}\,\overline{x_3}(1,2) + x_1\overline{x_2}x_3(1,2) + \overline{x_1}x_2(2) + x_1x_2(1).$$

Виділяємо для функції f_i імпліканти з ознакою, що містить ознаку i, отримаємо таку мінімальну диз'юнктивну нормальну форму системи.

$$\begin{cases} f_1 = \overline{x_1}\,\overline{x_2}\,\overline{x_3} + x_1\overline{x_2}x_3 + x_1x_2; \\ f_2 = \overline{x_1}\,\overline{x_2}\,\overline{x_3} + x_1\overline{x_2}x_3 + \overline{x_1}x_2. \end{cases}$$

Недолік: велика громіздкість проведення операцій склеювання та поглинання з ознакою.

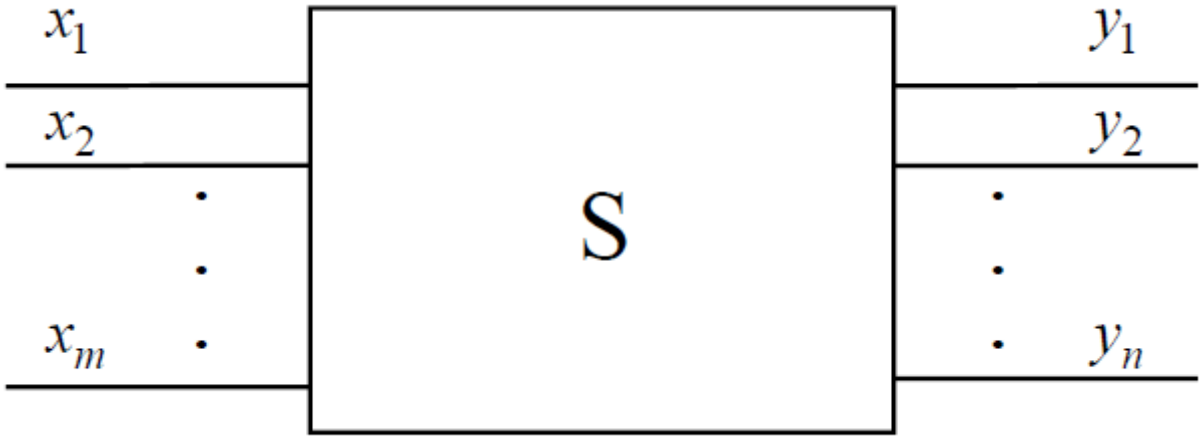
Лекція 3

КОМБІНАЦІЙНІ СХЕМИ, ЇХ ХАРАКТЕРИСТИКИ. ЛОГІЧНІ ЕЛЕМЕНТИ

СИНТЕЗ ОДНОТАКТНИХ СХЕМ

КОМБІНАЦІЙНІ СХЕМИ, ЇХ ХАРАКТЕРИСТИКИ. ЛОГІЧНІ ЕЛЕМЕНТИ

Комбінаційною схемою (КС) називають схему, що є технічною реалізацією булевої функції або сукупності булевих функцій. У загальному випадку КС можна зобразити у вигляді «чорної скриньки», де x_i –логічні входи, а y_j –її логічні виходи.



КС описується за допомогою системи рівнянь

$$\begin{cases} y_1 = F_1(x_1, x_2, \ldots, x_m), \\ y_2 = F_2(x_1, x_2, \ldots, x_m), \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ y_n = F_n(x_1, x_2, \ldots, x_m), \end{cases}$$

де $\{F_i(x_1, x_2, \ldots, x_m), i = \overline{1,n}\}$ – множина улевих функцій. Структурно КС може бути подана як система елементарних логічних схем –логічних елементів. Логічні елементи виконують над вхідними змінними елементарні логічні операції типу І, АБО, І-НІ, АБО-НІ тощо. Графічне зображення КС, у якому показані зв'язки між її різними логічними елементами, які зображені умовними позначеннями, називають *функціональною схемою*. У ході розроблення КС доводиться вирішувати задачі її **аналізу та синтезу**.

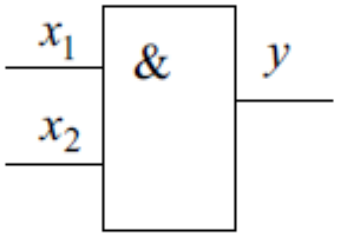
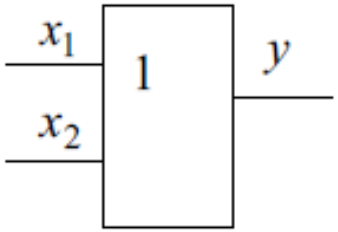
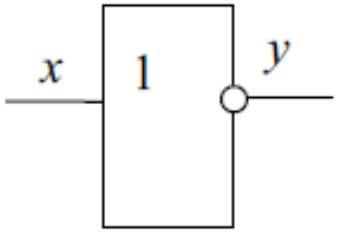
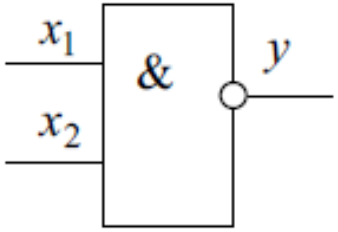
Аналіз КС полягає у визначенні її **статичних та динамічних** властивостей. У **статиці** визначаються булеві функції, які реалізуються КС за відомою її функціональною схемою. У **динаміці** досліджується наявність на виходах КС можливих небажаних імпульсних сигналів при зміні значень логічних змінних на її входах.

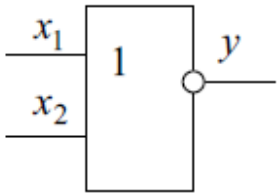
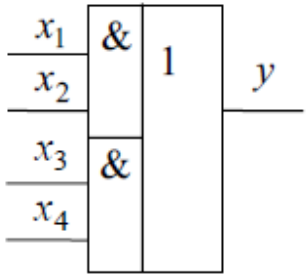
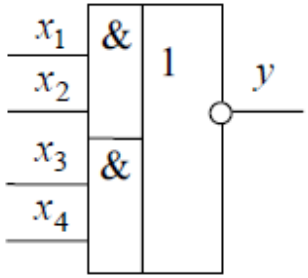
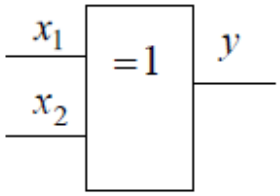
Синтез полягає в побудові КС, що реалізує задану систему булевих функцій із заданого набору логічних елементів. Розв'язок задачі синтезу не є однозначним, оскільки можна запропонувати різні варіанти КС, що реалізують одну йту ж систему булевих функцій. Розробник КС з цієї множини варіантів вибирає один, виходячи з додаткових критеріїв: мінімальної кількості логічних елементів, необхідних для реалізації КС, максимальної швидкодії функціонування КС тощо.

Логічні елементи

Логічний елемент –це технічний пристрій, що реалізує одну з логічних функцій. У таблиці зображені основні логічні елементи.

Кількість входів у логічних елементах різного призначення може бути різною, але входи кожного елемента рівнозначні. Деякі з них при роботі в конкретних схемах можуть не використовуватися. Входи, які не використовуються в схемах І, І-НІ, з’єднують із напругою живлення, а в схемах АБО, АБО-НІ, суматора за модулем 2 –із 0В.

Графічне позначення логічного елемента	Назва логічного елемента	Функція, що виконується логічним елементом
	Елемент І (кон'юнктор)	$y = x_1 x_2$
	Елемент АБО (диз'юнктор)	$y = x_1 + x_2$
	Елемент НІ (інвертор 1)	$y = \bar{x}$
	Елемент І-НІ	$y = \overline{x_1 x_2}$

Графічне позначення логічного елемента	Назва логічного елемента	Функція, що виконується логічним елементом
	Елемент АБО-НІ	$y = \overline{x_1 + x_2}$
	Елемент І-АБО	$y = x_1x_2 + x_3x_4$
	Елемент І-АБО-НІ	$y = \overline{x_1x_2 + x_3x_4}$
	Суматор за модулем 2	$y = \bar{x}_1x_2 + x_1\bar{x}_2$

Основними параметрами логічних елементів є:

- напруги живлення та сигналів, які відповідають 0 та 1 (напруга живлення складає +5В, рівень 1 - 2.4В – 5В, рівень 0 – 0В – 0.4В);

- коефіцієнти об'єднання за входом $K_{об}$ (визначає функцію скількох змінних може реалізувати цей елемент. Звичайно $K_{об}$ набуває значення від 2 до 4, рідше 8. Збільшення кількості входів пов'язано з ускладненням схеми елементів та призводить до погіршення інших параметрів;

- коефіцієнт розгалуження за виходом $K_{роз}$ (показує на скільки логічних входів може бути одночасно навантажений вихід даного логічного елемента. Як правило, у типових логічних елементах $K_{роз}$=10. Іноді замість $K_{роз}$ задається граничне допустиме значення вихідного струму логічного елемента в стані 0 або 1;

- розсіювана потужність;

- завадостійкість – це здатність елемента правильно функціонувати за наявності завад. Вона визначається максимально допустимою напругою завади, за якої не відбувається збою роботи логічного елемента. Переважно ця напруга складає 0.6В – 0.9В;

-швидкодія (характеризується часом затримки розповсюдження сигналу через логічний елемент. Цей параметр істотно залежить від технології виготовлення мікросхем та знаходиться в діапазоні від одиниць до сотень наносекунд).

Основні характеристики комбінаційних схем

Складність комбінаційної схеми оцінюється **кількістю її логічних елементів**. Для конкретної елементної бази кількість елементів визначається **числом корпусів (модулів)** інтегральних мікросхем, що використовуються в схемі. У теоретичних розробках КС орієнтуються на довільну елементну базу і використовується **оцінка складності за Квайном**, яка визначається сумарною кількістю входів логічних елементів у складі схеми (одиниця складності – один **вхід логічного елемента**).

Ціна інверсного входу дорівнює 2. Такий підхід є виправданим:

- складність КС легко обчислюється за булевими функціями, на основі яких будується схема: для ДНФ складність схеми дорівнює сумі кількості

символів (символу зі знаком заперечення відповідає ціна 2) та кількості знаків диз'юнкції + 1 для кожного диз'юнктивного логічного виразу;

- всі класичні методи мінімізації булевих функцій забезпечують мінімальність схеми саме в значенні ціни за Квайном.

Практика показує, що мінімальна за Квайном КС реалізується мінімальною кількістю конструктивних елементів.

Швидкодія комбінаційної схеми оцінюється **максимальною затримкою сигналу** при проходженні його від входу схеми до виходу, тобто визначається **проміжком часу від моменту надходження вхідних сигналів до моменту встановлення відповідних значень вихідних сигналів** КС. Затримка сигналу **кратна кількості елементів, через які проходить сигнал** від входу до виходу КС. Тому швидкодія схеми характеризується значенням

$r\tau$, де τ – затримка сигналу на одному елементі, r – кількість рівнів (глибина) КС. Кількість рівнів КС розраховується таким чином:

- входам КС приписується нульовий рівень;
- логічні елементи, які пов'язані тільки з входами схеми, відносяться до першого рівня;
- елемент відноситься до рівня k, якщо він пов'язаний зі входами елементів рівнів k-1, k-2 й т.д.;
- максимальний рівень елементів r визначає кількість рівнів КС та називається **рангом схеми.**

СИНТЕЗ ОДНОТАКТНИХ СХЕМ.

В інженерній практиці під логічним синтезом розуміють процес визначення логічних виразів, які описують схему автоматики, виходячи зі словесного формулювання умов роботи. Логічний синтез є початковим, але дуже важливим етапом проектування. Отриманий в результаті синтезу набір логічних виразів є основою для подальшого проектування – складання схеми на обраній елементній базі, складання програми для програмованих логічних інтегральних схем або програмованих контролерів.

Схема називається **однотактною (комбінаційною)**, якщо стан її виходів **визначається тільки комбінацією значень вхідних сигналів** і не залежить від послідовності їх надходження. Робота однотактної схеми повністю описується таблицею істинності.

Послідовність синтезу однотактних схем

1) за заданими умовами роботи скласти таблицю істинності;

2) за таблицею істинності записати логічні формули у ДДНФ або у ДКНФ;

3) якщо є можливість, записані функції мінімізувати.

Подальше перетворення логічних формул залежить від елементної бази, яку обрано для реалізації схеми. Досить часто скоротити кількість мікросхем можна, якщо використовувати ті, які реалізують функції І – АБО, І – АБО – НІ, ВИКЛЮЧНЕ АБО та ін. Для реалізації схеми на постійних програмованих запам’ятовувальних пристроях (ППЗУ) логічний вираз слід подати у ДДНФ, яку безпосередньо визначають з таблиці істинності, а в разі використання програмованих логічних матриць – у вигляді мінімізованої ДНФ.

Приклад 1. Виконати логічний синтез схеми, яка має три вхідні сигнали *a,b,c* і два вихідні *X* і *Y*; вихідний сигнал *X* = 1, якщо непарна кількість вхідних сигналів дорівнює одиниці; вихідний сигнал *Y* = 1, якщо парна кількість вхідних сигналів дорівнює одиниці (нуль - парне число).

За заданими умовами роботи складаємо таблицю істинності

Вхідні сигнали			Вихідні сигнали	
a	*b*	*c*	*X*	*Y*
0	0	0	0	1
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	0

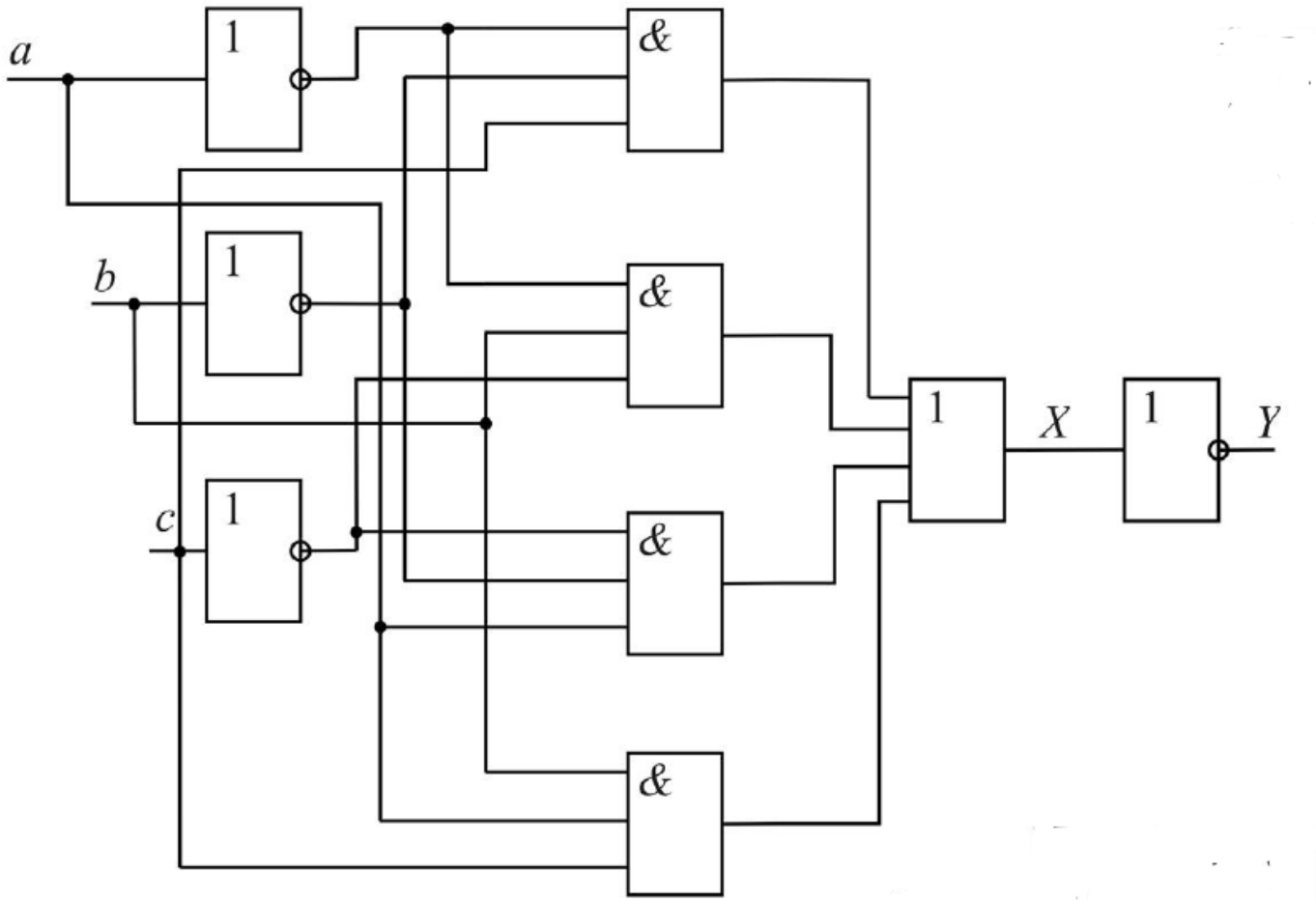
Побудуємо ДДНФ вихідних сигналів

$$X = \overline{a}\,\overline{b}c + \overline{a}\,b\,\overline{c} + a\,\overline{b}\,\overline{c} + abc;$$

$$Y = \overline{a}\,\overline{b}\,\overline{c} + \overline{a}bc + a\,\overline{b}\,c + a\,b\,\overline{c}.$$

Для спрощення схеми можна використати ту обставину, що $Y = \overline{X}$, і не будувати окрему схему для отримання сигналу *Y*, а обмежитися інвертуванням сигналу *X*.

Якщо схему будувати на елементах І, АБО, НІ, то витрати елементів будуть такими: чотири інвертори (три для отримання інверсій *a* ;*b* ;*c* та ще один для отримання $Y = \overline{X}$), чотири схеми І на три входи кожна та одна схема АБО з чотирма входами.



$$X = \bar{a}\,\bar{b}c + \bar{a}\,b\,\bar{c} + a\,\bar{b}\,\bar{c} + abc$$

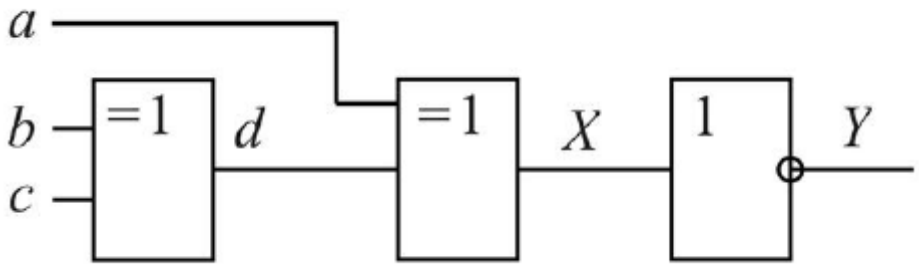
$$Y = \bar{a}\,\bar{b}\,\bar{c} + \bar{a}bc + a\,\bar{b}\,c + a\,b\,\bar{c}.$$

Значного спрощення схеми можна досягти, якщо застосувати елементи ВИКЛЮЧНЕ АБО. Для цього виконаємо такі перетворення виразу для *X*.

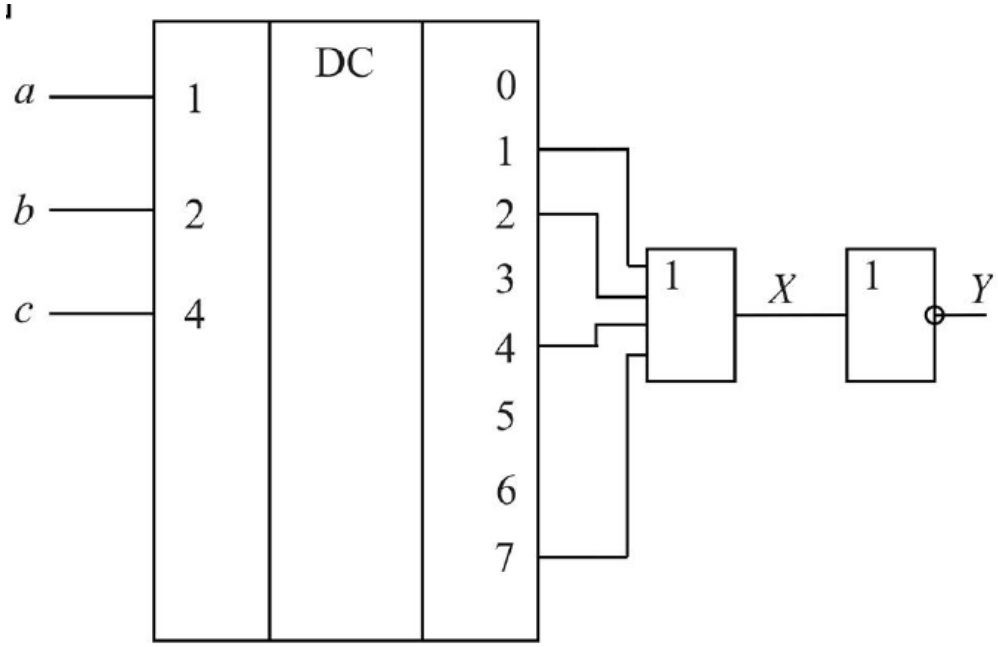
$$X = \overline{a}(\overline{b}c + b\overline{c}) + a(\overline{b}\,\overline{c} + bc).$$

$$\overline{b}c + b\overline{c} = d,\ \overline{b}\,\overline{c} + bc = \overline{d} \Rightarrow X = \overline{a}d + a\overline{d}.$$

d і *X* є функціями ВИКЛЮЧНЕ АБО аргументів *b* і *c* та *a* і *d*



Для побудови схеми можна використати трирозрядний дешифратор. Кожний з вихідних сигналів дешифратора є однією з конституент одиниці. Якщо сигнал *a* подати на вхід старшого розряду, а сигнал *c* – молодшого, то на виході 0 (000) матимемо конституенту $\overline{a}\,\overline{b}\,\overline{c}$, на виході 1 (001) – $\overline{a}\,\overline{b}\,c$, на виході 2 (010) – $\overline{a}\,b\,\overline{c}$ і т.д. Логічна сума сигналів 1, 2, 4, 7 визначає *X*.



$$X = \overline{a}\,\overline{b}c + \overline{a}\,b\,\overline{c} + a\,\overline{b}\,\overline{c} + abc;$$

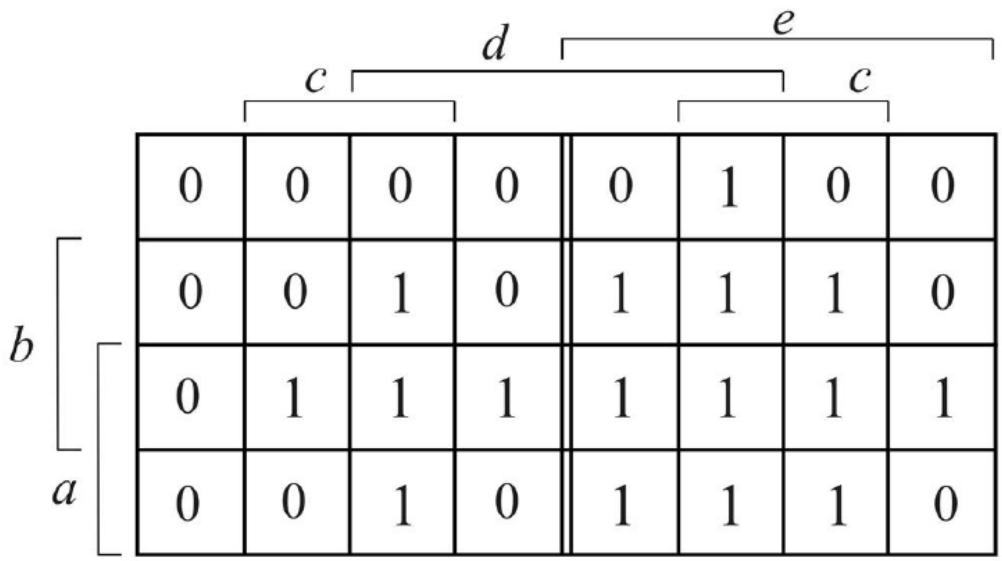
$$Y = \overline{a}\,\overline{b}\,\overline{c} + \overline{a}bc + a\,\overline{b}\,c + a\,b\,\overline{c}.$$

Отже, з розглянутого прикладу видно, що складність схем, які реалізують однакові функції, суттєво залежить від застосованої елементної бази.

За великої кількості вхідних змінних таблицею істинності користуватися незручно, оскільки вона складатиметься з великої кількості рядків. У цьому разі умови роботи схеми зручно одразу перенести на карту Карно і застосувати цю карту для мінімізації функції.

Приклад 2 Схема має п’ять вхідних сигналів *a, b, c, d, e* і один вихідний *X*. Вихідний сигнал реалізує принцип вибору за більшістю, тобто він дорівнює тому значенню, якого набуває більшість вхідних сигналів. Виконати логічний синтез схеми.

Побудуємо карту Карно для п’ятьох змінних і запишемо одиниці у тих клітинках карти, для яких одночасно три, чотири або п’ять змінних дорівнюють одиниці.



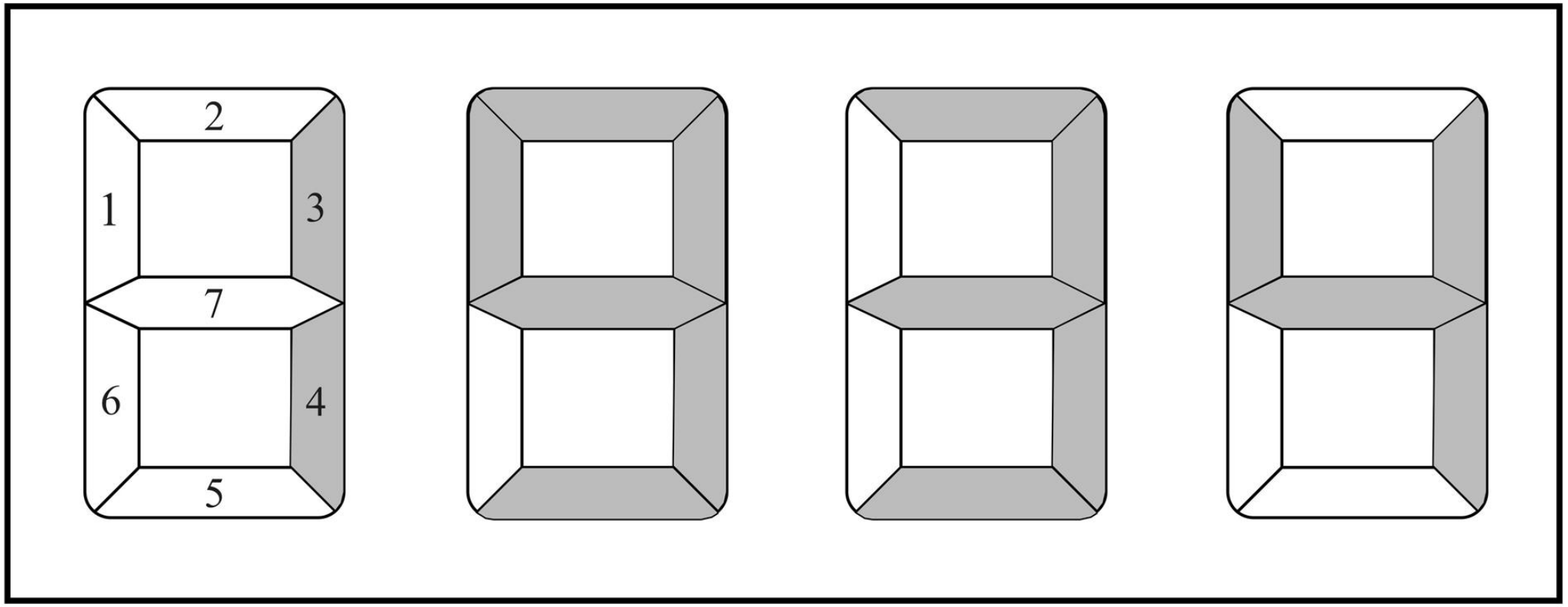
Виконавши мінімізацію функції X за картою Карно, отримаємо

$$X=cde+bce+bcd+bde+abe+abd+abc+acd+ace+ade.$$

Синтез схем при наявності невизначених станів

Розглянемо синтез схеми за наявності невизначених станів.

Приклад 3. Виконати логічний синтез схеми керування світлоцифровим табло.



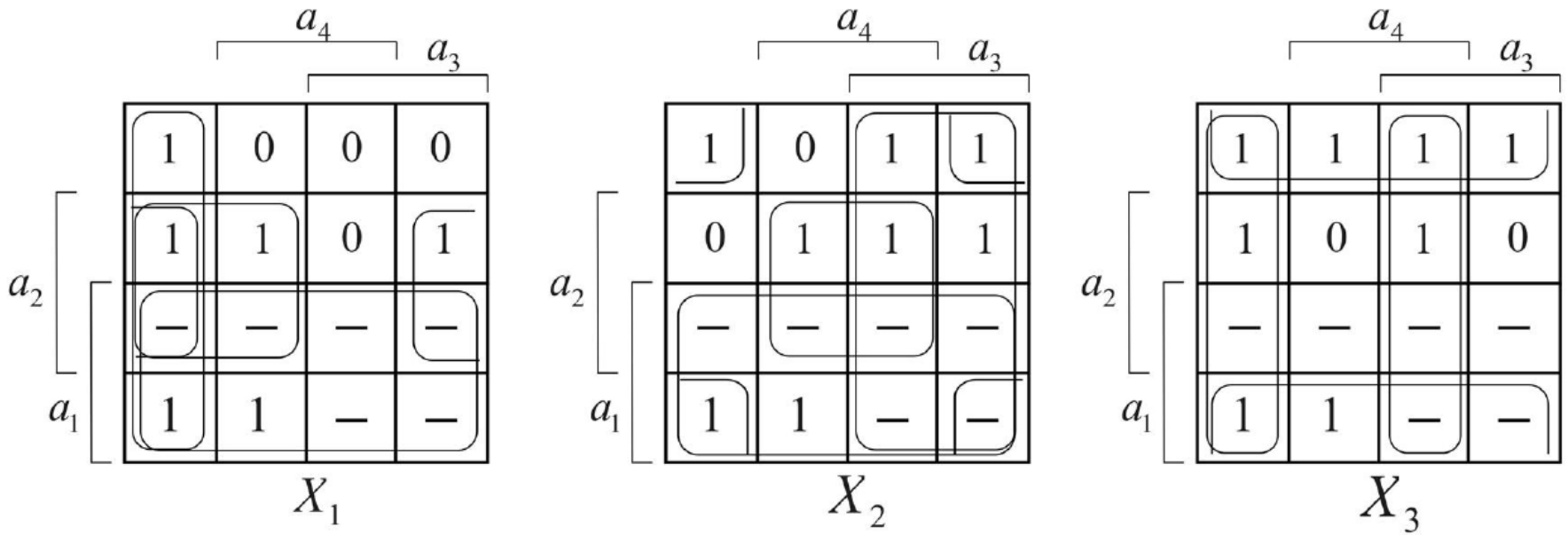
На вхід схеми подається інформація про цифри кожного десяткового розряду у вигляді чотирирозрядного двійкового коду, ці цифри мають зображатися на цифровому табло. При цьому засвічуються різні комбінації смужок, що створюють цифри від 0 до 9.

Виконаємо синтез схеми для одного десяткового розряду, оскільки схеми решти розрядів однакові. Застосуймо такі позначення: a_1 , a_2 , a_3 ,a_4 – вхідні сигнали, причому a_1 – старший розряд двійкового числа, a_4 – молодший; X_1, X_2, ..., X_7 – вихідні сигнали, що подають команди на засвічування першої, другої,…, сьомої смужок.

Роботу схеми можна описати таблицею істинності, де комбінації вхідних змінних від 10 до 15, на вхід не надходять. Тому стани вихідних змінних для цих комбінацій є невизначеними.

Десяткове число	Вхідні змінні				Вихідні змінні						
	a_1	a_2	a_3	a_4	X_1	X_2	X_3	X_4	X_5	X_6	X_7
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	0	1	1	0	0	0
2	0	0	1	0	0	1	1	0	1	1	1
3	0	0	1	1	0	1	1	1	1	0	1
4	0	1	0	0	1	0	1	1	0	0	1
5	0	1	0	1	1	1	0	1	1	0	1
6	0	1	1	0	1	1	0	1	1	1	1
7	0	1	1	1	0	1	1	1	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	1	0	1
10	1	0	1	0	-	-	-	-	-	-	-
11	1	0	1	1	-	-	-	-	-	-	-
12	1	1	0	0	-	-	-	-	-	-	-
13	1	1	0	1	-	-	-	-	-	-	-
14	1	1	1	0	-	-	-	-	-	-	-
15	1	1	1	1	-	-	-	-	-	-	-

Складаємо карти Карно для вихідних змінних і визначаємо їх алгебричні вирази . Об’єднуємо клітинки не тільки з одиницями, але й з рисками.



$$X_1 = a_1 + \bar{a}_3\bar{a}_4 + a_2\bar{a}_3 + a_2\bar{a}_4;$$

$$X_2 = a_1 + a_3 + a_2a_4 + \bar{a}_2\bar{a}_4;$$

$$X_3 = \bar{a}_2 + \bar{a}_3\bar{a}_4 + a_3a_4.$$

$$X_4 = a_1 + a_2 + \bar{a}_3 + a_4;$$

$$X_5 = a_1 + \bar{a}_2\bar{a}_4 + \bar{a}_2a_3 + a_3\bar{a}_4 + a_2\bar{a}_3a_4;$$

$$X_6 = \bar{a}_2\bar{a}_4 + a_3\bar{a}_4;$$

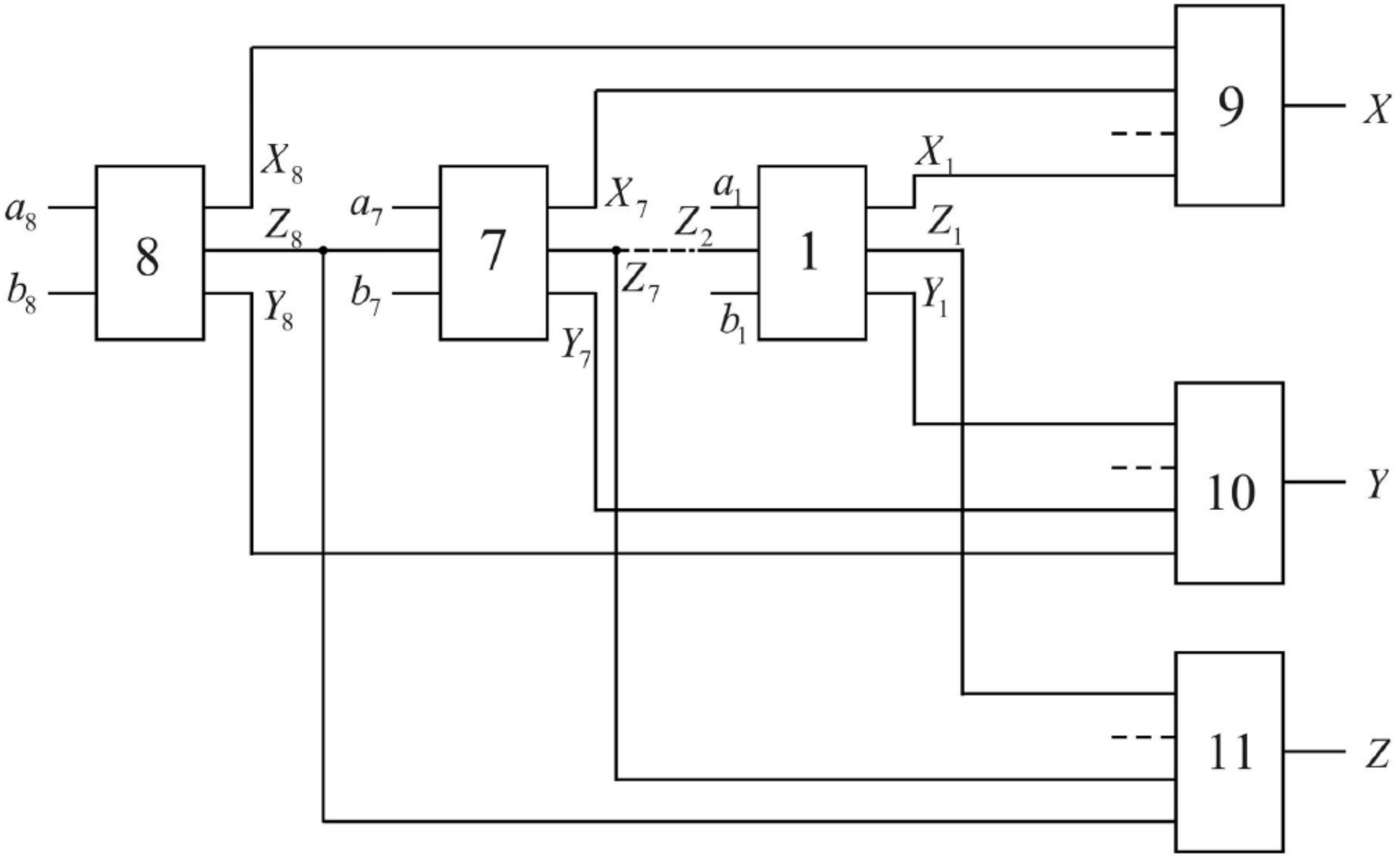
$$X_7 = a_1 + a_2\bar{a}_3 + \bar{a}_2a_3 + a_3\bar{a}_4.$$

Синтез схем з великою кількістю вхідних змінних

Коли кількість вхідних змінних невелика, умови роботи зручно подавати у вигляді таблиці істинності або карти Карно. Задача ускладнюється зі збільшенням кількості вхідних змінних, тому слід прагнути зменшувати її. Якщо є часовий розподіл сигналів, (деякі вхідні сигнали не сумісні в часі), то виконують синтез схем для кожної групи окремо. Іноді для зменшення кількості сигналів доцільно замінювати кілька простих сигналів одним складним, який визначається як комбінаційна функція цих сигналів. Нарешті, у багатьох випадках найбільш доцільним є розроблення функціональної схеми, тобто подання однієї складної схеми у вигляді сполучення окремих структурних елементів (функціональних вузлів або блоків), які мають невелику кількість вхідних сигналів.

Приклад 4. Виконати синтез схеми для порівняння за модулем двох восьмирозрядних двійкових чисел $A = a_8\ a_7\ ...a_1$ і $B = b_8\ b_7\ ...b_1$, де a_i, b_i – цифри (0 або 1) відповідних розрядів чисел A і B. Схема має 16 вхідних сигналів (по 8 розрядів кожного числа) і три вихідні сигнали X,Y,Z, причому $X = 1$, якщо $A > B$, $Y = 1$, якщо $A < B$, $Z = 1$, якщо $A = B$.

Якщо спробувати для цієї задачі скласти таблицю істинності, то вона мала б $2^{16} = 65536$ рядків. Зрозуміла річ, що такий шлях синтезу схеми непридатний. Поділимо схему на функціональні вузли, що виконують прості операції однорозрядного порівняння і мають невелику кількість вхідних сигналів. Отримаємо функціональну схему.


a_8
b_8
8
X_8
Z_8
Y_8
a_7
b_7
7
X_7
Z_7
Y_7
a_1
Z_2
b_1
1
X_1
Z_1
Y_1
9
X
10
Y
11
Z

Вузли 1–8 виконують порівняння цифр a_i і bi кожного розряду окремо. Вузол 8 призначений для порівняння цифр старшого розряду. Він має два вхідні сигнали a_8, b_8 і три вихідні – X_8, Y_8, Z_8 . Вузли 1–7 порівнюють решту розрядів. Причому, якщо у старшому розряді $Z_{i+1} = 0$, тобто $a_{i+1} \neq b_{i+1}$, то подальше порівняння не потрібне, оскільки вже відомо, яке число більше. Тому на входи вузлів 1–7, крім сигналів a_i і b_i, подається третій сигнал Z_{i+1} , який забороняє порівняння, якщо цифри попереднього розряду не є однаковими, тобто, якщо $Z_{i+1} = 0$, то усі вихідні сигнали вузла дорівнюють нулеві.

За умовами роботи вузлів 1–8 складемо таблиці істинності.

Вхідні змінні		Вихідні змінні		
a_8	b_8	X_8	Y_8	Z_8
0	0	0	0	1
0	1	0	1	0
1	0	1	0	0
1	1	0	0	1

Вхідні змінні			Вихідні змінні		
Z_{i+1}	a_i	b_i	X_i	Y_i	Z_i
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	1
1	0	1	0	1	0
1	1	0	1	0	0
1	1	1	0	0	1

$$X_8 = a_8\bar{b}_8; \quad Y_8 = \bar{a}_8 b_8; \quad Z_8 = a_8 b_8 + \bar{a}_8\bar{b}_8;$$

$$X_i = a_i\bar{b}_i Z_{i+1}; \quad Y_i = \bar{a}_i b_i Z_{i+1};$$

$$Z_i = \bar{a}_i\bar{b}_i Z_{i+1} + a_i b_i Z_{i+1}.$$

Вузли 9, 10 і 11 узагальнюють результати порозрядного порівняння. Сигнал $X = 1$, якщо хоча б у будь-якому розряді X_i=1. Тому

$$X = X_1 + X_2 + \ldots + X_8.$$

Аналогічно $Y = Y_1 + Y_2 + \ldots + Y_8.$

Сигнал $Z = 1$ тільки у тому разі, коли в усіх розрядах $Zi = 1$, тому

$$Z = Z_1 Z_2 \ldots Z_8.$$

Іноді для синтезу однотактних схем не обов’язково складати повну таблицю істинності, а достатньо розглянути тільки ті комбінації вхідних сигналів, які можуть бути у реальних умовах роботи.

Приклад 5. Виконати синтез схеми чотирирозрядного шифратора. Шифратор – це перетворювач десяткового коду в двійковий. Він перетворює сигнал на одному зі входів (обов’язково тільки на одному), номер якого відповідає десятковій цифрі від 1 до 9, у чотирирозрядний двійковий код. Шифратор має 9 вхідних змінних і 4 вихідних. Повна таблиця істинності складається з $2^9 = 512$ рядків. Складемо неповну таблицю, включивши до неї тільки можливі комбінації вхідних сигналів. Вхідні сигнали (змінні) позначимо через a_i, де i – номер входу, а вихідні – x_j, де j – номер розряду двійкового числа, тоді таблиця матиме вигляд

Вхідні змінні									Вихідні змінні			
a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	X_4	X_3	X_2	X_1
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	0	0	1	0	0	0	0	1	1
0	0	0	0	0	1	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	0	0	1	1	0
0	0	1	0	0	0	0	0	0	0	1	1	1
0	1	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	0	1	0	0	1

З таблиці видно, що X_1=1, якщо будь-яка із змінних a_1, a_3, a_5, a_7 або a_9 дорівнює одиниці, тому

$$X_1 = a_1 + a_3 + a_5 + a_7 + a_9.$$

Аналогічно для решти вихідних змінних

$$X_2 = a_2 + a_3 + a_6 + a_7; \qquad X_3 = a_4 + a_5 + a_6 + a_7; \qquad X_4 = a_8 + a_9.$$

Аналіз комбінаційних схем методом синхронного моделювання

У даному методі вважається, що всі логічні елементи перемикаються одночасно, без затримки. У результаті використання методу визначається стале значення сигналу на виході схеми.

Аналіз комбінаційних схем методом асинхронного моделювання

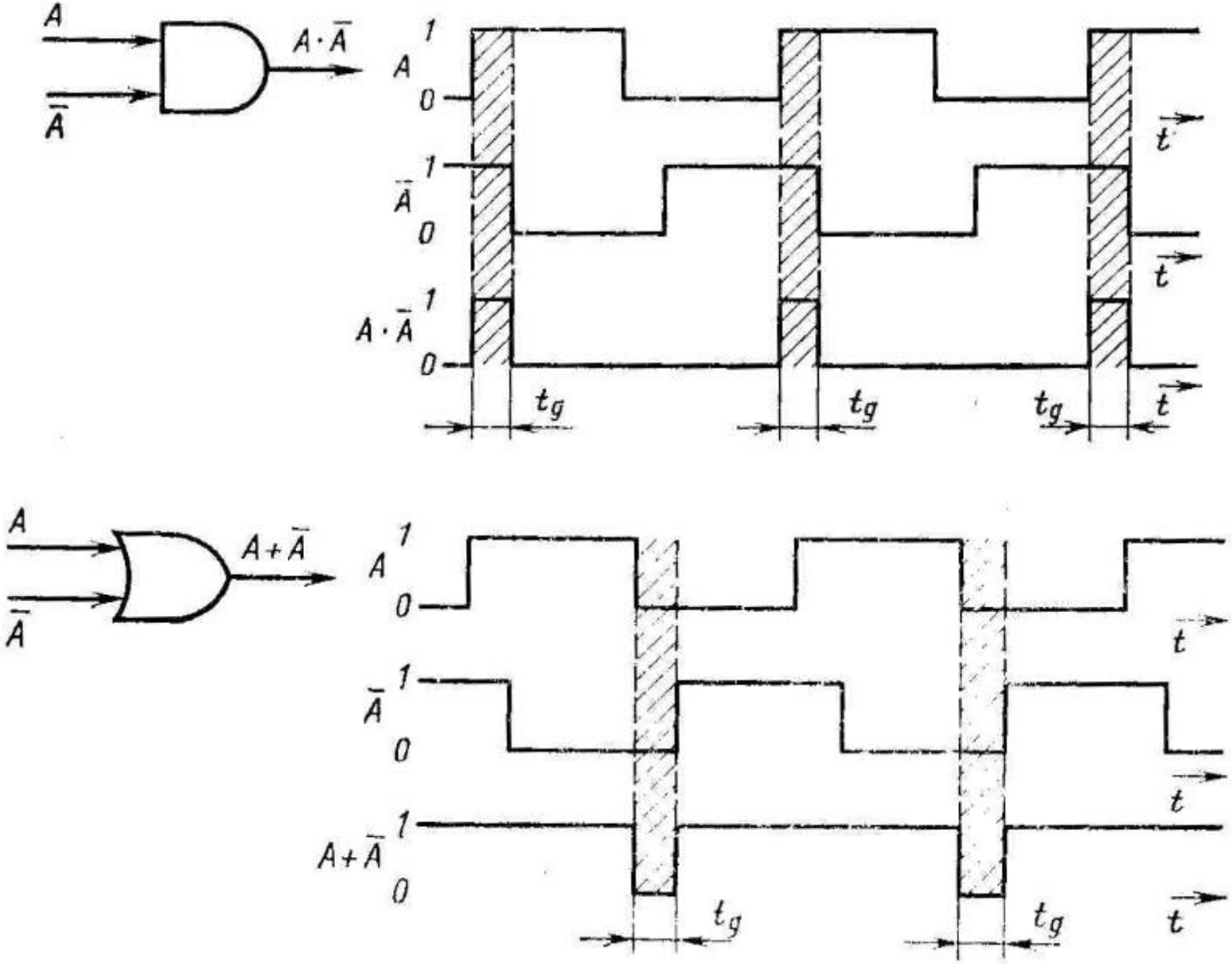
Реальний логічний елемент перемикається за деякий скінченний час, що залежить від технології його виготовлення, умов експлуатації, ємностей навантаження й т.д. Проходження сигналу послідовно через кілька логічних елементів призводить до накопичення часу затримки та виникнення зсуву в часі вихідного сигналу відносно вхідного. Наявність затримки та породжуваного нею часового зсуву сигналів може призвести до появи на виході окремих логічних елементів та всієї схеми в цілому

короткотермінових сигналів, які не передбачені булевою функцією, на основі якої реалізована КС. Таке явище називається **ризиком збою**. Розрізняють статистичний та динамічний ризики збою.

При *статичному ризику* збою до перехідного процесу та після нього стан вихідного сигналу один і той самий, а під час перехідного процесу можлива короткочасна поява протилежного сигналу.

При *динамічному ризику* збою до та після перехідного процесу стани вихідного сигналу протилежні, але в перехідному процесі вихідний сигнал кілька разів змінює своє значення.

Радикальним способом усунення ризиків збою є введення стробування для зняття вихідного сигналу КС. Стробуючий імпульс подається після закінчення перехідного процесу в КС.


A
$\bar{A}$
$A \cdot \bar{A}$
1
0
t
t_g
$A + \bar{A}$

Лекція 4

АБСТРАКТНА ТЕОРІЯ ЦИФРОВИХ ПРИСТРОЇВ (АВТОМАТІВ)

АБСТРАКТНА ТЕОРІЯ ЦИФРОВИХ ПРИСТРОЇВ (АВТОМАТІВ).

Логічний елемент — це електронна схема, що реалізує певну перемикальну функцію. На функціональних схемах логічний елемент зображується за допомогою **умовного графічного позначення (УГП)**.

Сукупність взаємозв'язаних логічних елементів називають **логічною схемою.** Суперпозиції логічних функцій відповідає певна логічна схема.

$$\begin{cases} y_1 = f(x_1, x_2, ..., x_n); \\ y_2 = f(x_1, x_2, ..., x_n); \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ y_m = f(x_1, x_2, ..., x_n), \end{cases}$$

Існують два різновиди логічних схем: **комбінаційні і послідовнісні**.

Функціонування **комбінаційної** схеми може бути описане системою перемикальних функцій. Вона має **тільки один стан**. Вихідні сигнали комбінаційної схеми залежать тільки від значень вхідних логічних сигналів. Основною ознакою цих схем є відсутність **петель** - шляхів від виходу логічного елемента до його входу, можливо, через інші логічні елементи.

Крім комбінаційних схем існують схеми, вихідні сигнали яких залежать не тільки від значень сигналів у даний момент часу, але й від значень сигналів, що надходять на входи в попередні моменти часу, тобто від її передісторії.

Послідовнісні схеми мають **більш ніж один стан**. У зв'язку з чим вихідні сигнали схеми залежать не тільки від вхідних сигналів, а й від стану, в якому перебуває схема. Основною ознакою послідовнісних схем є наявність петель.

Як комбінаційні, так і послідовнісні логічні схеми називають **цифровими автоматами.** Комбінаційні схеми мають назву **тривіальних автоматів**, або автоматів, що не мають пам'яті, а послідовнісні схеми називають **автоматами з пам'яттю**. Для проектування і синтезу цифрових автоматів застосовується математичний апарат теорії кінцевих (цифрових) автоматів.

Цифровий автомат (пристрій) – це дискретний перетворювач інформації, що здатний приймати різні внутрішні стани, переходити під впливом вхідних сигналів або команд закладеної в нього програми розв'язання задачі з одного стану в інший та видавати вихідні сигнали. Перехід автомата з одного стану в інший здійснюється в дискретний момент часу.

Під час описання автомата фіксується деяка множина станів, у яких знаходиться автомат. **Стан** саме і відповідає деякій пам'яті про минуле, дозволяючи **усунути час як змінну** і позначити вихідні сигнали як функцію станів і входів у даний момент часу.

У кожний момент дискретного часу на вхід автомата надходить один вхідний сигнал, що здійснює перехід автомата в новий стан і на виході з'являється один вихідний сигнал.

Процес функціонування автомата полягає у тому, що при подачі на його вхід деякої послідовності вхідних сигналів, він переходить з одного стану в інший і формує послідовність вихідних сигналів.

На практиці найбільше поширення набули два класи абстрактних автоматів – **автомати Мілі (Mealy) та Мура (Moore)**.

Представлення кінцевого автомата фактично зводиться до опису завдання його автоматних функцій.

Існують декілька способів завдання кінцевих автоматів, наприклад :

- табличний (матриці переходів і виходів);
- графічний (за допомогою графів);
- аналітичний (за допомогою формул).

Аналітичний спосіб – автомат задається системою рівнянь. З такої системи виходить, що **при кінцевому числі можливих внутрішніх станів кількість можливих значень автоматних функцій також виявляється кінцевою**.

Математичною моделлю цифрового автомата є так званий **абстрактний автомат**, означений як 6-компонентний об’єкт

$$\mathbf{S} = \{\mathbf{A}, \mathbf{Z}, \mathbf{W}, \delta, \lambda, a_1\}$$

$\mathbf{A} = \{a_m, m = \overline{1, M}\}$ – множина станів (внутрішній алфавіт);

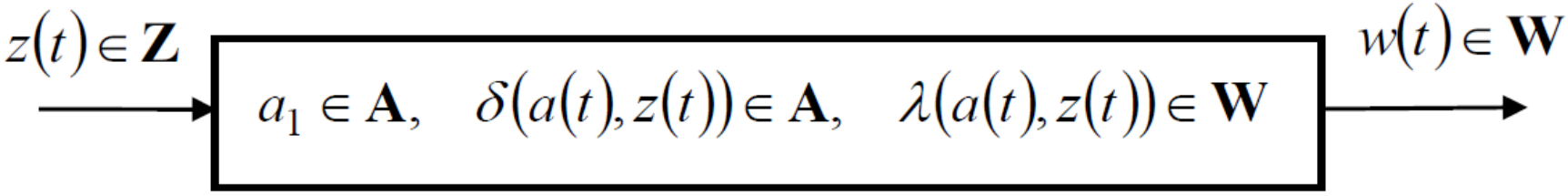
$\mathbf{Z} = \{z_f, f = \overline{1, F}\}$ – вхідний алфавіт, множина вхідних сигналів (символів);

$\mathbf{W} = \{w_g, g = \overline{1, G}\}$ – вихідний алфавіт, множина вихідних сигналів;

$\delta : \mathbf{A} \times \mathbf{Z} \to \mathbf{A}$ – функція переходів, парам (a_m, z_f) ставить у відповідність стани автомата $a_s = \delta(a_m, z_f),\ a_s \in \mathbf{A}$;

$\lambda : \mathbf{A} \times \mathbf{Z} \to \mathbf{W}$ – функція виходу, парам (a_m, z_f) ставить у відповідність вихідні сигнали автомата $w_g = \lambda(a_m, z_f),\ w_g \in \mathbf{W}$;

$a_1 \in \mathbf{A}$ – початковий стан автомата.



Абстрактний автомат має один вхід та один вихід. Його функціонування відбувається в дискретному часі, що подається цілими додатними значеннями t=0,1,2,…В кожен момент дискретного часу автомат знаходиться в деякому стані $a(t)$ з множини можливих станів автомата, причому в початковий момент він завжди знаходиться в початковому стані $a(0)=a_1$. У момент t, знаходячись в стані $a(t)$, автомат здатний сприйняти на вході букву вхідного алфавіту $z(t)$. Відповідно до функції виходів цифровий автомат видає в той же момент часу букву вихідного алфавіту $w(t)$

$$w(t) = \lambda(a(t), z(t))$$

та відповідно до функції переходів перейде в наступний стан $a(t+1)$

$$a(t+1) = \delta(a(t), z(t))$$

Закон функціонування **автомата**

Мілі задається рівняннями

$$a(t+1) = \delta(a(t), z(t));$$
$$w(t) = \lambda(a(t), z(t)),\ t = 0,1,2\ldots$$

Закон функціонування **автомата**

Мура задається рівняннями

$$a(t+1) = \delta(a(t), z(t));$$
$$w(t) = \lambda(a(t)),\ t = 0,1,2\ldots$$

На відміну від автомата Мілі, вихідний сигнал в автоматі Мура залежить тільки від поточного стану автомата та в явному вигляді не залежить від вхідного сигналу. Додатково необхідно вказати початковий стан.

Абстрактні автомати поділяють на:

- повністю визначені та частково визначені;
- детерміновані та ймовірнісні;
- синхронні та асинхронні.

Повністю визначеним називають абстрактний цифровий автомат, у якого функція переходів та функція виходів визначені для всіх пар декартового добутку **ZxA**.

Частково визначеним називаєють абстрактний автомат, у якого функція переходів або функція виходів, або обидві ці функції визначені не для всіх пар декартового добутку **ZxA**.

Детермінований – це автомат, який при заданому стані a_m та заданому вхідному сигналі z обов'язково (з ймовірністю 1) перейде в деякий стан a_n.

Ймовірнісний автомат – це автомат, який при заданому стані a_m та заданому вхідному сигналі z може перейти із заданими ймовірностями в різні стани.

Для означення **синхронних** та **асинхронних** автоматів вводиться поняття стійкого стану.

Стан стійкий, якщо потрапивши в цей стан під дією деякого сигналу z_j, автомат вийде з нього тільки під дією іншого сигналу z_k, відмінного від z_j.

Автомат, у якого **всі стани стійкі**, називають **асинхронним**, у протилежному випадку автомат називається **синхронним**.

Абстрактний автомат називають **скінченним автоматом**, якщо є скінченними множини

$$\mathbf{A} = \{a_m, m = \overline{1, M}\}, \quad \mathbf{Z} = \{z_f, f = \overline{1, F}\}, \quad \mathbf{W} = \{w_g, g = \overline{1, G}\}$$

Автомат називають **ініціальним автоматом**, якщо в ньому виділений початковий стан a_1.

Синтез цифрових автоматів складається з етапів **абстрактного та структурного синтезу**.

Результатом **абстрактного синтезу** автомата є його формальний опис у вигляді $\mathbf{S} = \{\mathbf{A}, \mathbf{Z}, \mathbf{W}, \delta, \lambda, a_1\}$, а структурний синтез дозволяє одержати логічну схему автомата **в заданому елементному базисі.**

Абстрактний синтез охоплює такі етапи:

- формування вхідного та вихідного алфавітів;
- складання алгоритму функціонування автомата;
- формування алфавіту внутрішніх станів автомата;
- визначення функцій переходів і виходів за допомогою графа чи відповідних таблиць.

Вхідний алфавіт формується на основі вивчення зовнішніх для автомата сигналів, які можуть розглядати як логічні умови, що змінюють режим функціонування автомата. Різні комбінації вхідних сигналів розглядаються як букви вхідного алфавіту.

Автомат, як правило, забезпечує управління деяким операційним пристроєм, що перетворює цифрову інформацію шляхом виконання псиних операторів. Кожному оператору відповідає певний набір управляючих сигналів, який на абстрактному рівні розглядається як певна буква **вихідного алфавіту** управляючого автомата.

Отримання алгоритму функціонування автомата не є формалізованим процесом. Для цього необхідно чітко представляти спосіб перетворення інформації.

Методи описування та задавання автоматів

Для того, щоб задати скінченний автомат, необхідно описати всі елементи множини $\mathbf{S} = \{\mathbf{A}, \mathbf{Z}, \mathbf{W}, \delta, \lambda, a_1\}$. Множини **A**, **Z, W** описуються та задаються простим переліком своїх елементів.

Способи задавання функцій δ та λ: **табличний, матричний та графічний**.

При **табличному** методі задавання **автомат Мілі** описується за допомогою двох таблиць. Одна з них (таблиця переходів) задає функцію δ, друга (таблиця виходів) – функцію λ

Таблиця переходів автомата Мілі

	a_1	a_2	a_3	a_4
z_1	a_2	a_1	a_2	a_3
z_2	a_3	a_4	a_1	a_1

Таблиця виходів автомата Мілі

	a_1	a_2	a_3	a_4
z_1	w_1	w_2	w_1	w_2
z_2	w_2	w_3	w_4	w_5

Кожному стовпцю з наведених таблиць поставлено у відповідність один стан із множини **A**, кожному рядку – один вхідний символ з множини **Z**.

На перетині стовпця та рядка таблиці переходів записується стан a_s, в який повинен перейти автомат зі стану a_m під дією вхідного символу z_f: $a_s = \delta(a_m, z_f)$. На перетині стовпця a_m та рядка z_f таблиці виходів записується вихідний символ w_g, виданий автоматом у стані a_m під час надходження на вхід символу z_f: $w_g = \lambda(a_m, z_f)$

Суміщена таблиця переходів-виходів для автомата Мілі

	a_1	a_2	a_3	a_4
z_1	a_2 / w_1	a_1 / w_2	a_2 / w_1	a_3 / w_2
z_2	a_3 / w_2	a_4 / w_3	a_1 / w_4	a_1 / w_5

Автомат Мура задається **однією таблицею** переходів, в якій кожному стовпцю приписується не тільки стан a_m, але й вихідний символ $w_g = \lambda(a_m)$, що відповідає цьому стану.

Таблиця переходів автомата Мура

	w_1	w_2	w_3	w_4
	a_1	a_2	a_3	a_4
z_1	a_1	a_2	a_2	a_3
z_2	a_2	a_3	a_4	a_1

Для **часткових** автоматів на місці **невизначених станів** та вихідних символів ставиться **прочерк**. У таких автоматах вихідний сигнал на будь-якому переході завжди невизначений, якщо невизначеним є стан переходу. Крім того, вихідний сигнал може бути невизначеним й для деяких існуючих переходів.

Матричний метод задавання абстрактних автоматів полягає в поданні автомата у вигляді **матриці з'єднань**. Матриця з'єднань довільного абстрактного автомата є **квадратною** та **має стільки стовпців (рядків), скільки різних станів має автомат**. Кожен стовпець (рядок) матриці з'єднань помічається буквою стану автомата. Якщо автомат ініціальний, то перший зліва стовпець та перший зверху рядок матриці його з'єднань помічаються буквою початкового стану автомата. В клітинці матриці з'єднань, що знаходиться на перетині стовпця, поміченого буквою стану a_i, та рядка, поміченого буквою стану автомата a_j, ставиться вхідний сигнал z_f (або диз'юнкція вхідних сигналів), під дією якого здійснюється перехід автомата зі стану a_j в стан a_i.

Якщо матрицею з'єднань задається абстрактний автомат Мілі, то поруч з буквою вхідного сигналу z_f в дужках вказується буква вихідного сигналу w_g, який автомат Мілі видає, здійснюючи перехід $a_j = \delta(a_i, z_f)$.

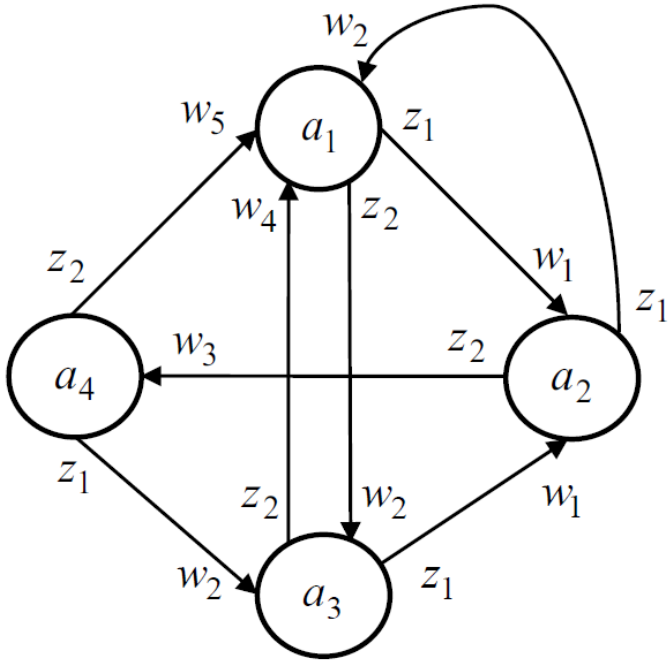
Матриця з'єднань автомата Мілі

	a_1	a_2	a_3	a_4
a_1	-	$z_1(w_1)$	$z_2(w_2)$	-
a_2	$z_1(w_2)$	-	-	$z_2(w_3)$
a_3	$z_2(w_4)$	$z_1(w_1)$	-	-
a_4	$z_2(w_5)$	-	$z_1(w_2)$	-

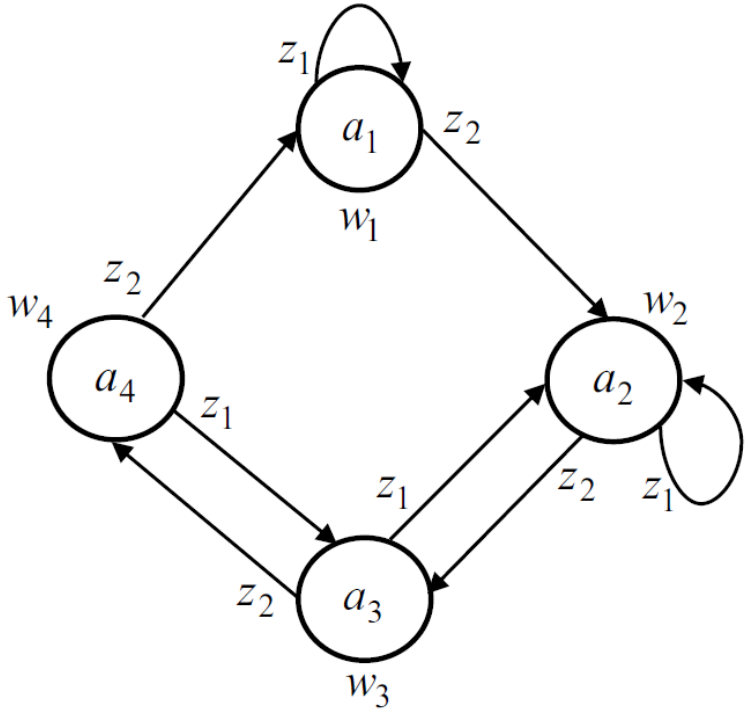
Якщо матрицею з'єднань задається абстрактний автомат Мура, то вихідними сигналами помічаються стани автомата, що ідентифікують рядки матриці з'єднань.

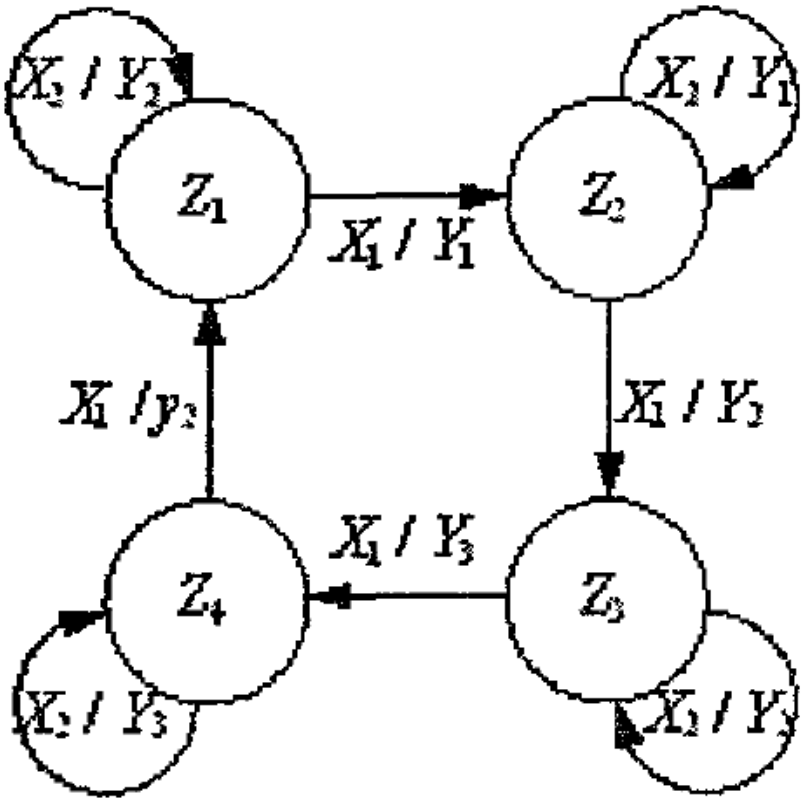
При **графічному методі** автомат задається у вигляді **орієнтованого графа**, **вершини** відповідають **станам**, а **дуги – переходам** між ними. Дуга задає перехід в автоматі зі стану a_m в стан a_s. На початку дуги записується вхідний символ $z_f \in \mathbf{Z}$, що викликає перехід $a_s = \delta(a_m, z_f)$.

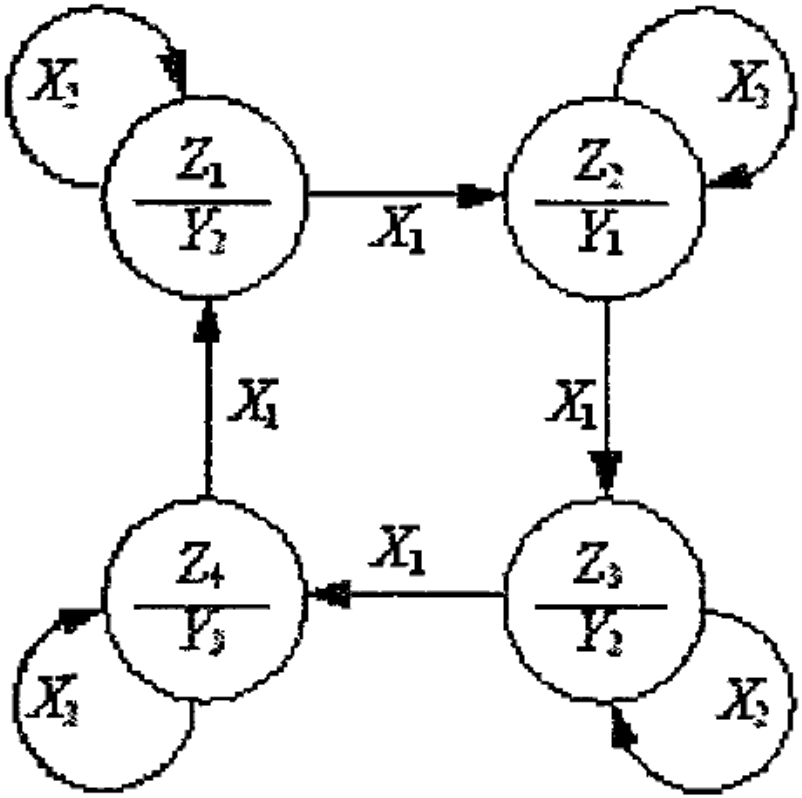
Граф автомата Мілі



Граф автомата Мура




X_2 / Y_2
Z_1
X_1 / Y_1
Z_2
X_2 / Y_1
X_1 / y_2
X_1 / Y_2
X_1 / Y_3
Z_4
Z_3
X_2 / Y_3
X_2 / Y_2


X_2
Z_1 / Y_2
X_1
Z_2 / Y_1
X_2
X_1
X_1
Z_4 / Y_3
X_1
Z_3 / Y_2
X_2
X_2

Зв’язок між автоматами Мілі та Мура

Приклад. Автомат Мілі:

	a_1	a_2	a_3
z_1	a_2	a_3	a_3
z_2	a_3	a_1	a_1

	a_1	a_2	a_3
z_1	w_2	w_1	w_2
z_2	w_2	w_1	w_1

Реакція автомата Мілі

Послідовність станів	a_1	a_2	a_1	a_3	a_3	a_1	a_3
Вхідне слово ξ	z_1	z_2	z_2	z_1	z_2	z_2	
Вихідне слово ω	w_2	w_1	w_2	w_2	w_1	w_2	

Вихідне слово $\omega = \lambda(a_1, \xi)$ називається **реакцією** автомата Мілі в стані a_1 на вхідне слово $\xi = z_1 z_2 z_2 z_1 z_2 z_2$: $\omega = w_2 w_1 w_2 w_2 w_1 w_2$

У відповідь на вхідне слово довжиною k автомат Мілі видасть послідовність станів довжиною k+1 та вихідне слово довжиною k.

У загальному вигляді поведінку автомата Мілі, встановленого в стан a_m, приході вхідного слова $\xi = z_{i1}, z_{i2}, \ldots, z_{ik}$ можна записати у вигляді

Вхідне слово	z_{i1}	z_{i2}	z_{i3}
Послідовність станів	a_m	$a_{i2} = \delta(a_m, z_{i1})$	$a_{i3} = \delta(a_{i2}, z_{i2})$
Вихідне слово	$w_{i1} = \lambda(a_m, z_{i1})$	$w_{i2} = \lambda(a_{i2}, z_{i2})$	$w_{i3} = \lambda(a_{i3}, z_{i3})$

Аналогічно поведінка автомата Мура, що знаходиться в стані a_m:

Вхідне слово	z_{i1}	z_{i2}	z_{i3}	z_{i4}
Послідовність станів	a_m	$a_{i2} = \delta(a_m, z_{i1})$	$a_{i3} = \delta(a_{i2}, z_{i2})$	$a_{i4} = \delta(a_{i3}, z_{i3})$
Вихідне слово	$w_{i1} = \lambda(a_m)$	$w_{i2} = \lambda(a_{i2})$	$w_{i3} = \lambda(a_{i3})$	$w_{i4} = \lambda(a_{i4})$

Для автомата Мура вихідний символ $w_{i1} = \lambda(a_m)$ момент часу i_1 не залежить від вхідного символу z_{i1} та визначається тільки станом w_{i1}. Отже, символ w_{i1} **ніяк не пов'язаний з вхідним словом** ξ. У зв'язку з цим під реакцією автомата Мура, встановленого в стан a_m, на вхідне слово $\xi = z_{i1}, z_{i2}, \ldots, z_{ik}$ розуміють вихідне слово тієї ж довжини $\omega = \lambda(a_m, \xi) = w_{i2} w_{i3} \ldots w_{ik+1}$, зсунуте відносно ξ на один такт.

Приклад. Автомат Мура

	w_1	w_2	w_3	w_4
	a_1	a_2	a_3	a_4
z_1	a_2	a_3	a_4	a_4
z_2	a_4	a_1	a_1	a_1

$\xi = z_1 z_2 z_2 z_1 z_2 z_2$ (те ж саме, що і для попереднього приклада Мілі).

Реакція автомата Мура

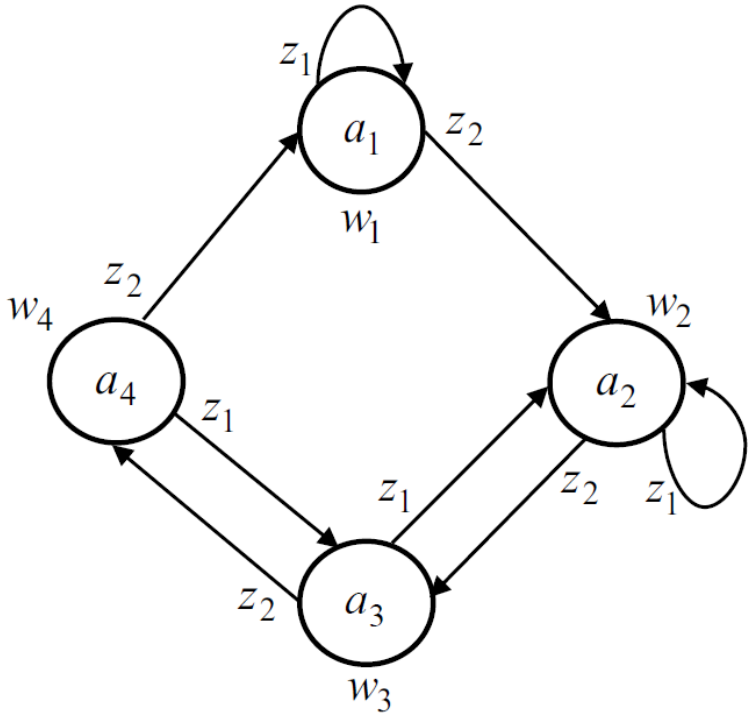
Послідовність станів	a_1	a_2	a_1	a_4	a_4	a_1	a_4
Вхідне слово ξ	z_1	z_2	z_2	z_1	z_2	z_2	
Вихідне слово ω	w_1	w_2	w_1	w_2	w_2	w_1	w_2

Реакції автомата Мілі та автомата Мура на одне й те саме слово ξ співпадають. Отже, автомати Мілі та Мура реалізують одне й те саме перетворення слів вхідного алфавіту і називаються **еквівалентними**.

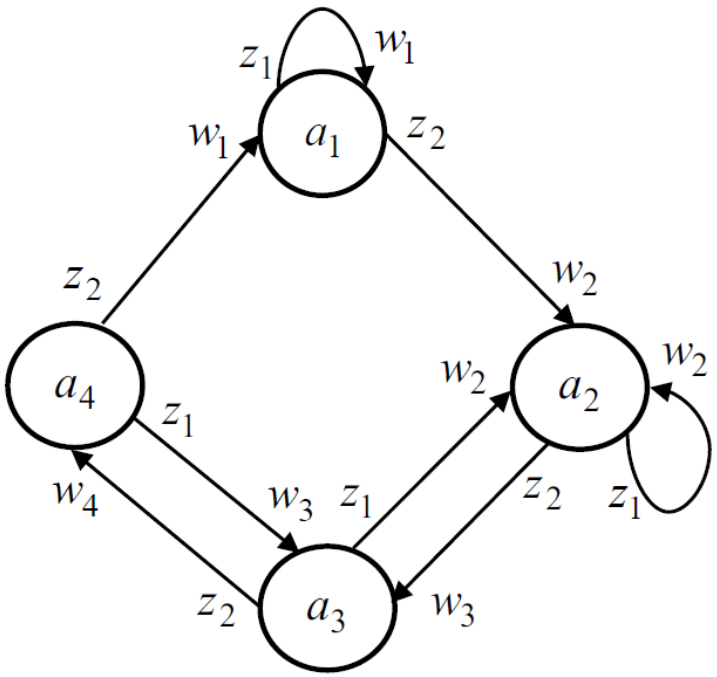
Два автомати **з однаковими вхідними та вихідними алфавітами** називають **еквівалентними**, якщо після встановлення їх у початковий стан їх реакції на будь-яке вхідне слово **співпадають**.

Твердження. Для кожного автомата Мілі A може бути побудований еквівалентний йому автомат Мура B, зокрема у випадку скінченності автомата A автомат B також може бути вибраний скінченним із кількістю станів, що дорівнює $(m+1)n$, де m – кількість вхідних сигналів, а n – кількість станів в автоматі A.

Перехід від автомата Мура до еквівалентного йому автомата Мілі тривіальний та легко здійснюється при графічному методі подання автомата. Для отримання графа автомата Мілі необхідно вихідний сигнал w_g, записаний поряд з вершиною a_s початкового автомата Мура, перенести на всі дуги, що входять у цю вершину. В еквівалентному автоматі Мілі кількість станів така ж, як й в початковому автоматі Мура.



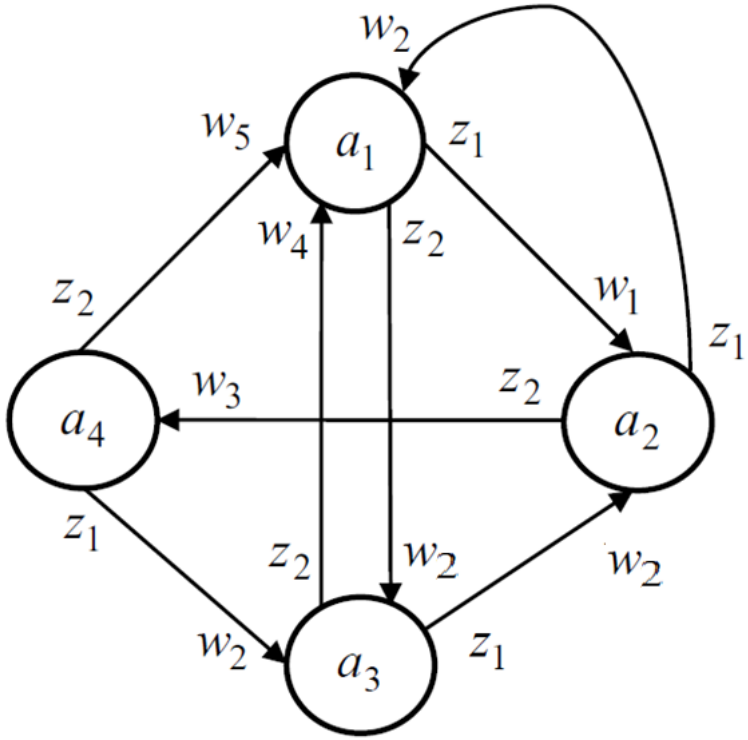
автомат Мура



еквівалентний автомат Мілі

Перехід від автомата Мілі до еквівалентного йому автомата Мура складніший. В автоматі Мура в кожному стані видається тільки один вихідний символ. Перехід найбільш наочно можна зробити при графічному методі задавання автомата. Тоді кожен стан a_i початкового автомата Мілі

породжує стільки станів автомата Мура, скільки різних вихідних сигналів генерується в початковому автоматі при попаданні в стан a_i.



При потраплянні початкового автомата Мілі в стан a_1 формуються вихідні символи w_2, w_4, w_5, при попаданні в стан $a_2 - w_1$, w_2, . $a_3 - w_2$, $a_4 - w_3$.

Кожній парі (a_i, w_j) поставимо у відповідність стан b_k еквівалентного автомата Мура з тим самим вихідним сигналом:

$$w_j : b_1 = (a_1, w_2), \quad b_2 = (a_1, w_4), \quad b_3 = (a_1, w_5),$$

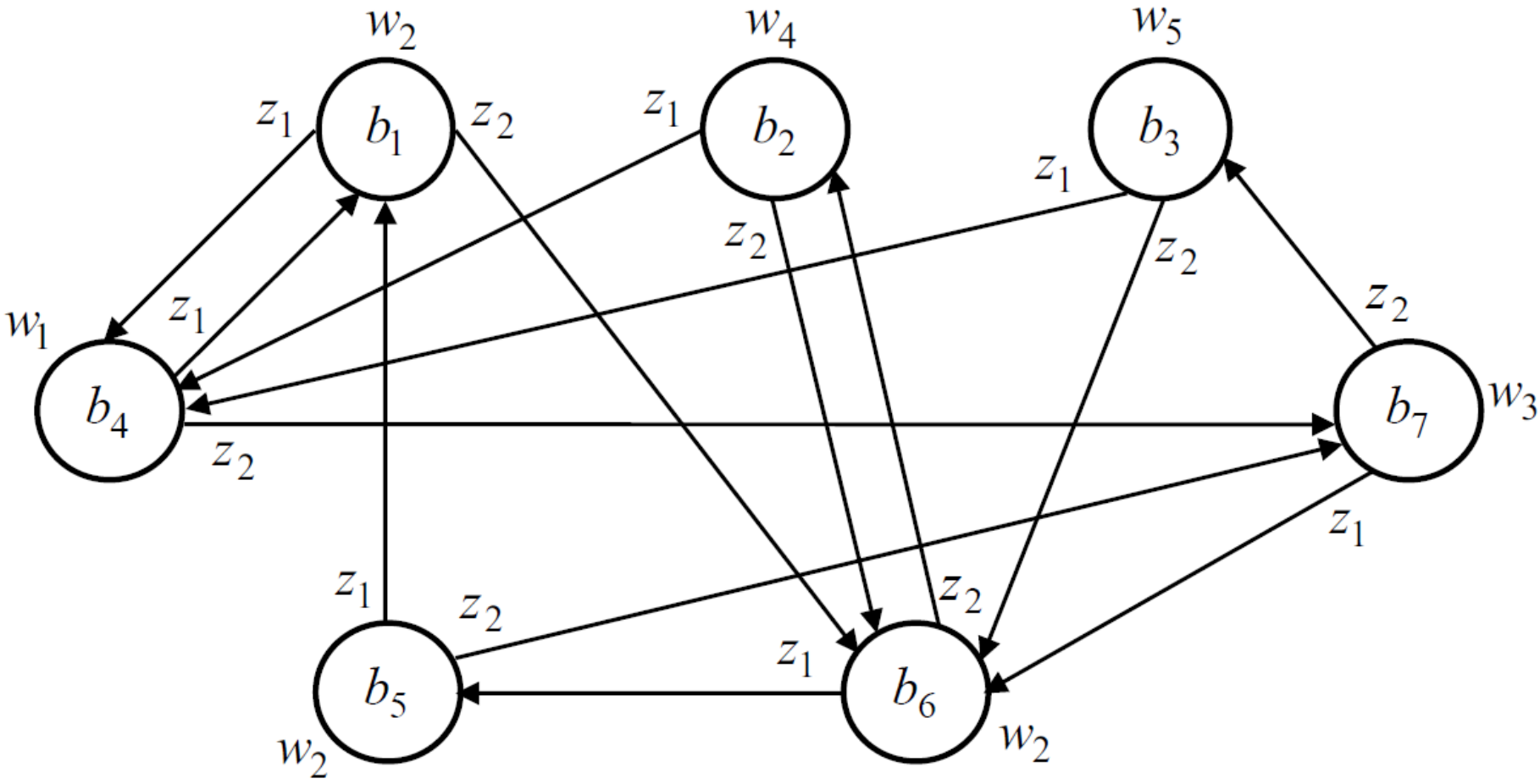
$$b_4 = (a_2, w_1), \quad b_5 = (a_2, w_2), \quad b_6 = (a_3, w_2), \quad b_7 = (a_4, w_3)$$

Кожен стан a_i автомата Мілі породжує деяку множину станів A_i еквівалентного автомата Мура:

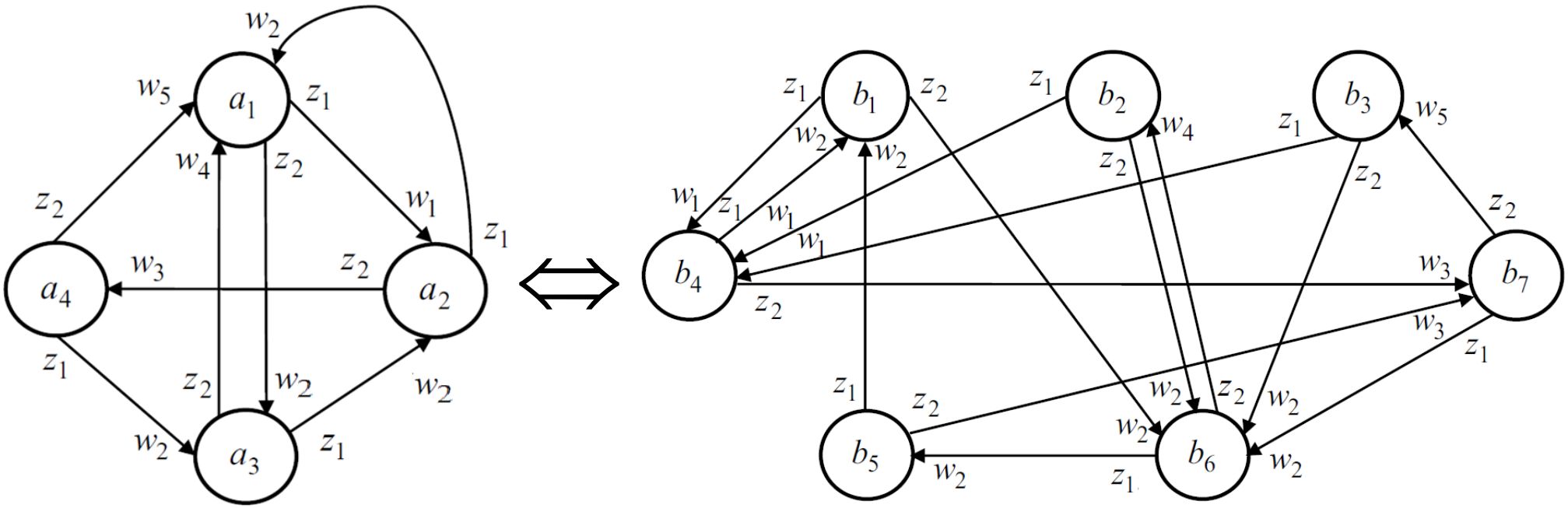
$$\mathbf{A}_1 = \{b_1, b_2, b_3\}, \quad \mathbf{A}_2 = \{b_4, b_5\}, \quad \mathbf{A}_3 = \{b_6\}, \quad \mathbf{A}_4 = \{b_7\}$$

В еквівалентному автоматі Мура кількість станів 7.

Граф автомату Мура , що еквівалентний автомату Мілі



Якщо від цього автомата Мура знову перейти до автомата Мілі, то отримаємо



Два автомати Мілі також будуть **еквівалентними,** але в останнього будуть **на 3 стани більше**. Тобто, **еквівалентні між собою автомати можуть мати різну кількість станів**, у зв'язку з чим виникає задача знаходження **мінімального** (тобто з мінімальною кількістю станів) автомата в класі еквівалентних між собою автоматів.

Лекція 5

- АБСТРАКТНА ТЕОРІЯ ЦИФРОВИХ АВТОМАТІВ
- ЕТАПИ СИНТЕЗУ АВТОМАТІВ
- СТРУКТУРНИЙ СИНТЕЗ ЦИФРОВИХ АВТОМАТІВ

АБСТРАКТНА ТЕОРІЯ ЦИФРОВИХ АВТОМАТІВ

$$\mathbf{S} = \{\mathbf{A}, \mathbf{Z}, \mathbf{W}, \delta, \lambda, a_1\}$$

$\mathbf{A} = \{a_m, m = \overline{1, M}\}$ – множина станів (внутрішній алфавіт);

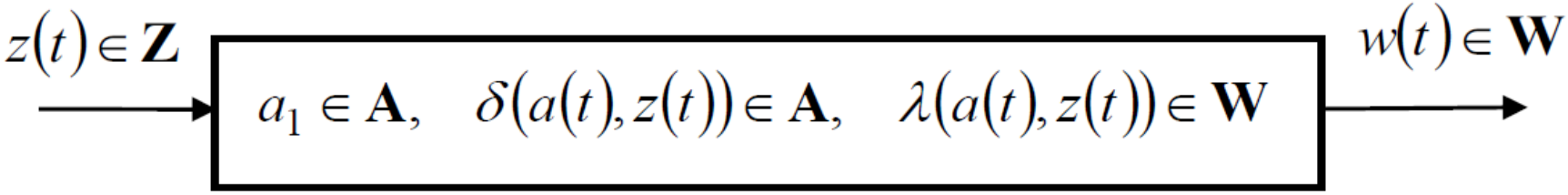
$\mathbf{Z} = \{z_f, f = \overline{1, F}\}$ – вхідний алфавіт, множина вхідних сигналів (символів);

$\mathbf{W} = \{w_g, g = \overline{1, G}\}$ – вихідний алфавіт, множина вихідних сигналів;

$\delta : \mathbf{A} \times \mathbf{Z} \to \mathbf{A}$ – функція переходів $a_s = \delta(a_m, z_f),\ a_s \in \mathbf{A}$;

$\lambda : \mathbf{A} \times \mathbf{Z} \to \mathbf{W}$ – функція виходу, $w_g = \lambda(a_m, z_f),\ w_g \in \mathbf{W}$;

$a_1 \in \mathbf{A}$ – початковий стан автомата.



Закон функціонування **автомата**

Мілі задається рівняннями

$$a(t+1)=\delta(a(t),z(t));$$
$$w(t)=\lambda(a(t),z(t)),\ t=0,1,2\ldots$$

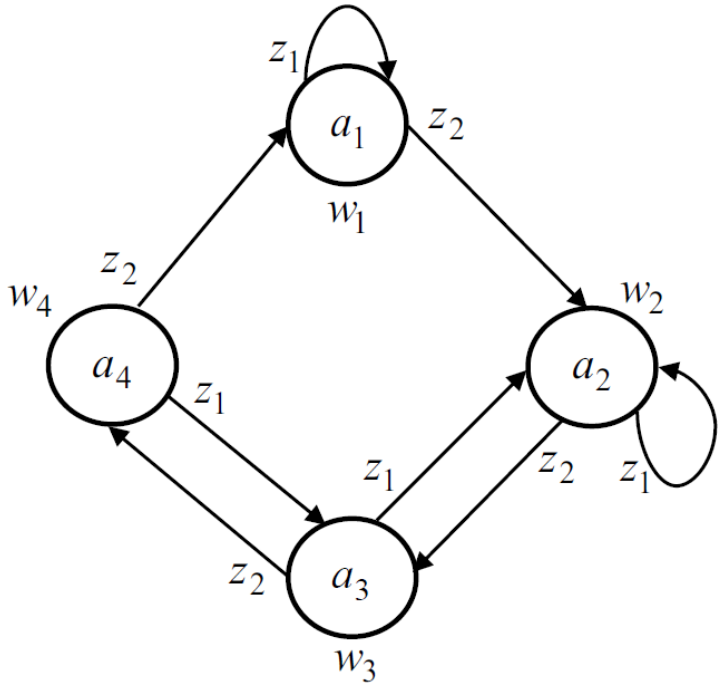
Закон функціонування **автомата**

Мура задається рівняннями

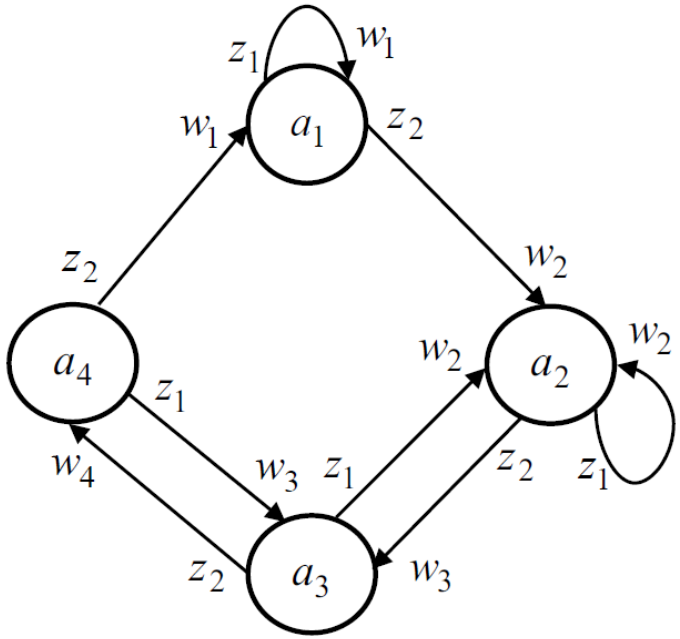
$$a(t+1)=\delta(a(t),z(t));$$
$$w(t)=\lambda(a(t)),\ t=0,1,2\ldots$$

Синтез цифрових автоматів складається з етапів **абстрактного та структурного синтезу**.

Результатом **абстрактного синтезу** автомата є його формальний опис у вигляді $\mathbf{S}=\{\mathbf{A},\mathbf{Z},\mathbf{W},\delta,\lambda,a_1\}$, а структурний синтез дозволяє одержати логічну схему автомата **в заданому елементному базисі.**

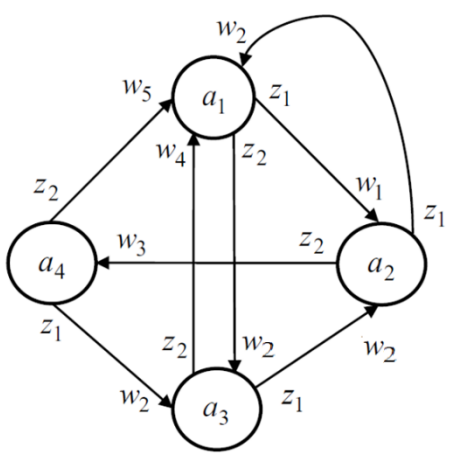


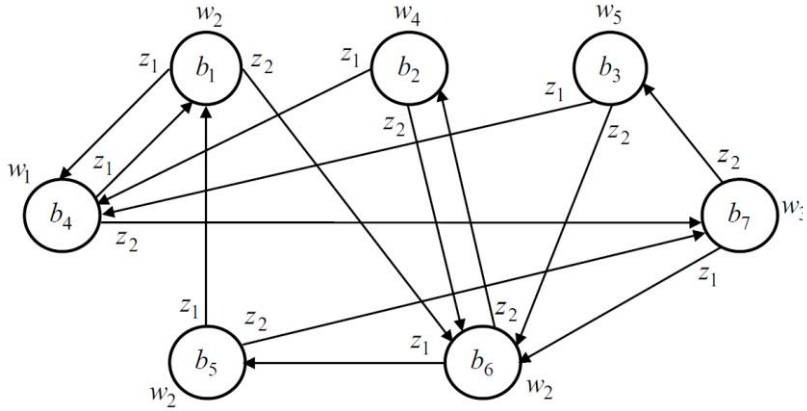
автомат Мура

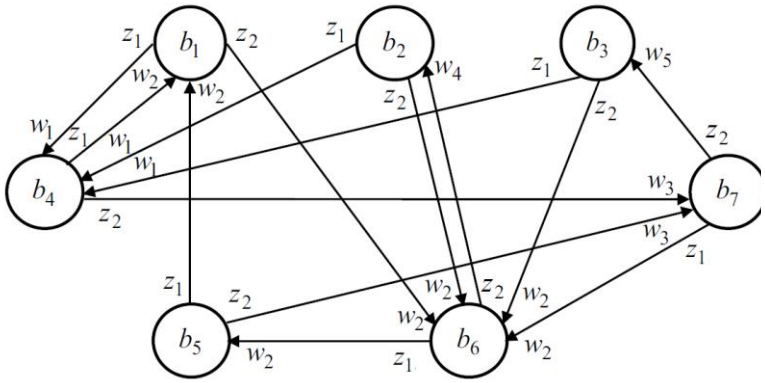


еквівалентний автомат Мілі

Перехід від автомата Мілі до еквівалентного йому автомата Мура складніший.







Мілі 1 Мура Мілі 2

Еквівалентні між собою автомати можуть мати різну кількість станів, тому виникає задача знаходження **мінімального** (з мінімальною кількістю станів) автомата в класі еквівалентних між собою автоматів.

Особливістю графа **асинхронного** автомата є таке: щоб автомат підтримував себе у стійкому стані до того часу, поки прийде новий вхідний сигнал, необхідно вказати дугу, яка **зациклює вершину саму себе з тим самим вхідним сигналом**.

ЕТАПИ СИНТЕЗУ АВТОМАТІВ

На **І** (попередньому) етапі формується (часто словами) умови роботи автомата, тобто визначаються умови його взаємодії з іншими пристроями або об'єктами, **виявляються необхідні вхідні та вихідні сигнали автомата**, їх кількість, загальний закон появи вихідних сигналів залежно від впливу на входи автомата. Синтезі складного автомата розбивають на окремі блоки, тому це **етап блочного синтезу** автомата.

На **ІІ** етапі (**абстрактного синтезу)** виявляють закони функціонування автомата, тобто визначають **функції переходів та виходів**. Опис автомата **має бути представлений одним із прийнятих способів** (матриця, граф). На цьому етапі **не цікавляться фізичними елементами**, з яких має складатися автомат, не розглядається, які конкретні числові значення можуть приймати

вхідні та вихідні сигнали та елементи пам'яті. Важливо знати **кількість можливих різних внутрішніх станів, станів входів і виходів, а також закони зміни внутрішнього стану автомата і вироблення вихідних сигналів при надходженні вхідних сигналів**.

На **III** етапі **мінімізують кількість внутрішніх станів автомата**.

На **IV** етапі синтезу проводиться кодування внутрішніх станів автомата **(розміщення внутрішніх станів)**, вхідні і вихідні сигнали. Після кодування складаються канонічні рівняння. IV етап є **на межі абстрактного і структурного** синтезу автомата.

На **V** етапі **(структурного синтезу)** завершується вибір структури, будується так звана **функціональна схема з комбінаційної частини й автоматів пам'яті**.

VI етап синтезу включає проведення електричного та інших розрахунків елементів схем, складання принципової схеми пристрою та моделювання роботи автомата з метою перевірки його працездатності.

На **VII** етапі складаються монтажні схеми та технічна документація.

Перші п'ять етапів синтезу називають **логічним синтезом автомата (проектуванням)**. Шостий та сьомий - **технічним синтезом** автомата. Поділ процесу синтезу автомата дані сім етапів, з одного боку, полегшує процеси синтезу автомата. З іншого боку, це може призвести до ускладнення структури автомата. Отже, на кожному етапі намагаються врахувати його впливом наступні етапи.

Мінімізація кількості внутрішніх станів повністю визначених абстрактних автоматів

Метод мінімізації повністю визначених автоматів запропонований **Ауфенкампом** та **Хоном**. Ідея методу полягає в розбитті всіх станів початкового автомата на попарно непересічні класи еквівалентних станів та заміні кожного класу еквівалентності одним станом. **Мінімальний автомат** у результаті **має стільки станів, на скільки класів еквівалентності розбиваються стани початкового автомата**.

Два стани абстрактного автомата називають **одноеквівалентними** в тому випадку, якщо реакції автомата в цих станах на будь-які вхідні слова співпадають. Об'єднання всіх одноеквівалентних станів абстрактного автомата утворює **перший клас еквівалентності**.

Одноеквівалентні стани автомата називають **двоеквівалентними**, якщо вони переводяться будь-яким вхідним сигналом також в одноеквівалентні стани. Об'єднання всіх двоеквівалентних станів утворює **другий клас еквівалентності**.

За індукцією, можна дати означення ***i*-еквівалентних** станів та ***i*-класів еквівалентності**.

Якщо для деякого i розбиття станів автомата на $(i+1)$-класи **співпадає** з розбиттям на i-класи, то воно є розбиттям на **∞-класи** еквівалентності.

Розбиття множини внутрішніх станів автомата **на ∞-класи** є необхідним розбиттям на класи еквівалентності, при цьому таке розбиття **може бути отримано за скінченну кількість кроків**. Це стосується **мінімізації автомата Мілі**.

При мінімізації повністю визначених **автоматів Мура** вводиться поняття **0-еквівалентності станів** та розбиття множини станів на **0-еквівалентні класи**, до яких відносяться однаково відзначені стани автомата Мура.

Якщо **два** 0-еквівалентні стани будь-яким вхідним символом переводяться в два 0-еквівалентні стани, то їх називають **одноеквівалентними**. Всі подальші класи еквівалентності станів для автомата Мура означуються аналогічно наведеному для автоматів Мілі.

Приклад мінімізації автомата Мілі

	a_1	a_2	a_3	a_4	a_5	a_6
z_1	a_3	a_4	a_3	a_4	a_5	a_6
z_2	a_5	a_6	a_5	a_6	a_1	a_2

	a_1	a_2	a_3	a_4	a_5	a_6
z_1	w_1	w_1	w_1	w_1	w_1	w_1
z_2	w_1	w_1	w_2	w_2	w_1	w_1

З таблиці виходів отримуємо розбиття на 1-класи еквівалентності π_1, об'єднуючи в еквівалентні класи b_i стани з однаковими значеннями w_i у стовпцях таблиці: $\pi_1 = \{b_1, b_2\}; b_2 = \{a_1, a_2, a_5, a_6\}; b_2 = \{a_3, a_4\}$.

Для отримання 2-еквівалентних станів будуємо **таблицю 1-розбиття**, замінюючи в таблиці переходів стану a_i відповідними класами еквівалентності b_1 або b_2:

	b_1				b_2	
	a_1	a_2	a_5	a_6	a_3	a_4
z_1	b_2	b_2	b_1	b_1	b_2	b_2
z_2	b_1	b_1	b_1	b_1	b_1	b_1

З **1-розбиття** отримуємо 2-класи еквівалентності та розбиття c_i:

$$\pi_2 = \{c_1, c_2, c_3\}\ ,\ \text{де}\ c_1 = \{a_1, a_2\},\ c_2 = \{a_5, a_6\},\ \ c_3 = \{a_3, a_4\}.$$

Порівнюючи π_1 та π_2, відзначаємо, що це розбиття **відрізняється** одне від одного. Тому аналогічно будуємо таблицю 2-розбиття, знову замінюючи в таблиці переходів стану a_i відповідними класами еквівалентності c_i.

	c_1		c_2		c_3	
	a_1	a_2	a_5	a_6	a_3	a_4
z_1	c_3	c_3	c_2	c_2	c_3	c_3
z_2	c_2	c_2	c_1	c_1	c_2	c_2

З отриманої таблиці 2-розбиття знаходимо 3-класи еквівалентності та розбиття d_i: $\pi_3 = \{d_1, d_2, d_3\}$, де $d_1 = \{a_1, a_2\}$, $d_2 = \{a_5, a_6\}$, $d_3 = \{a_3, a_4\}$.

$\pi_3 = \pi_2 \Rightarrow$ **отримано розбиття на ∞-еквівалентні класи.**

Оскільки було отримано всього **три класи**, то мінімальний автомат міститиме всього **три стани**. Вибираємо з кожного класу по одному стану та отримуємо множину станів мінімального автомата **A′**.

Наприклад $\mathbf{A}' = \{a_1, a_3, a_5\}$. Для отримання мінімального автомата з первинних таблиць переходів та виходів викреслюємо стовпці, які відповідають "зайвим станам". У результаті виходить мінімальний автомат Мілі, який еквівалентний початковому автомату:

	a_1	a_3	a_5
z_1	a_3	a_3	a_5
z_2	a_5	a_5	a_1

	a_1	a_3	a_5
z_1	w_1	w_1	w_1
z_2	w_1	w_2	w_1

Мінімізація частково визначених автоматів

Мінімізація недовизначеного автомату S – це пошук автомата S', який серед всіх автоматів, **що покривають** S**,** має найменшу кількість станів.

Нині **відсутній простий алгоритмізований метод**, що дозволяє отримати мінімальний недовизначений автомат. Перебір всіх можливих варіантів стає неможливим навіть для автомата з відносно невеликим числом внутрішніх станів. У зв'язку з складністю є дві тенденції:

1. Розробка наближених, але алгоритмизируемых методів, які **не гарантують** побудову мінімального автомата, але дозволяють запрограмувати цей процес.
2. Розробка методів мінімізації окремих приватних класів недовизначених автоматів, котрим є можливість побудови мінімального автомата.

СТРУКТУРНИЙ СИНТЕЗ ЦИФРОВИХ АВТОМАТІВ

Після етапом абстрактного синтезу автомата, що закінчується мінімізацією числа внутрішніх станів, слідує етап структурного синтезу. Метою його є **побудова схеми автомата із логічних елементів заданого типу.** Якщо абстрактний автомат був лише математичною моделлю дискретного пристрою, то структурному автоматі необхідно враховувати структуру вхідних і вихідних сигналів, а також внутрішню будова автомата на рівні структурної чи функціональної (логічної) схеми. Основним завданням структурної теорії автоматів є **побудова композиції автоматів, тобто методів побудови складних автоматів із простіших автоматів.**

Композиція елементарних автоматів.Нехай задані елементарні автомати S_1, S_2……..S_k. Для об'єднання елементарних автоматів у систему спільно працюючих пристроїв введемо деяку кінцеву множину вузлів, які поділяються на **зовнішні** та **внутрішні**. У композиції спільна робота всіх елементарних автоматів від сигналу, поданого одним із зовнішніх вхідних вузлів, **починається одночасно**. Вхідний сигнал запускає роботу всієї системи загалом, ототожнюючи внутрішні вузли композиції автомата. Після проведених ототожнення всіх вузлів система автомата перетворюється так на звану **схему автоматів** або **мережу автоматів**.

На зовнішні вхідні вузли подається набір **вхідних сигналів (структурний вхідний сигнал схеми)** і всіх зовнішніх вихідних вузлів знімається набір **вихідних сигналів (структурний вихідний сигнал)**.

При побудові схеми автоматів має виконуватися **умова коректності**. Тобто всі елементарні автомати, що входять до композиції, повинні мати **однакові структурні вхідні і вихідні алфавіти і повинні працювати в тому самому автоматному часі**.

Нині найпоширенішим структурним алфавітом є **двійкова система** числення. Для двійкового алфавіту розроблено зручний апарат булевих функцій, що дозволяє робити численні операції над схемами формально.

При побудові структурного автомата попередньо вибираються елементарні автомати, у тому числі шляхом їх композиції будується структурна схема автомата (Милі, Мура чи **С-автомати**).

Нехай задана скінченна множина автоматів та довільний скінченний автомат A в одному й тому ж структурному алфавіті Z . Назвемо автомати з множини **елементарними автоматами** та припустимо, що таких елементарних автоматів є **необмежена кількість екземплярів**. Необхідно знайти алгоритм, що дає змогу за заданим автоматом A **побудувати композицію елементарних автоматів** так, щоб отриманий у результаті композиції автомат індукував відображення, яке **тотожне відображенню**, що індукується автоматом A.

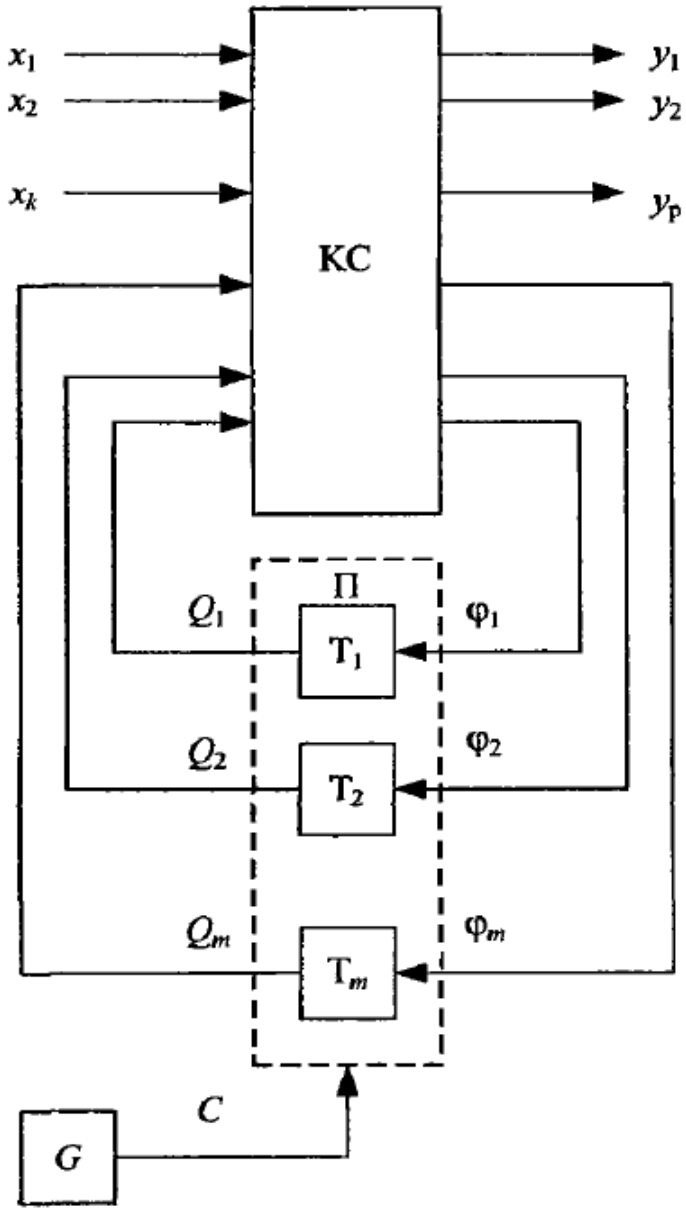
Ця задача має розв'язок **не за будь-якого вибору** системи елементарних автоматів**.** Однак, якщо розв'язок є (для довільного скінченного автомата A), то вважають, що задана система елементарних автоматів **структурно повна** або може мати **ослаблену структурну повноту**.

Елементарні автомати поділяють на **2** класи.

1-й –автомати з пам’яттю (**елементи пам’яті, запам’ятовуючі елементи**, мають більше 1 внутрішнього стану).

2-й - автомати без пам’яті (**комбінаційні** або **логічні елементи** з 1 внутрішнім станом.

Немає ефективних методів (простіших, ніж метод перебору всіх варіантів) розв’язання основної задачі структурного синтезу за довільного вибору структурно повних систем елементарних автоматів.



Канонічний метод зводить структурний синтез **довільних автоматів** до синтезу **автоматів без пам'яті** (використовується для структурно повних систем елементарних автоматів спеціального виду). **Елементами пам'яті** вибираються **автомати Мура** з **повними системами переходів** та **виходів.**

Результатом є **система логічних (канонічних) рівнянь** залежності **вихідних сигналів автомата** і сигналів, що подаються **на входи елементів пам'яті** від сигналів, що приходять на **вхід автомата** і сигналів, що знімаються **з виходів елементів пам'яті**.

Не можна, щоб сигнали на вході елементів пам'яті **безпосередньо** брали участь у формуванні вихідних сигналів. Тому елементами пам'яті повинен бути **автомати Мура**, а не Мілі. **Структурно повна** система елементів автомата **повинна містити хоча б один повний автомат Мура.**

Повнота системи переходів в автоматі Мура означає, що для будь-якої впорядкованої пари станів цього автомата знайдеться вхідний символ, що переводить перший елемент цієї пари в другий. Для повноти системи переходів у цьому автоматі необхідно та достатньо, щоб для будь-якої пари його станів (a_i, a_j) рівняння $a_j = \delta(a_i, z)$ **мало б розв'язок** відносно вхідного сигналу *z*.

Повнота системи виходів в автоматі Мура означає, що кожному стану автомата відповідає свій власний вихідний символ, відмінний від вихідного символу, що відповідає будь-якому іншому стану. Для повноти системи виходів автомата Мура необхідно й достатньо, щоб його зсунута функція виходів $w = \lambda(a)$ здійснювала **взаємно однозначне відображення** множини станів автомата на множину всіх його вихідних символів.

У автоматі Мура з **повною системою виходів** можна ототожнити **внутрішній стан з вихідними символами автомата**. Тоді один і той самий алфавіт застосовується не лише для позначення вхідних та вихідних символів, але й для кодування внутрішніх станів цього автомата.

Теорема про структурну повноту

Будь-яка система елементарних автоматів вважається структурно-повною системою, якщо в цій системі є автомат Мура з нетривіальною пам'яттю, який має повну систему переходів й повну систему виходів та будь-яку функціонально повну систему логічних елементів. Існує загальний конструктивний прийом (канонічний метод структурного синтезу), що дає змогу в даному випадку звести завдання структурного синтезу довільних скінченних автоматів до завдання структурного синтезу КС.

Структурна схема — схема, яка визначає основні функціональні частини виробу, їх взаємозв'язки та призначення. Під функціональною частиною розуміють складову частину схеми: елемент, пристрій, функціональну групу, функціональну ланку.

Структурна схема призначена для відображення **загальної структури** пристрою, тобто його основних блоків, вузлів, частин та головних зв'язків між ними.

Із структурної схеми повинно бути **зрозуміло, навіщо потрібний** даний пристрій і **як він працює** в основних режимах роботи, як взаємодіють його частини. Позначення елементів структурної схеми можуть обиратись довільно, хоча загальноприйнятих правил виконання схем слід дотримуватись.

Кодування внутрішніх станів абстрактних автоматів

Процес кодування станів абстрактних автоматів є першим етапом канонічного методу структурного синтезу автоматів.

Виберемо **запам'ятовуючі елементи** $A_1,\ldots, A_R$, які мають повні системи переходів та виходів, такі, що добуток кількості їх станів не менший, ніж кількість станів автомата A . Кожному стану a_i автомата A поставимо у відповідність скінченну послідовність $(a_1,\ldots,a_r)$ станів автоматів $A_1,\ldots, A_R$ так, що різним станам автомата A ставляться у відповідність різні послідовності. Назвемо цей процес **кодуванням станів** автомата A.

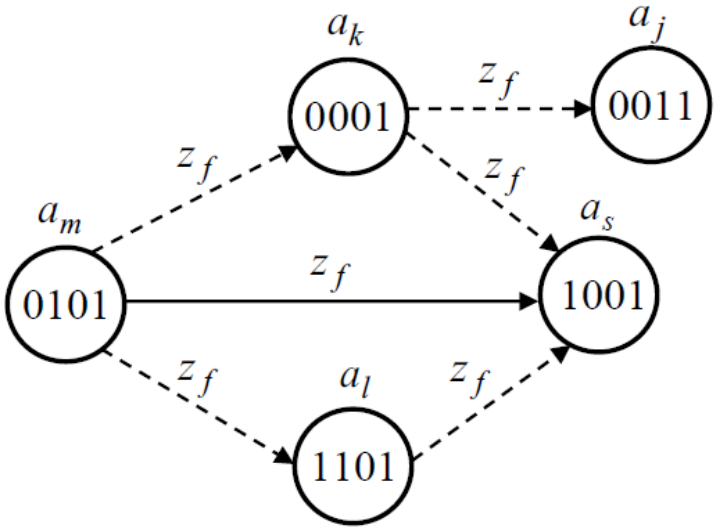
Кодування внутрішніх станів автомата полягає **у зіставленні кожному стану автомата набору значень відповідних станів елементарних автоматів пам'яті**.

Після кодування станів (виконуваного довільним чином) виникають структурні стани автомата. При цьому набори для всіх станів повинні мати однакову довжину, а різним станам автомата повинні відповідати різні набори. Якщо елементи пам'яті двійкові, то їх кількість $R \geq \log_{2|} A$, де A – кількість станів автомата A.

Кодування станів автомата можна здійснювати різними способами. Це може бути **довільне кодування**, коли кожному стану ставиться у відповідність **довільний набір двійкових символів**, кількість яких дорівнює R. Синтезований на основі такого кодування автомат не буде оптимальним, оскільки, по-перше, його КС може мати підвищену складність та, по-друге, за відсутності синхронізації й подвійної пам'яті в процесі функціонування цього автомата можуть виникнути так звані **гонки**.

Гонки (перегони) виникають внаслідок того, що елементи пам’яті мають **різні, хоча йдостатньо близькі часи спрацювання**. Різні також затримки сигналів збудження, що надходять на вхідні канали елементарних автоматів по логічних ланцюгах неоднакової довжини.

Якщо під час переходу автомата з одного стану в інший повинні змінити свої стани **відразу кілька запам’ятовуючих елементів**, то між ними починаються гонки. Той елемент, який виграє ці гонки, тобто змінить свій стан раніше, ніж інші елементи, може через ланцюг зворотного зв’язку змінити сигнали на входах деяких запам’ятовуючих елементів до того, як інші, що беруть участь угонках елементи, змінять свої стани. Це може при призвести до переходу автомата в стан, що **не передбачений** його графом.



Тому в процесі переходу зі стану a_m у стан a_s під дією вхідного сигналу z_f автомат може виявитися в стані a_k або a_l. Якщо потім при тому ж вхідному сигналі z_f автомат з a_k або з a_l перейде в a_s , то такі гонки є **допустимими** або **некритичними.** Якщо ж у цьому автоматі є перехід, наприклад, з a_k в $a_j \neq a_s$ під дією того самого сигналу z_f, то автомат може перейти в a_j, а не в a_s та правильність його роботи буде порушена. Такі гонки називаються **критичними**, що вказує на необхідність вжиття заходів для їх усунення.

Усунути гонки можна **апаратно** або **спеціальними методами кодування**.

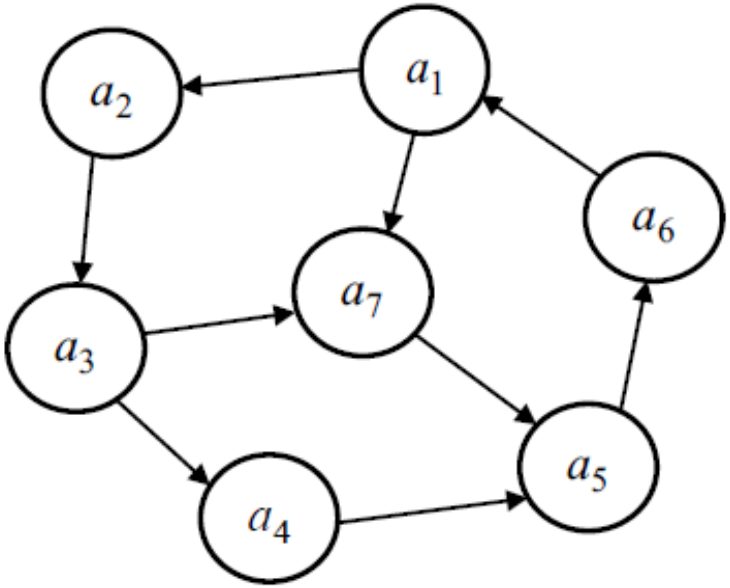
Тактування (стробування) вхідних сигналів автомата **імпульсами** певної тривалості. Окрім вхідних каналів $x_1,\ldots, x_l$ **додається ще канал** C від **генератора синхроімпульсів**, по якому надходить сигнал C=1 у момент приходу імпульсу та C=0 за його відсутності. Тому вхідним сигналом на переході (a_m, a_s) буде не z_f, а Cz_f. Тоді, якщо тривалість імпульсу менше найкоротшого шляху проходження тактового сигналу зворотного зв'язку t_c, то до моменту переходу в проміжний стан a_k сигнал $C=Cz_f=0$, що виключає гонки. **Недолік** – складність підбору **необхідної тривалості імпульсу**.

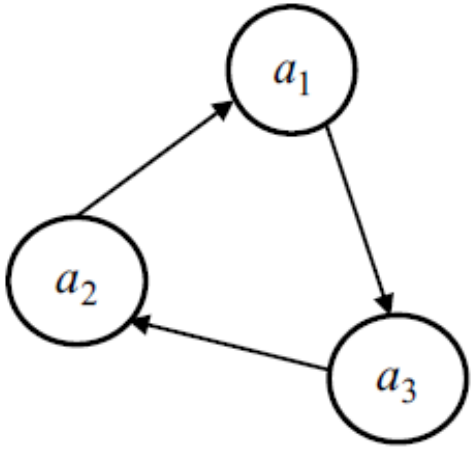
Використанні подвійної пам'яті: кожен елемент пам'яті дублюється, перезапис з першого ступеня в другий йде без синхронізації імпульсу.

Протигоночне кодування – **сусіднє кодування**, гарантує відсутність гонок: будь-які 2 стани, які зв’язані дугою на графі автомата, кодуються наборами, що **відмінні лише на одну одиницю**. Можливе, якщо:

1) у графі автомата не повинно бути циклів з непарною кількістю вершин;

2) 2 сусідні стани 2-о порядку (шлях між якими має 2 ребра незалежно від орієнтації) не повинні мати більше 2-х станів, що лежать між ними.





При сусідньому кодуванні переважно використовують карти Карно.

Стани, які зв’язані дугою, розташовуютьу сусідніх клітинках карти.

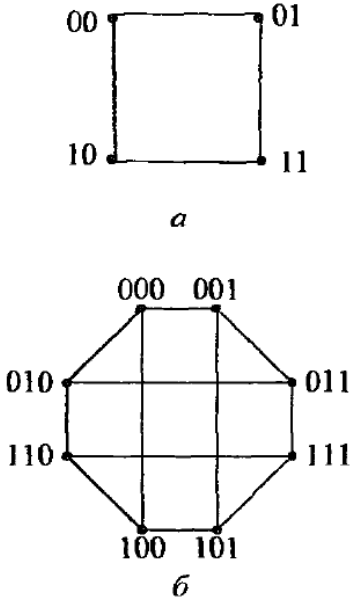
	00	01	11	10
0	a_1	a_2	a_3	a_7
1	a_6		a_4	a_5

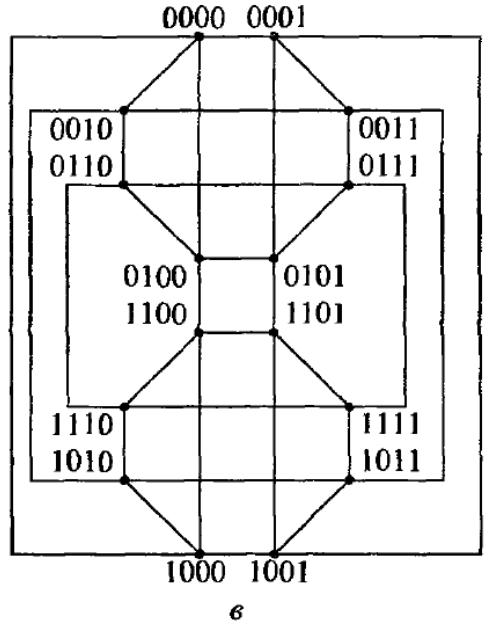
$a_1 = 000$ $a_2 = 010$ $a_3 = 110$ $a_4 = 111$

$a_5 = 101$ $a_6 = 001$ $a_7 = 100$

Таким чином, при сусідньому кодуванні на кожному переході перемикається лише один лемент пам’яті, що усуває гонки.

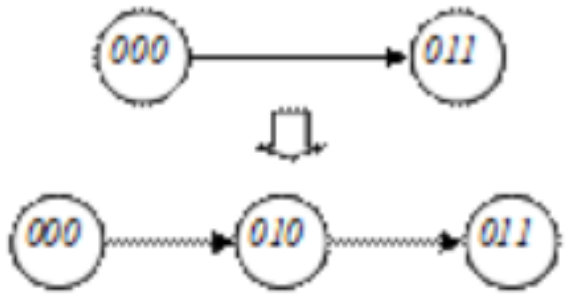
Можна також використовувати шаблони для кодування станів з різним числом бітів.





Або розв'язувати кілька кодів.

Тобто, між двома станами, закодованими не сусідніми кодами, додавали стан з кодом, який був сусіднім по відношенню до цих обох кодів.



Якщо автомат має l станів, то кодування сусідніми наборами при використанні $h = log_2 l$ внутрішніх змінних у більшості випадків виконати неможливо, тому в процесі кодування розширюють алфавіт станів автомата та **вводять додаткові нестійкі стани**, які забезпечують реалізацію всіх переходів у схемі зі зміною тільки одного елементу пам'яті.

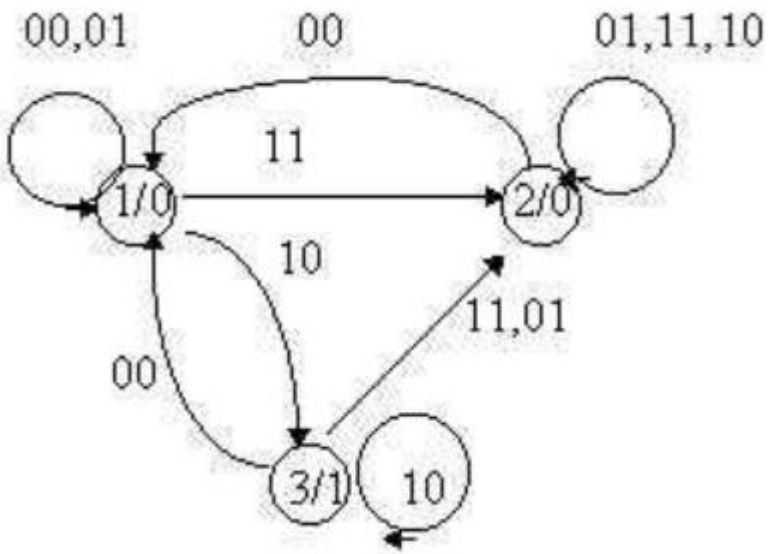
Універсальний спосіб кодування дозволяє за допомогою введення додаткових станів виконати кодування будь-якого асинхронного автомата. Для кодування автомата, що має m станів, потрібно m елементів пам'яті. Кожному стану приписується код, що містить одну одиницю та інші нулі. Ця одиниця розташована у позиції, **номер якої визначається номером стану**. Для кожної пари станів s_i та s_j, між якими існує хоча б один перехід, вводиться додатковий нестійкий стан, якому приписується код, що містить 2 одиниці в позиціях, що визначаються номерами станів s_i та s_j. Такий додатковий стан може бути використаний для всіх переходів між станами s_i та s_j, оскільки вони повинні виконуватися при дії **різних вхідних сигналів**. Введення таких додаткових станів завжди можливе, оскільки **коди з двома одиницями не використовуються** для кодування основних станів.

Приклад. Кодування станів автомата Мура А1, заданого табл.

Таблиця входів і виходів

w	s_i	p_k			
		00	01	11	10
0	1	(1)	(1)	2	3
0	2	1	(2)	(2)	(2)
1	3	1	2	2	(3)

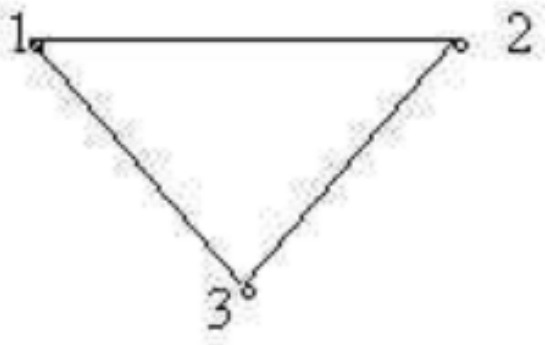
Граф автомату



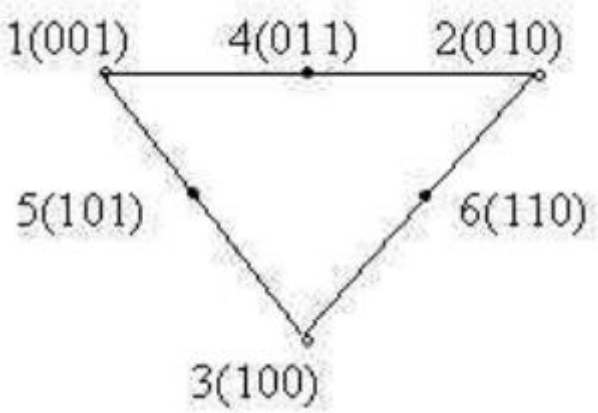
Припишемо вузлам графа зв'язків коди з однією одиницею і одержуємо граф, у якому кожному переході змінюються **внутрішні змінні**, що створює умови для змагань.

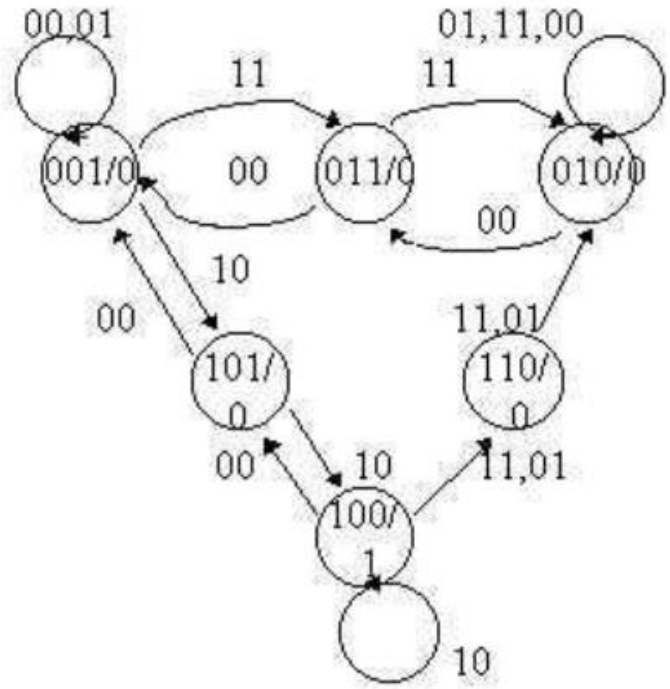
Щоб побудувати граф зв'язків, що гарантує відсутність змагань, введемо додаткові стани в кожне ребро та припишемо коди **з 2 одиницями** (додаткові стани позначені на цьому малюнку темними кружальцями).

Граф зв’язків автомату



Граф зв’язків автомату з додатковими станами





Кодована таблиця переходів автомата A1, що відповідає побудованому графу зв'язків, може бути представлена у вигляді

s_i	p_k			
	00	01	11	10
001	(001)	(001)	001	101
010	011	(010)	(010)	(010)
100	101	110	110	(100)
011	001	-	010	-
101	001	-	-	100
110	-	010	010	-

Лекція 6

АБСТРАКТНА ТЕОРІЯ ЦИФРОВИХ АВТОМАТІВ

ЕТАПИ СИНТЕЗУ АВТОМАТІВ. СТРУКТУРНИЙ СИНТЕЗ ЦИФРОВИХ АВТОМАТІВ

ТРИГЕРИ

АБСТРАКТНА ТЕОРІЯ ЦИФРОВИХ АВТОМАТІВ

$$\mathbf{S} = \{\mathbf{A}, \mathbf{Z}, \mathbf{W}, \delta, \lambda, a_1\}$$

$\mathbf{A} = \{a_m, m = \overline{1, M}\}$ – множина станів (внутрішній алфавіт);

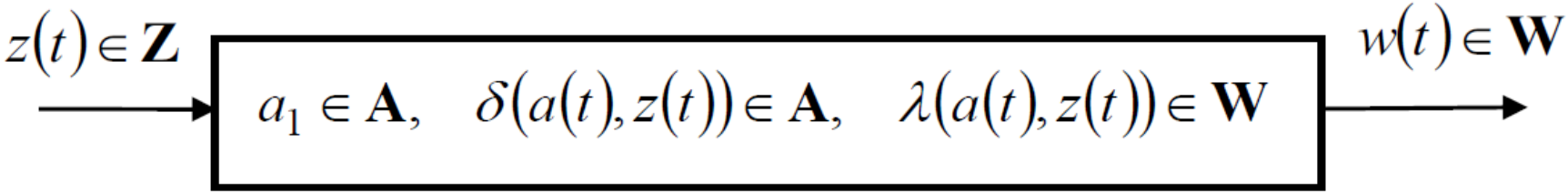
$\mathbf{Z} = \{z_f, f = \overline{1, F}\}$ – вхідний алфавіт, множина вхідних сигналів (символів);

$\mathbf{W} = \{w_g, g = \overline{1, G}\}$ – вихідний алфавіт, множина вихідних сигналів;

$\delta : \mathbf{A} \times \mathbf{Z} \to \mathbf{A}$ – функція переходів $a_s = \delta(a_m, z_f),\ a_s \in \mathbf{A}$;

$\lambda : \mathbf{A} \times \mathbf{Z} \to \mathbf{W}$ – функція виходу, $w_g = \lambda(a_m, z_f),\ w_g \in \mathbf{W}$;

$a_1 \in \mathbf{A}$ – початковий стан автомата.



ЕТАПИ СИНТЕЗУ АВТОМАТІВ

I (попередній) **етап блочного синтезу** автомата.

II етап (**абстрактного синтезу) -**визначають **функції переходів та виходів**.

На **III** етапі **мінімізують кількість внутрішніх станів автомата**.

На **IV** етапі синтезу проводиться кодування внутрішніх станів автомата **(розміщення внутрішніх станів),** вхідні і вихідні сигнали.

На **V** етапі (**структурного синтезу)** будується **функціональна схема з комбінаційної частини й автоматів пам'яті**.

VI етап проведення електричного та інших розрахунків елементів схем.

На **VII** етапі складаються монтажні схеми та технічна документація.

СТРУКТУРНИЙ СИНТЕЗ ЦИФРОВИХ АВТОМАТІВ

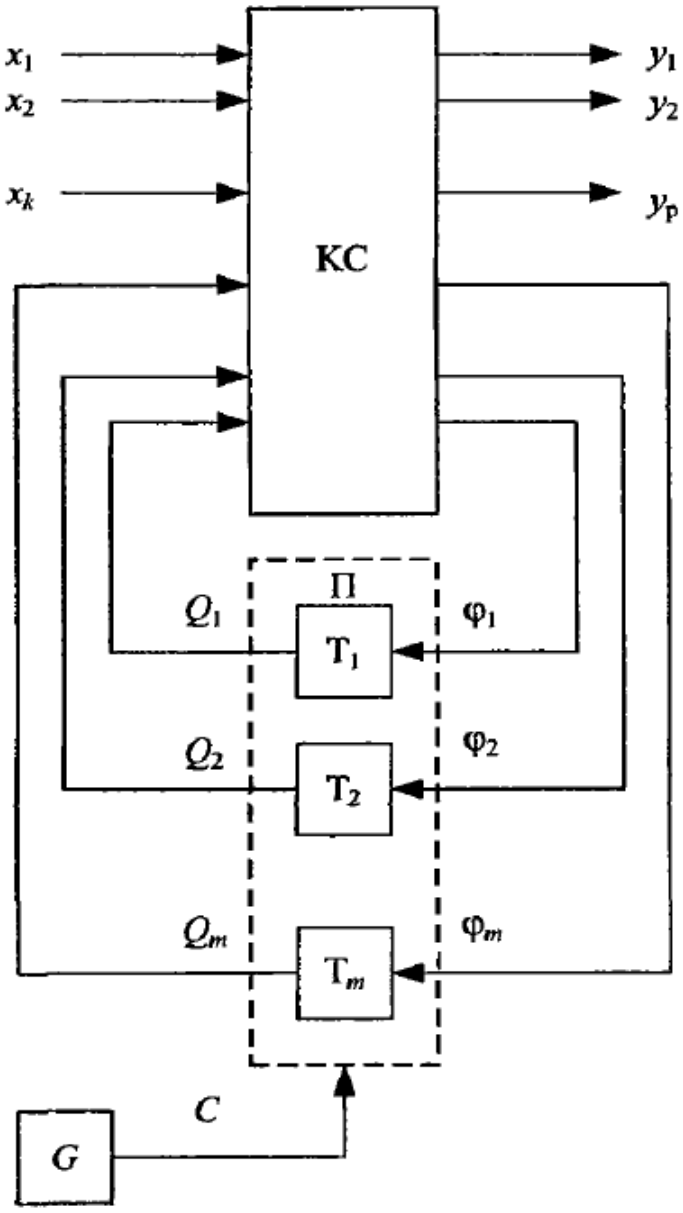
Після етапом абстрактного синтезу автомата, що закінчується мінімізацією числа внутрішніх станів, слідує етап структурного синтезу. Метою його є **побудова схеми автомата із логічних елементів заданого типу.** Якщо абстрактний автомат був лише математичною моделлю дискретного пристрою, то структурному автоматі необхідно враховувати структуру вхідних і вихідних сигналів, а також внутрішню будова автомата на рівні структурної чи функціональної (логічної) схеми. Основним завданням структурної теорії автоматів є **побудова композиції автоматів, тобто методів побудови складних автоматів із простіших автоматів.**

Елементарні автомати поділяють на **2** класи.

1-й –автомати з пам’яттю (**елементи пам’яті, запам’ятовуючі елементи**, мають більше 1 внутрішнього стану).

2-й - автомати без пам’яті (**комбінаційні** або **логічні елементи** з 1 внутрішнім станом.

Немає ефективних методів (простіших, ніж метод перебору всіх варіантів) розв’язання основної задачі структурного синтезу за довільного вибору структурно повних систем елементарних автоматів.



ТРИГЕРИ

Комбінаційна схема зі зворотними зв'язками, що має **два стійких стани** та призначена для записування та **зберігання одного біта** інформації, називається **елементарним автоматом** або **тригером**.

Під дією вхідних сигналів тригер переходить з одного стійкого стану в інший. При цьому напруга на його виході стрибкоподібно змінюється. Для коректної роботи цифрових автоматів необхідно виключити вплив перехідних процесів в тригерах та КС на зміну станів цифрового автомата та на вихідний сигнал. Ця вимога виконується при використанні складної багатофазної системи синхронізуючих сигналів для блока пам'яті та вихідної КС.

Як правило, тригер має **два виходи – прямий та інверсний**. Кількість входів залежить від структури та функцій, що виконуються тригером.

За **логічним функціонуванням** розрізняють тригери типів *RS*, *D*, *T*, *JK*, та ін. Окрім того, використовуються комбіновані тригери, в яких поєднуються одночасно декілька типів, та тригери зі складною вхідною логікою (групами входів, пов'язаних між собою логічними залежностями).

За **способом записування** інформації **є асинхронні** та **синхронні** тригери.

В **асинхронних** тригерах інформація може записуватися **неперервно** та **визначається інформаційними сигналами**, які діють на входах у даний момент часу. Якщо інформація записується в тригер **тільки в момент дії синхронізуючого сигналу**, то такий тригер називають **синхронним**.

Окрім інформаційних входів, синхронізовані тригери мають **вхід синхронізації (тактовий вхід)**. За способом сприйняття тактових сигналів тригери поділяються керовані **рівнем** та керовані **фронтом**.

Керування рівнем означає, що при одному рівні тактового сигналу тригер сприймає вхідні сигнали та реагує на них, а при іншому не сприймає та лишається в незмінному стані.

При керуванні фронтом дозвіл на перемикання дається тільки у момент перепаду тактового сигналу (на його фронті або спаді). В інший час незалежно від рівня тактового сигналу тригер не сприймає вхідні сигнали та залишається в незмінному стані. Тригери, що керуються фронтом, називаються тригерами з **динамічним керуванням**.

Динамічний вхід може бути **прямим** або **інверсним**. **Прямий** – дозвіл на перемикання при зміні тактового сигналу **з нульового значення на одиничне**, **інверсний** – при зміні **з одиничного на нульовий**.

За характером процесу переключення тригери поділяються на **одноступеневі** та **двоступеневі**. В одноступеневому тригері переключення в новий стан відбувається **одразу**, а в двоступеневому – **за етапами**. Двоступеневі тригери складаються з **вхідної** та **вихідної** сходинки. Перехід в новий стан відбувається в обох сходинках одночасно. Один із рівнів тактового сигналу дозволяє приймати інформацію у вхідну сходинку при незмінному стані вихідної сходинки. Другий рівень тактового сигналу дозволяє передачу нового стану з вхідної сходинки у вихідну.

Двоступеневі тригери позначаються двома буквами *Т*.

У цифровій техніці прийняті такі позначення **входів** та **виходів** тригерів:

- Q – прямий вихід тригера;
- $\overline{Q}$ – інверсний вихід тригера;
- S – роздільний вхід встановлення в одиничний стан (висока напруга на Q);
- R – роздільний вхід встановлення в нульовий стан (низька напруга на Q);
- D – інформаційний вхід (на нього подається інформація, яка призначена для занесення в тригер);
- C – вхід синхронізації;
- T – лічильний вхід.

Найбільше розповсюдження в цифрових пристроях отримали RS -тригер з двома стійкими входами, D -тригер, лічильний T-тригер та JK-тригер.

Асинхронний *RS*-тригер

Має 2 вхідних канали: вхід *S* (**set**) – вхід встановлення в одиницю, вхід *R* (**reset**) встановлення в нуль.

Таблиця переходів

R_t	S_t	Q_t	Q_{t+1}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	-
1	1	1	-

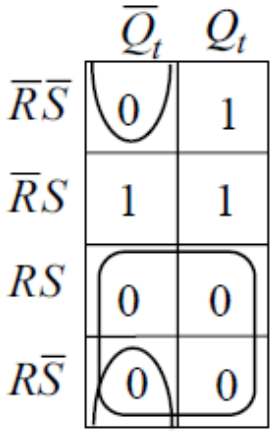
R_t, S_t, Q_t – значення логічних змінних у момент часу t на входах R, S та виході Q;

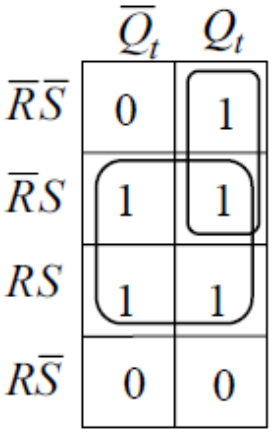
Q_{t+1} – стан тригера після перемикання;

“-” – невизначений стан на тих наборах, де вхідні сигнали R_t та S_t одночасно набувають значення 1 (заборонена комбінація сигналів).

Неповністю визначена карта Карно

	$\overline{Q}_t$	Q_t
$\overline{R}\overline{S}$	0	1
$\overline{R}S$	1	1
RS	-	-
$R\overline{S}$	0	0





Логічні рівняння асинхронного *RS*-тригера:

$$Q_{t+1} = \overline{R}_t (S_t + Q_t),$$

$$Q_{t+1} = S_t + \overline{R}_t Q_t.$$

Для спрощення індекс *t* у правій частині логічного виразу далі опускаємо.

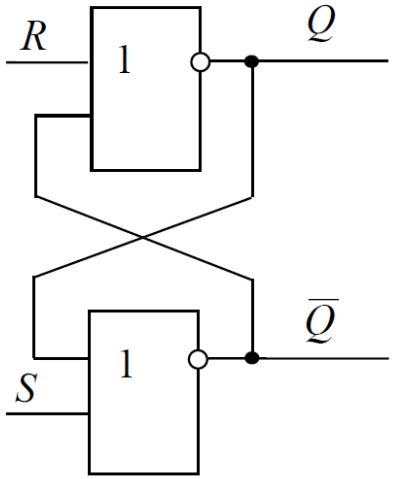
Розрізняють *RS*-тригери з **прямими** та **інверсними** входами.

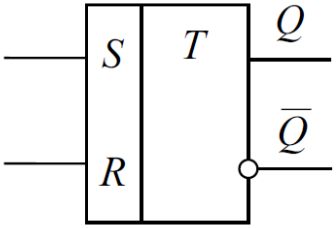
Активним називають рівень на вході елемента що **однозначно визначає рівень вихідного сигналу (незалежно від рівнів на інших входах)**.

АБО-НІ – активний **високий рівень**, І-НІ – активний **низький рівень**. Рівні, подача яких на один із входів **не призводить до модифікації** логічного рівня на виході елемента, називають **пасивними**.

З прямими входами

$$Q_{t+1} = \overline{\overline{R}(S_t + Q_t)} = \overline{R + \overline{(S_t + Q_t)}}$$

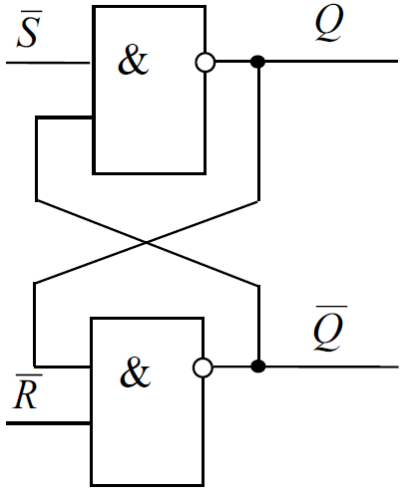


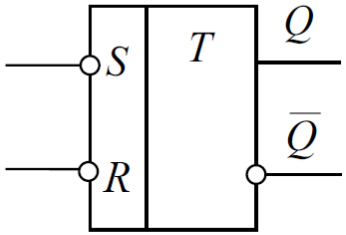


S	R	Q_t	Q_{t+1}	
0	0	Q	Q	зберігання інформації
0	1	*	0	установка «0»
1	0	*	1	установка «1»
1	1	*	-	заборонена

З інверсними входами

$$Q_{t+1} = \overline{\overline{S + \overline{R}Q}} = \overline{\overline{S} \cdot \overline{\overline{R} \cdot Q}}$$



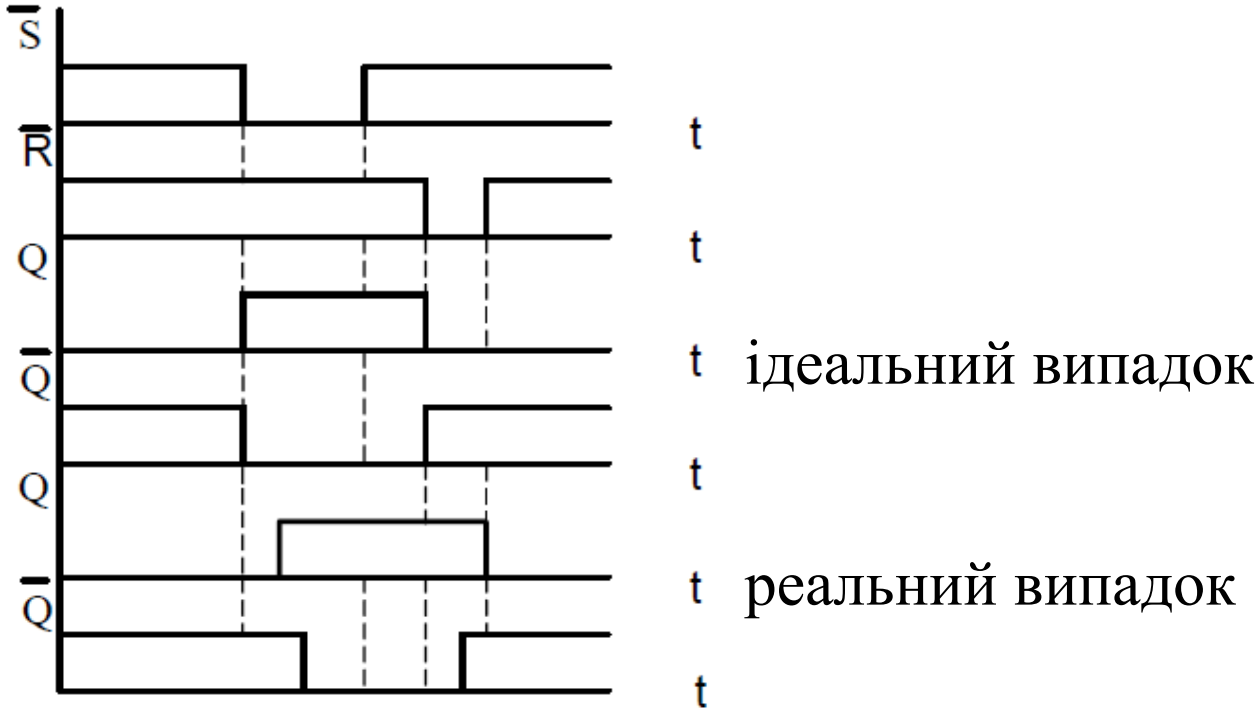


S	R	Q_t	Q_{t+1}	
1	1	Q	Q	зберігання інформації
0	1	*	0	установка «1»
1	0	*	1	установка «0»
0	0	*	-	заборонена

Q^t – рівні на виході тригера **до подачі на його входи активних рівнів**.

Q^{t+1} – рівні на виході тригера **після подачі інформації на його входи**.

Часова діаграма переходів

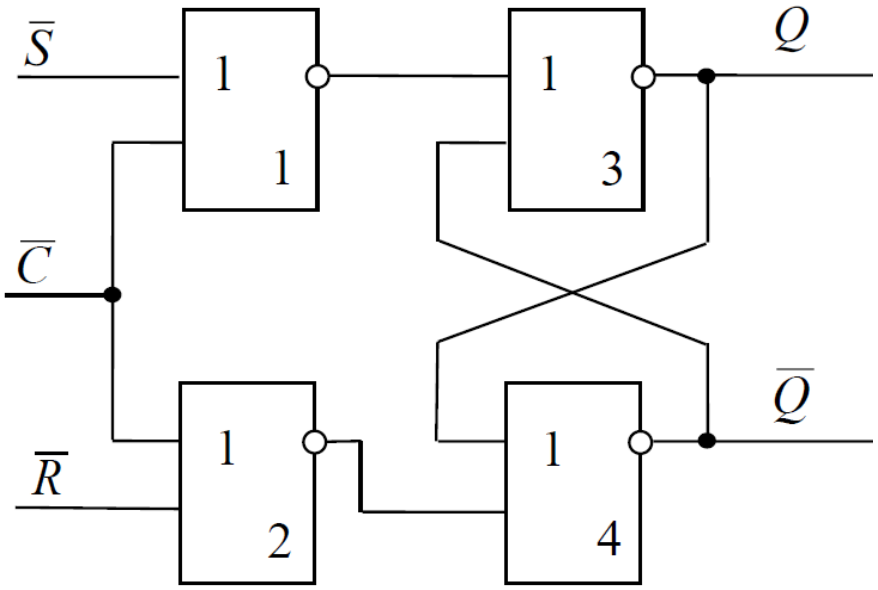


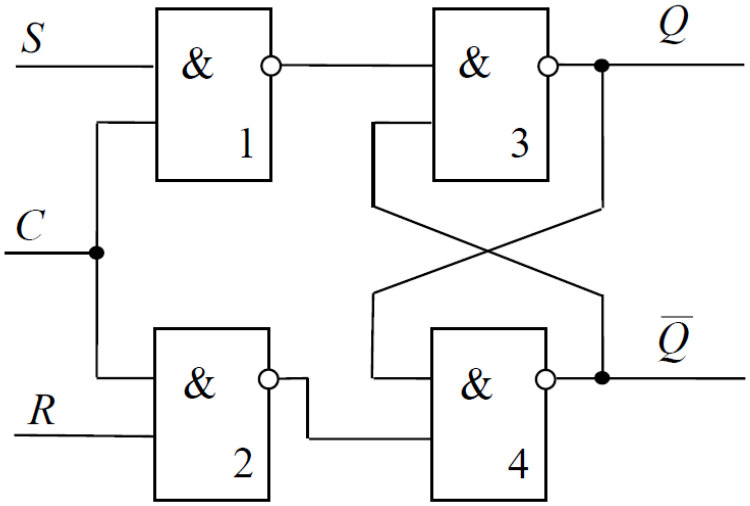
Відображено умовну затримку сигналу τ на одному логічному елементі при розгляді реальних схем (пунктирні лінії).

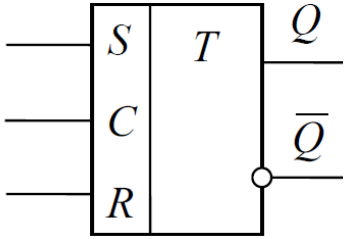
Синхронний *RS*-тригер

Схема *RS*-тригера дає змогу запам'ятовувати стани комбінаційної схеми, але оскільки в початковий момент часу може виникати **перехідний процес (гонки)**, то запам'ятовувати стани комбінаційної схеми потрібно тільки у визначені моменти часу, коли всі перехідні процеси завершені. Це означає, що більшість цифрових схем потребують **сигналу синхронізації (тактового сигналу)**. Всі перехідні процеси в комбінаційній схемі повинні закінчуватися за час періоду **синхросигналу**, що подається на входи тригерів. При сигналі синхронізації C=0 відбувається зберігання інформації, а при C=1 – переходи відповідно до таблиці для асинхронного *RS*-тригера. Для побудови **синхронного однотактного *RS*-тригера** на елементах І-НІ потрібно замінити в логічному виразі S та R на добутки CS та CR

$$Q_{t+1} = = \overline{\overline{CR} + \overline{(\overline{CS} + Q)}} = \overline{\overline{C} + \overline{R} + \overline{(\overline{C} + \overline{S} + Q)}}$$







$$Q_{t+1} = \overline{\overline{CS} \cdot \overline{\overline{CR} \cdot Q}}$$

Елементи 1 та 2 складають **схему керування** з прямими чи інверсними входами, а елементи 3 та 4 утворюють **фіксатор – елемент пам'яті тригера** на двох інверторах і вихід одного з'єднаний з входом іншого.

Таблиця переходів синхронного *RS*-тригера

C	R	S	Q^t	Q^{t+1}	Примітка
0	X	X	0	0	Режим зберігання інформації
0	X	X	1	1	
1	0	0	0	0	Режим зберігання інформації
1	0	0	1	1	
1	0	1	0	1	Режим встановлення одиниці $S=1$
1	0	1	1	1	
1	1	0	0	0	Режим записування нуля $R=1$
1	1	0	1	0	
1	1	1	0	-	$R=S=1$ заборонена комбінація
1	1	1	1	-	

Робота тригера може описуватися **таблицею функцій входів**, в якій вказується, **які набори керуючих сигналів** у поточний момент автоматного часу необхідно подати **на входи** тригера, щоб у наступний момент автоматного часу він перейшов з поточного вказаного стану в **новий вказаний стан**. Таблиця функцій входів будується за таблицею переходів для тригера.

Таблиця функцій входів синхронного *RS*-тригера

Q^t	Q^{t+1}	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

Із таблиці можна отримати **функцію збудження пам'яті автомата при синтезі на базі асинхронного *RS*-тригера з прямими входами**. Наприклад, якщо деякий автомат, що містить у своєму складі три *RS*-тригери, переходить зі 010 стану у стан 110, то для забезпечення такого переходу функції збудження повинні бути:

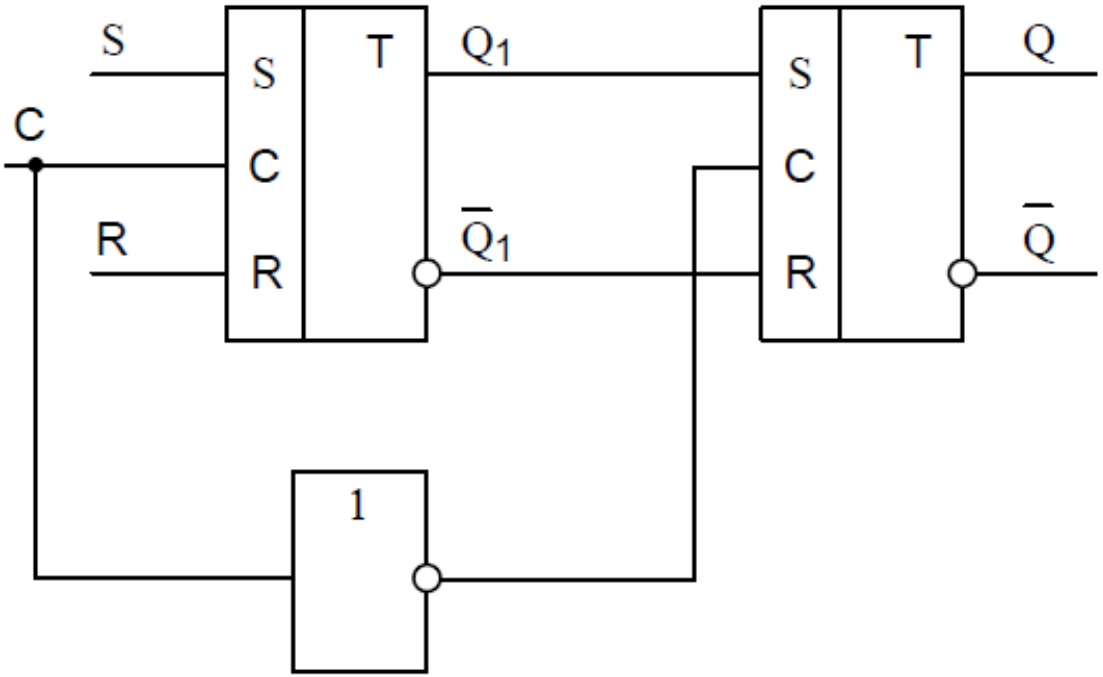
- для першого тригера при переході з 0 в 1 – $R_1 = 0,\ \ S_1 = 1,$
- для другого тригера при переході з 1 в 1 – $R_2 = 0,\ \ S_2 = X,$
- для третього тригера при переході з 0 в 0 – $R_3 = X,\ S_3 = 0.$

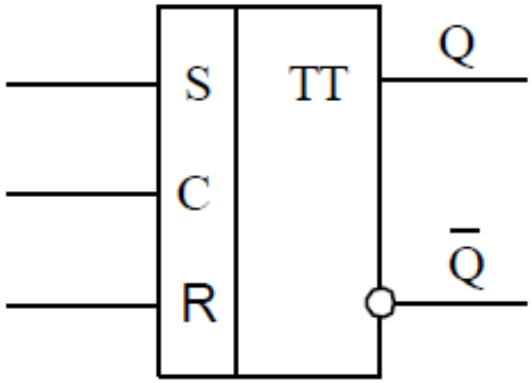
Двотактний RS-тригер

Стійка робота однотактних *RS*-тригерів у довільній схемі можлива тільки у випадку, якщо занесення в тригер інформації здійснюється **після завершення передачі інформації про колишній його стан в інший тригер**. Це досить просто забезпечується при використанні **двох серій синхросигналів**, що знаходяться **в протифазі**. Такий принцип обміну інформацією реалізовано у **двотактних** *RS*-тригерах.

Найпростіша схема двовходового **двотактного** *RS*-тригера складається з **двох однотактних** *RS*-тригерів **та інвертора** в ланцюзі синхронізації. При надходженні на вхід *RS*-тригера сигналу C=1 вхідна інформація заноситься в перший однотактний *RS*-тригер, а другий при цьому зберігатиме інформацію, що відноситься до попереднього періоду подання.

Синхронний двотактний RS-тригер



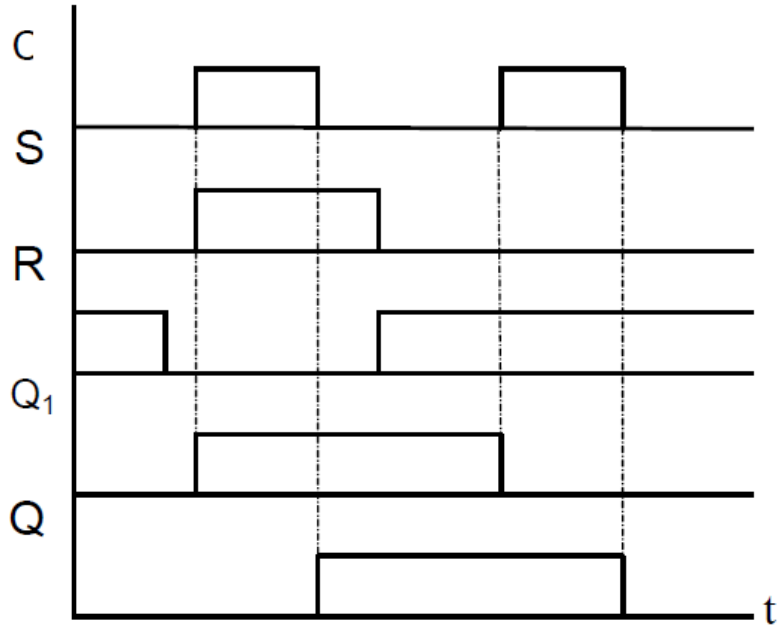


Після закінчення дії сигналу синхронізації, коли C=0, а $\overline{C}$ =1 **перший** *RS*-тригер **перейде в режим зберігання**, а **другий перепише** з нього **нове значення вихідного сигналу**.

Двотактний тригер змінює свої стани лише після закінчення дії сигналу синхронізації C=1 (перехід у режим зберігання інформації).

Тому **з двотактних тригерів можна будувати довільні схеми, зокрема подавати сигнали з виходу тригера з його вхід**.

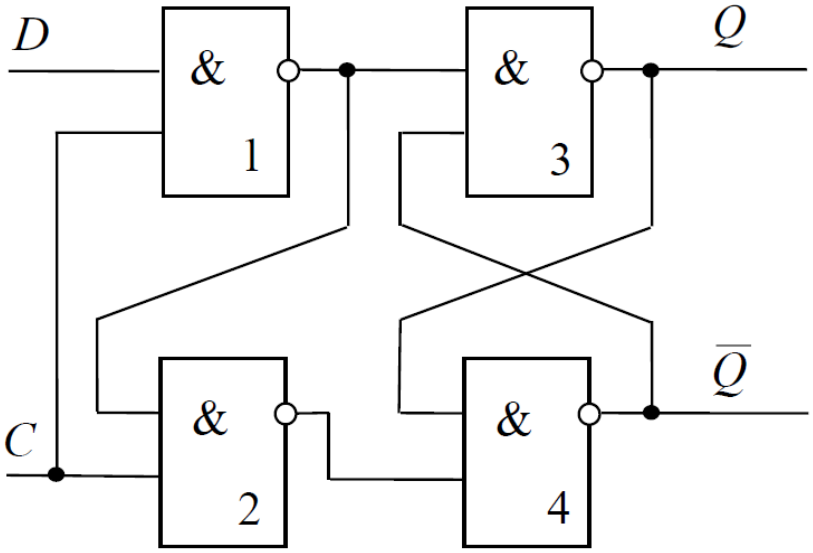
Часова діаграма перемикань двотактного тригера

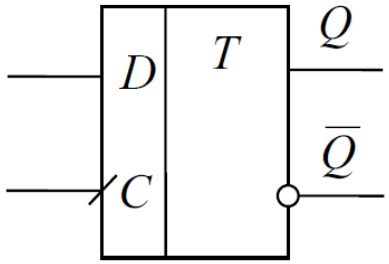


Схеми *RS*-тригерів становлять основу для побудови інших тригерних схем, таких як *T*-, *D*- і *JK*-тригери.

D-тригер

Тригером типу *D* називається запам’ятовуючий елемент з двома стійкими станами, що має один інформаційний вихід та один вхід синхронізації.





$$Q_{t+1} = C_t \cdot D_t .$$

Рівняння показує, що після перемикання стан *D*-тригера повторює значення сигналу на *D*-вході в тактові моменти часу, тому їх називають тригерами затримки (від Delay – затримка). Якщо C=0, стан тригера стійкий та не залежить від рівня сигналу на інформаційному вході.

При цьому на входи *RS*-тригера з інверсними входами (елементи 3 та 4) надходять пасивні рівні (*S*=*R*=1). При *C*=1 інформація на прямому виході буде повторювати інформацію, що подається на вхід *D*.

Таблиця переходів *D*-тригера

D	Q^t	Q^{t+1}
0	0	0
0	1	0
1	0	1
1	1	1

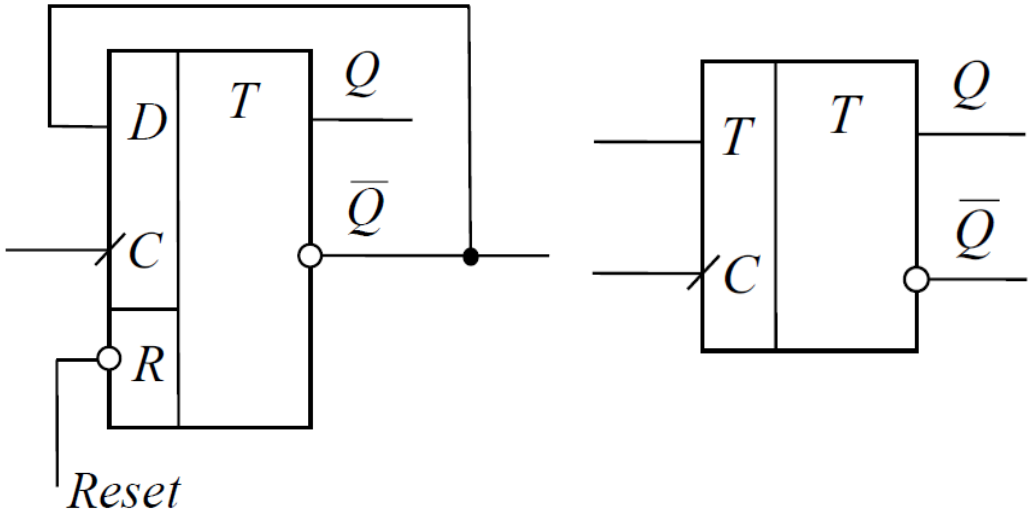
Таблиця функції входів D-тригера

Q^t	Q^{t+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

Для стійкої роботи тригера необхідно, щоб протягом синхроімпульсу інформація на вході була незмінною. **Таблиця функцій збудження пам'яті автомата, що синтезується, з використанням *D*-тригерів повністю співпадатиме з кодованою таблицею переходів цього автомата.**

Лічильний *T*-тригер

Тригером типу *T* називається запам'ятовуючий елемент з двома стійкими станами та одним інформаційним *T*-входом.



$$Q_{t+1} = \overline{T}_t \cdot Q + T_t \cdot \overline{Q}_t$$

Інформація на виході асинхронного *T*-тригера змінює свій знак на протилежний за кожного позитивного (або за кожного негативного) перепаду напруги на вході. Синхронний *T*-тригер змінює свій стан тільки тоді, коли $T=1$ та $C=1$.

Таблиця переходів *T*-тригера

T	Q^t	Q^{t+1}
0	0	0
0	1	1
1	0	1
1	1	0

Таблиця функції входів *T*-тригера

Q^t	Q^{t+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

У серіях мікросхем, що випускаються, *T*-тригерів, як правило, немає. Але тригер такого типу може бути побудований на базі тактового *D*-тригера, якщо його інверсний вихід з'єднати з інформаційним входом. Частота сигналу на виході *T*-тригера в два разу нижче частоти сигналу на вході, тому такий тригер можна використовувати як подільник частоти та двійковий лічильник.

JK-тригер

Тригером типу *JK* називається запам’ятовуючий елемент з двома стійкими станами та інформаційними входами *J* (аналог *S*) та *K* (аналог *R*), які забезпечують відповідно роздільну установку станів ‘1’ та ‘0’. Даний тригер функціонує подібно до *RS*-тригера, але при збігу сигналів *JK*=1 перемикається в протилежний стан, тобто **реалізує додавання сигналів за модулем два**. Таким чином, ***JK*-тригер не має заборонених комбінацій вхідних сигналів**. Тригер типу *JK* **є універсальним**, оскільки може виконувати функції *RS*-тригера (при роздільному надходженні сигналів *J* та *K*), *T*-тригера (при одночасній подачі сигналів *J* та *K*), *D*-тригера (при подачі сигналу від входу *J* через інвертор на вхід *K*).

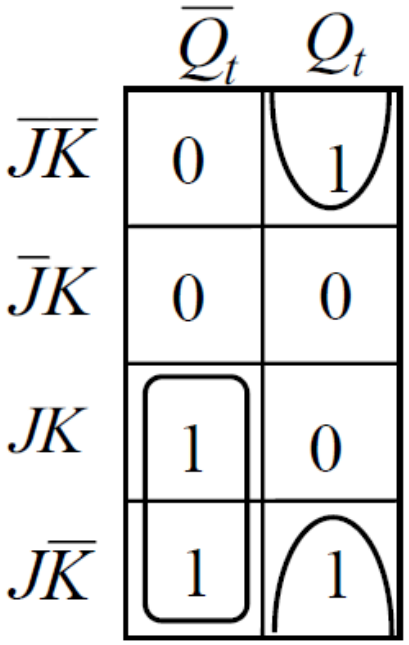
Таблиця переходів *JK*-тригера

C	K	J	Q^t	Q^{t+1}	Примітка
0	X	X	0	0	Режим зберігання інформації
0	X	X	1	1	
1	0	0	0	0	Режим зберігання інформації
1	0	0	1	1	
1	0	1	0	1	Режим встановлення одиниці $J=1$
1	0	1	1	1	
1	1	0	0	0	Режим записування нуля $K=1$
1	1	0	1	0	
1	1	1	0	1	$K=J=1$ лічильний режим тригера
1	1	1	1	0	

Як випливає з таблиці переходів, для комбінацій вхідних сигналів *JK*=00÷10 тригер поводиться як *RS*-тригер, а при комбінації *JK*=11 – як *T*-тригер.

Карта Карно

для *JK*-тригера



$$Q_{t+1} = \overline{K} \cdot Q + J \cdot \overline{Q}$$

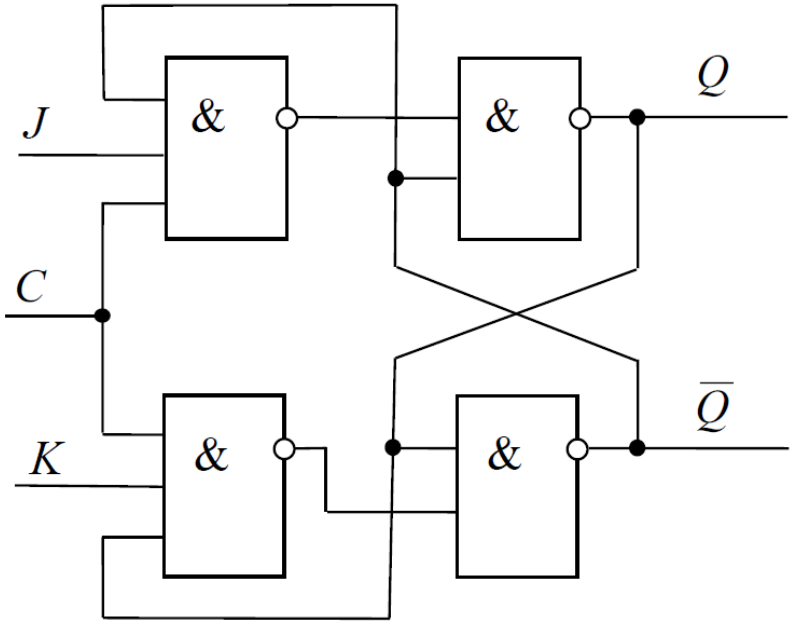
Таблиця функцій

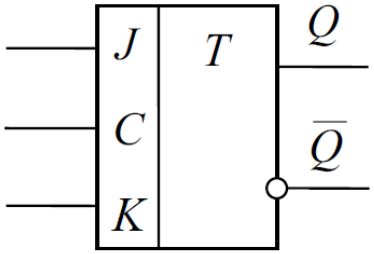
входів *JK* –тригера

Q^t	Q^{t+1}	J	K
0	0	X	0
0	1	1	X
1	0	X	1
1	1	0	X

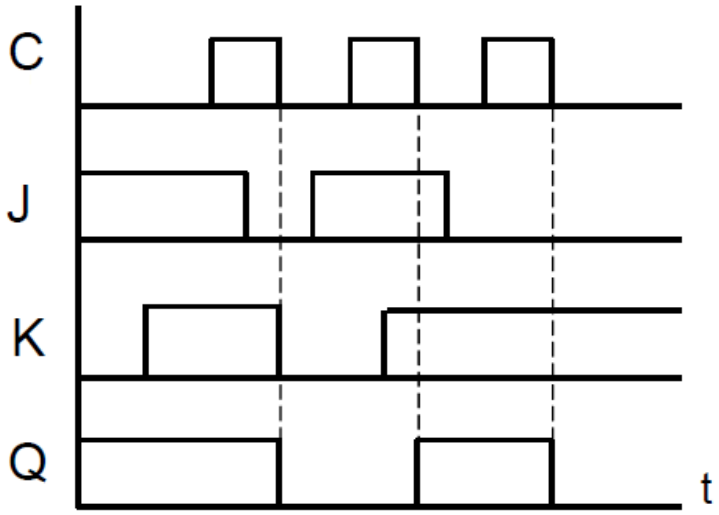
Для побудови одноступеневого синхронного *JK*-тригера на елементах І-НІ потрібно замінити в рівнянні роботи тригера змінні *J* та *K* на сполучення *CK* та *CJ*, після чого виконати перетворення на основі правил подвійного заперечення та закону де Моргана

$$Q_{t+1} = \overline{\overline{\overline{C \cdot K} \cdot Q} + C \cdot J \cdot \overline{Q}} = \overline{\overline{\overline{C \cdot K} \cdot Q} \cdot \overline{C \cdot J \cdot \overline{Q}}}$$





J	K	Q_t	Q_{t+1}	
0	0	Q	Q	зберігання інформації
0	1	*	0	установка «0»
1	0	*	1	установка «1»
1	1	Q	$\overline{Q}$	інверсія



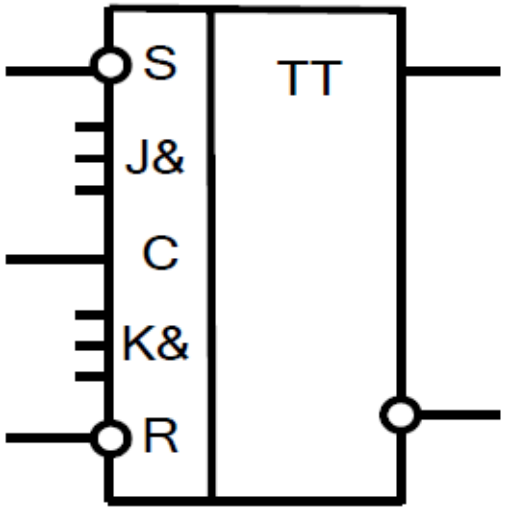
Із таблиці функцій входів *JK* –тригера можна отримати функцію збудження елементів пам'яті при синтезі автомата на *JK*-тригерах. Наприклад, під час переходу деякого автомата, що містить у своєму складі три *JK*-тригери, зі стану 010 у стан 110, функції збудження повинні бути:

- для першого тригера при переході з 0 в 1 – $J1 = 1$, $K1 = X$;
- для другого тригера при переході з 1 в 1 – $J2 = 0$, $K2 = X$;
- для третього тригера при переході з 0 в 0 – $J3 = X$, $K3 = 0$.

Таблиця входів для *JK* - *SR* - *і D* - тригерів.

Стан		*JK*-тригер		*SR*- тригер		*D*- тригер
Q_t	Q_{t+1}	*J*	*K*	*S*	*R*	*D*
0	0	0	*	0	*	0
0	1	1	*	1	0	1
1	0	*	1	0	1	0
1	1	*	0	*	0	1

У аналізованого тригера, як і в тригерів інших типів можуть бути входи асинхронної (несинхронизируемой) установки R і S , за допомогою яких при C= 0 тригер може бути встановлений в стан 1 шляхом подачі значень R=1 і S= 0, або в стан 0 шляхом подачі R=0 і S=1.



При подачі сигналів R=S=1, які не змінюють стан схеми, робота тригера здійснюється під впливом входів, що синхронізуються (C=1). І тут функціонування тригера описується його таблицею. На рис. зображені також вбудовані кон'юнктивні входи, що розширюють можливості схеми, при цьому $J = J_1J_2J_3$ і $K = K_1K_2K_3$.

Лекція 7

СТРУКТУРНИЙ СИНТЕЗ ЦИФРОВИХ АВТОМАТІВ

ТИПОВІ КОМБІНАЦІЙНІ СХЕМИ КОМП’ЮТЕРІВ

СТРУКТУРНИЙ СИНТЕЗ ЦИФРОВИХ АВТОМАТІВ

Після етапом абстрактного синтезу автомата, що закінчується мінімізацією числа внутрішніх станів, слідує етап структурного синтезу. Метою його є **побудова схеми автомата із логічних елементів заданого типу.** Якщо абстрактний автомат був лише математичною моделлю дискретного пристрою, то структурному автоматі необхідно враховувати структуру вхідних і вихідних сигналів, а також внутрішню будова автомата на рівні структурної чи функціональної (логічної) схеми. Основним завданням структурної теорії автоматів є **побудова композиції автоматів, тобто методів побудови складних автоматів із простіших автоматів.**

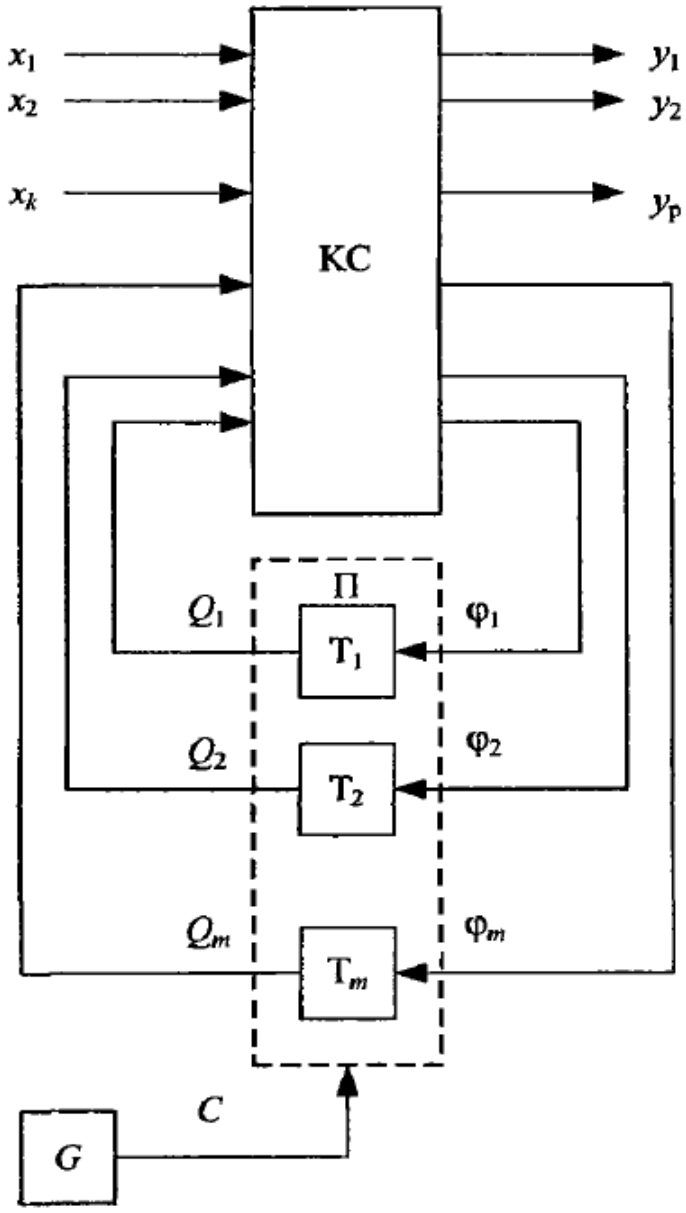
Елементарні автомати поділяють на **2** класи.

1-й –автомати з пам’яттю (**елементи пам’яті, запам’ятовуючі елементи**).

2-й - автомати без пам’яті (**комбінаційні**).

Тригери

Комбінаційна схема зі зворотними зв’язками, що має **два стійких стани** та **призначена для записування та зберігання одного біта інформації**, називається елементарним автоматом або **тригером**. Під дією вхідних сигналів тригер переходить з одного стійкого стану в інший.



ТИПОВІ КОМБІНАЦІЙНІ СХЕМИ КОМП'ЮТЕРІВ

Суматори

Суматором називається комбінаційна схема, яка виконує **арифметичне** (**на противагу логічному**) **додавання** та віднімання чисел.

Суматори використовуються у мікропроцесорах як для додавання двійкових чисел, так і для формування фізичної адреси комірки пам'яті.

Залежно від **системи числення** розрізняють: двійкові, двійково-десяткові, десяткові, інші (наприклад, амплітудні) суматори.

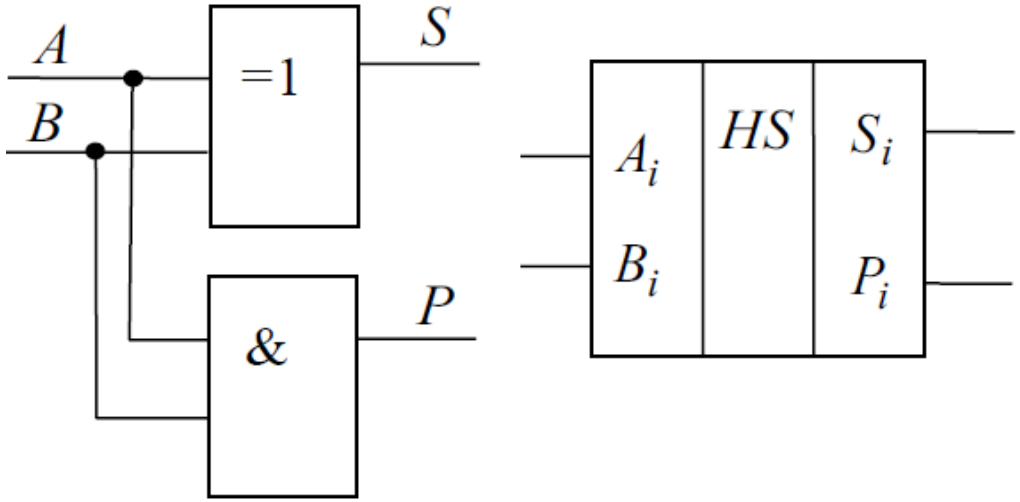
За **кількістю розрядів** доданків, що обробляються одночасно, суматори поділяються на: одноразрядні та багаторозрядні. На практиці використовуються **одноразрядні** суматори наступних типів: чвертьсуматори, напівсуматори та повні суматори.

Чвертьсуматори (елементи «**сума за модулем 2**») характеризуються наявністю двох входів, на які подаються два одноразрядні числа, та одним виходом, на якому реалізується їх арифметична сума.

Двійковий напівсуматор є найпростішою комбінаційною схемою та виконує додавання двох одноразрядних двійкових чисел. Він має два входи для доданків: A і B та два виходи: суми S та переносу P.

A	B	S	P	Примітка
0	0	0	0	0+0=0
0	1	1	0	0+1=1
1	0	1	0	1+0=1
1	1	0	1	1+1=0(P=1)

$$S = \overline{A}B + A\overline{B} = A \oplus B,$$
$$P = AB.$$



Повний двійковий суматор виконує додавання трьох одноразрядних двійкових чисел. Має три входи: доданків A , B та перенесення з попереднього молодшого розряду p та два виходи: суми S та переносу в наступний старший розряд P.

A	B	p	S	P
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

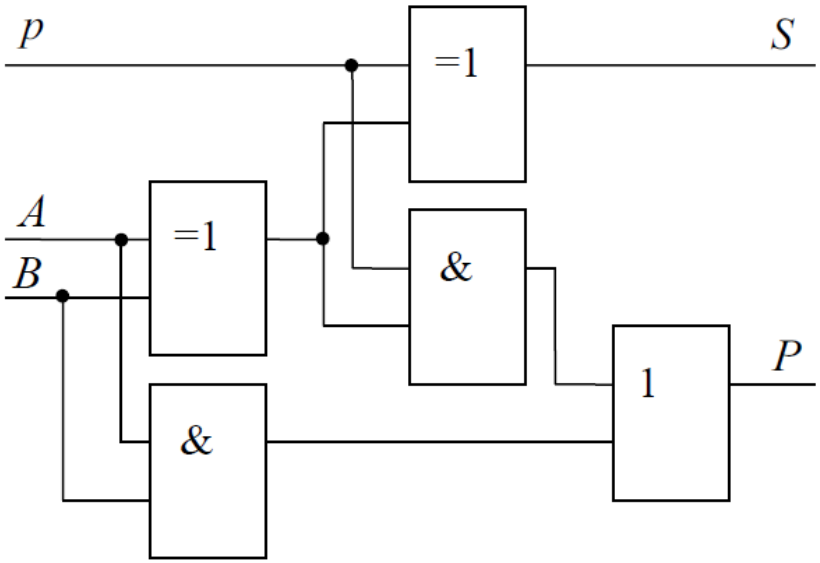
$$S = \overline{A}\,\overline{B}p + \overline{A}B\overline{p} + ABp + A\overline{B}\overline{p}$$

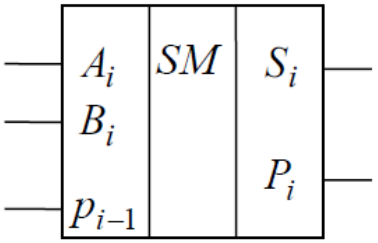
$$P = Ap + Bp + AB$$

	$\overline{p}$	p
$\overline{A}\,\overline{B}$	0	1
$\overline{A}B$	1	0
AB	0	1
$A\overline{B}$	1	0

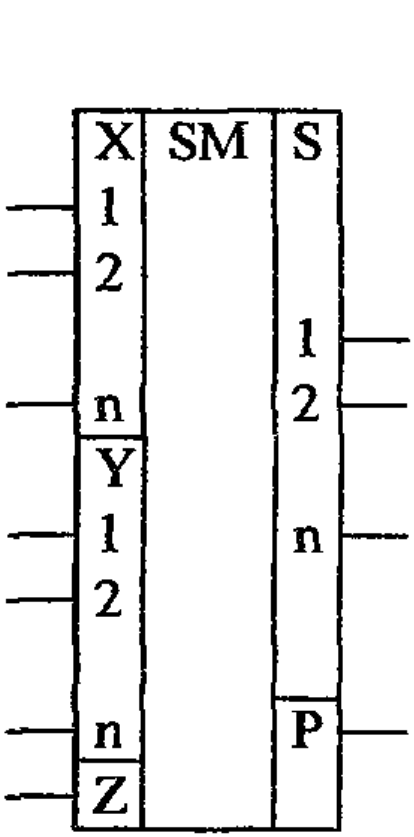
	$\overline{p}$	p
$\overline{A}\,\overline{B}$	0	0
$\overline{A}B$	0	1
AB	1	1
$A\overline{B}$	0	1

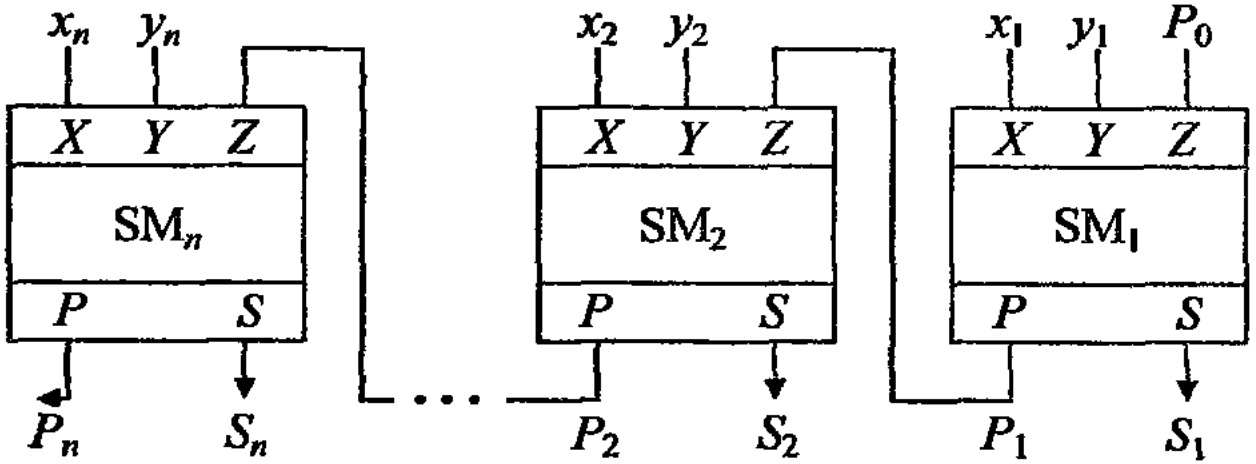
$$S = A \oplus B \oplus p, \quad P = p(A \oplus B) + AB$$

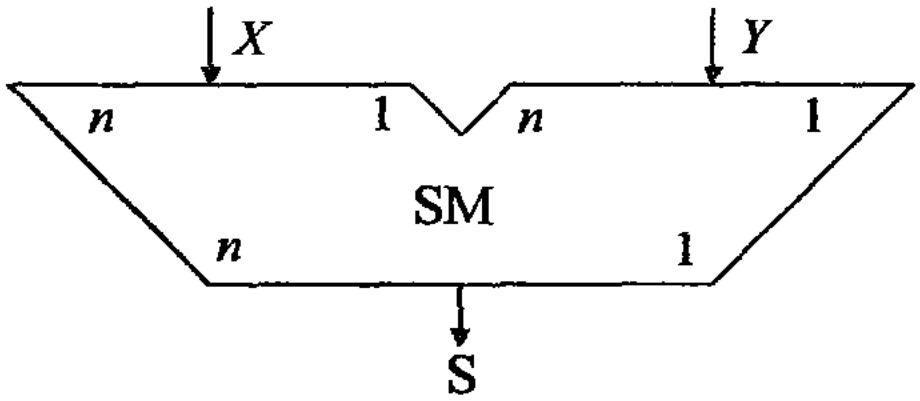




Багаторазрядний суматор.





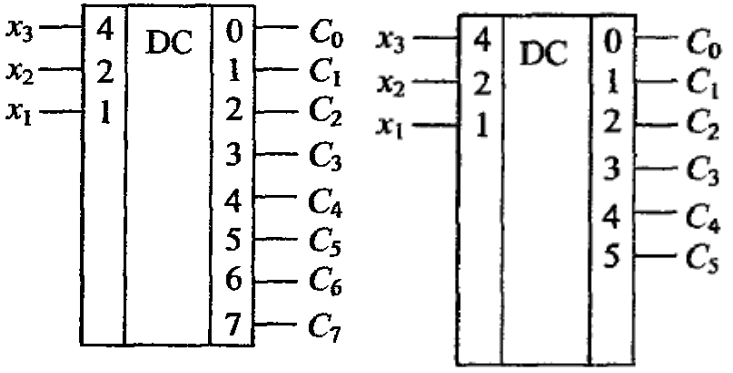


Дешифратори та шифратори

Дешифратор (декодер) – схема з n входами та 2^n виходами призначена для реалізації **конституент одиниці**. Розрізняють **повні** та **неповні** дешифратори. Повні дешифратори реалізують 2^n конституентнеповні дешифратори реалізують менше ніж 2^n конституент.

X_2	X_1	X_0	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

$$F_0 = \overline{X}_{n-1}\overline{X}_{n-2}\ldots\overline{X}_1\overline{X}_0;$$
$$F_1 = \overline{X}_{n-1}\overline{X}_{n-2}\ldots\overline{X}_1 X_0;$$
$$\ldots$$
$$F_{2^n-1} = X_{n-1}X_{n-2}\ldots X_1X_0,$$



Індекс функції F_i визначає номер обраного виходу та відповідає **десятковому еквіваленту вхідного коду**. Вихід, на якому з'являється керуючий сигнал, називають **активним**. Якщо значення сигналу на активному виході відображається логічною 1 (H), то на решті пасивних виходів встановлюється логічний 0 (L).

Двійковий код, який завжди містить тільки одну одиницюрешта – нулі, називається **унітарним**. Дешифратори є перетворювачами вхідного позиційного коду в унітарний вихідний. Крім позначень H та L можуть вживатись X (байдужий – 0 чи 1) та Z, що відповідає буферу виходу з Z станом.

8 виходів, на виходу з номером N формується 0, на інших – 1.

$$N = C \cdot 2^2 + B \cdot 2^1 + A \cdot 2^0$$

$$Y_i = \begin{cases} 0, & якщо \quad i = k; \\ 1, & якщо \quad i \neq k, \end{cases}$$

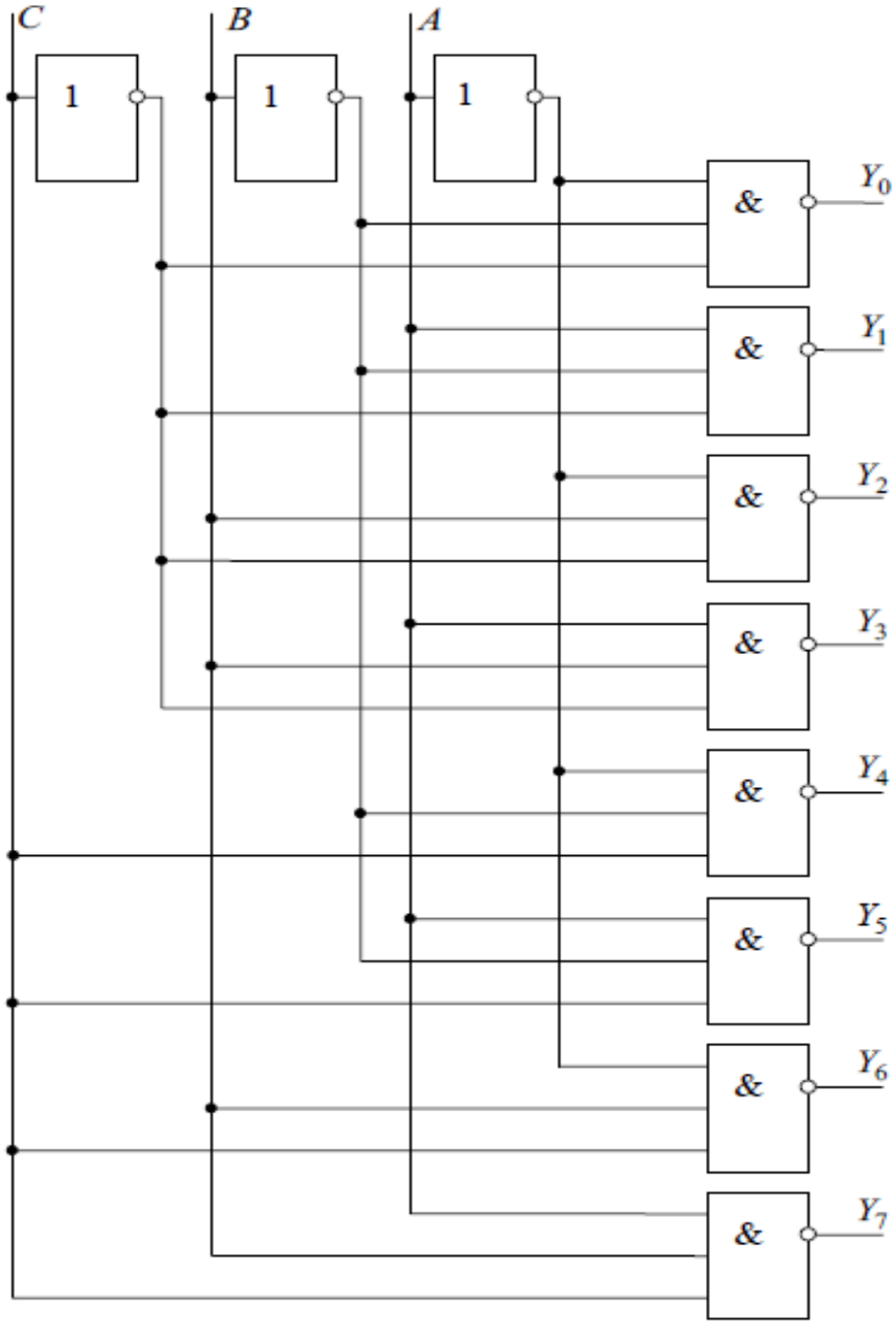
$$k = C \cdot 2^2 + B \cdot 2^1 + A \cdot 2^0$$

$$Y0 = \overline{\overline{C} \cdot \overline{B} \cdot \overline{A}}, \; Y1 = \overline{\overline{C} \cdot \overline{B} \cdot A},$$

$$Y2 = \overline{\overline{C} \cdot B \cdot \overline{A}}, \; Y3 = \overline{\overline{C} \cdot B \cdot A},$$

$$Y4 = \overline{C \cdot \overline{B} \cdot \overline{A}}, \; Y5 = \overline{C \cdot \overline{B} \cdot A},$$

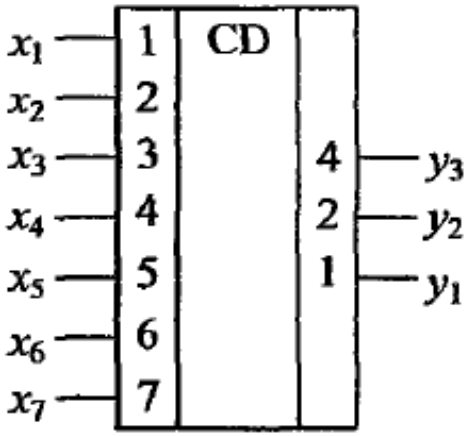
$$Y6 = \overline{C \cdot B \cdot \overline{A}}, \; Y7 = \overline{C \cdot B \cdot A}.$$



Шифратор – це типова комбінаційна схема, призначена для перетворення просторового **унітарного коду в двійковий код**.

Унітарний код, як уже зазначалось, має в своєму записі одну одиницю. З урахуванням цього можна вважати, що шифратор виконує функцію **зворотну функції дешифратора**, хоча в загальному випадку вихідний код може відрізнятися від коду з природним порядком ваг.

X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0	F_2	F_1	F_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1



Мультиплексори та демультиплексори

Мультиплексор – комбінаційна логічна схема, що є керованим перемикачем, який **під'єднує до виходу** один із інформаційних **входів** даних. Номер під'єднаного входу визначається комбінацією логічних значень на входах керування (**адресою**). Схема містить **вхід дозволу**, при активації якого мультиплексор переходить в пасивний стан – сигнал на виході зберігає постійне значення незалежно від значення інформаційних та керуючих сигналів. Число інформаційних входів дорівнює 2^n.

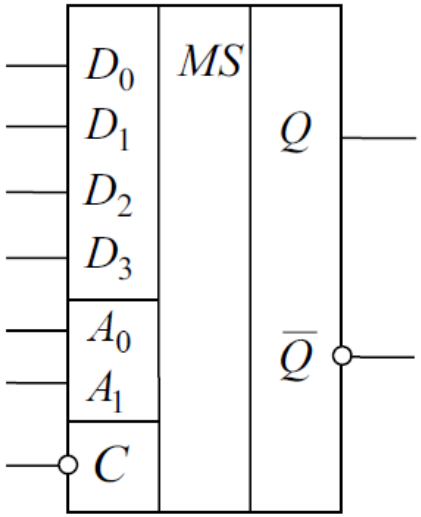
Приклад Мультиплексор з 4 інформаційними входами D_0, D_1, D_2, D_3, 2 адресними входами A_0, A_1, входом для подачі стробуючого сигналу C та 1 виходом Q. Кожному інформаційному входу мультиплексора приписується номер, що називається **адресою**.

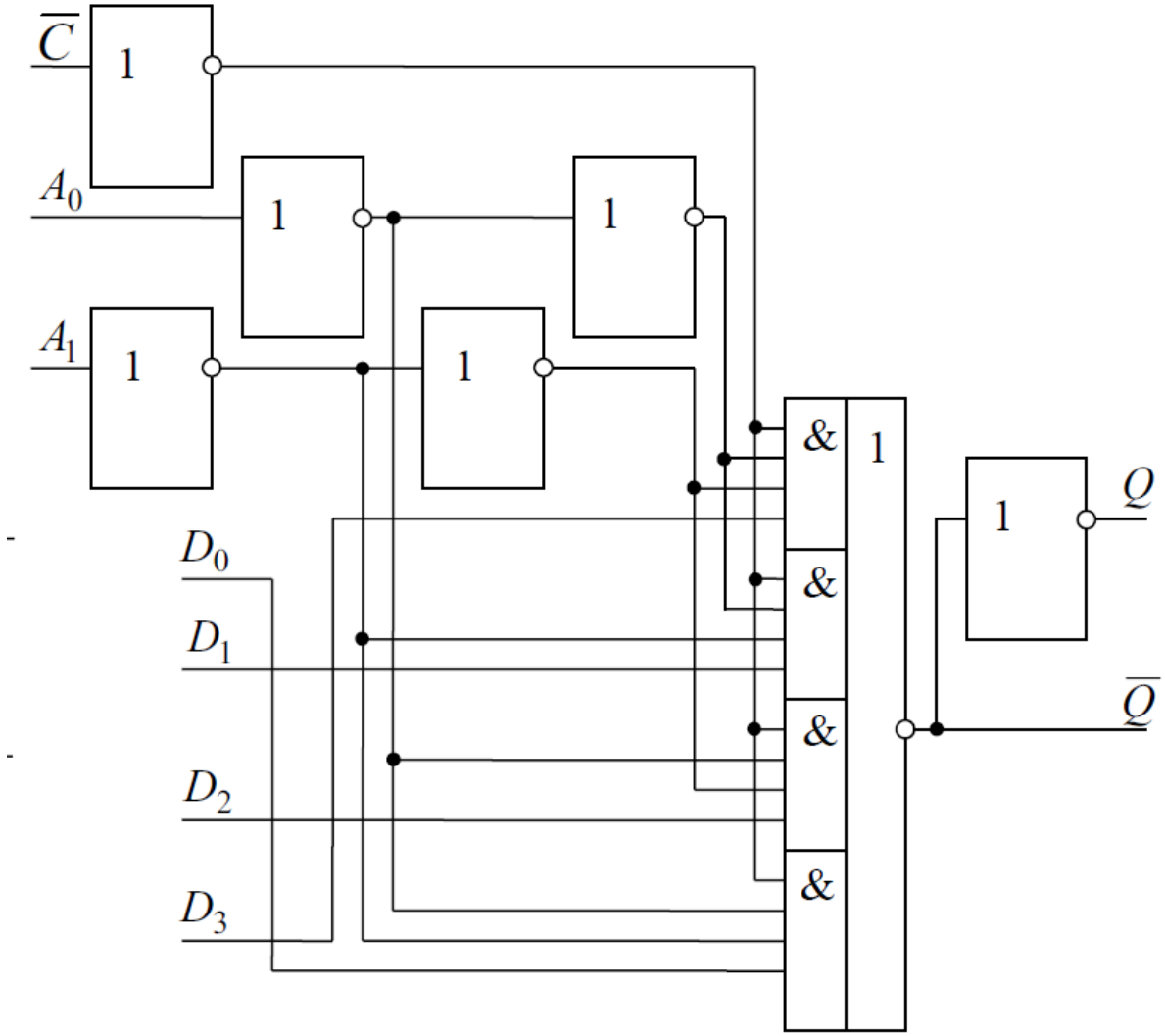
При C=1 мультиплексор вибирає один з входів, адреса якого задається двійковим кодом на адресних входах, та з'єднує його з виходом. Подаючи на адресні входи адреси різних інформаційних входів, можна передавати цифрові сигнали з цих входів на вихід Q. Кількість інформаційних входів $n_{інф}$ та кількість адресних входів $n_{адр}$ пов'язані співвідношенням $n_{інф}=2^{n_{адр}}$. При C=0 зв'язок між інформаційними входами та виходами відсутній (Q=0).

Застосовуються у мікропроцесорах для видачі на одні й ті самі виводи адреси та даних, що істотно скорочує загальну кількість виводів мікросхеми; у мікропроцесорних системах керування на віддалених об'єктах для передачі інформації по одній лінії від декількох сенсорів.

A_1	A_0	C	Q
*	*	0	0
0	0	1	D_0
0	1	1	D_1
1	0	1	D_2
1	1	1	D_3

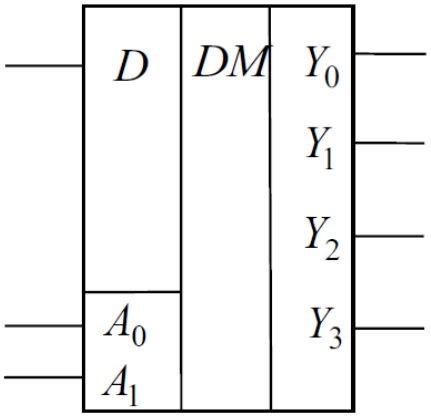
$$Q = \left(D_0 \overline{A}_1 \overline{A}_0 + D_1 \overline{A}_1 A_0 + D_2 A_1 \overline{A}_0 + D_3 A_1 A_0\right) \cdot C .$$

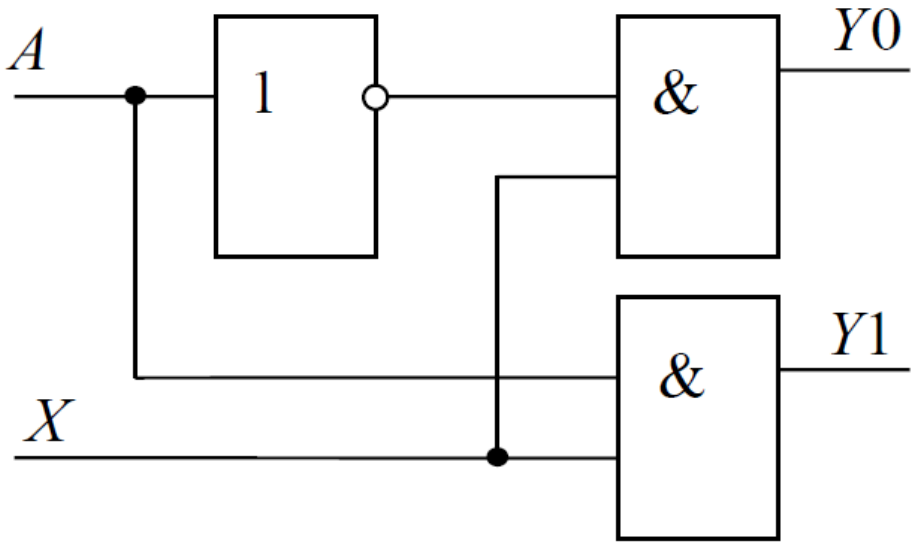




Демультиплексори функціонально протилежні мультиплексорам: сигнали з **одного інформаційного входу розподіляються у потрібній послідовності на декілька виходів**. Вибір потрібного виходу забезпечується встановленням відповідного коду на адресних входах.

A_1	A_0	Y_0	Y_1	Y_2	Y_3
0	0	D	0	0	0
0	1	0	D	0	0
1	0	0	0	D	0
1	1	0	0	0	D





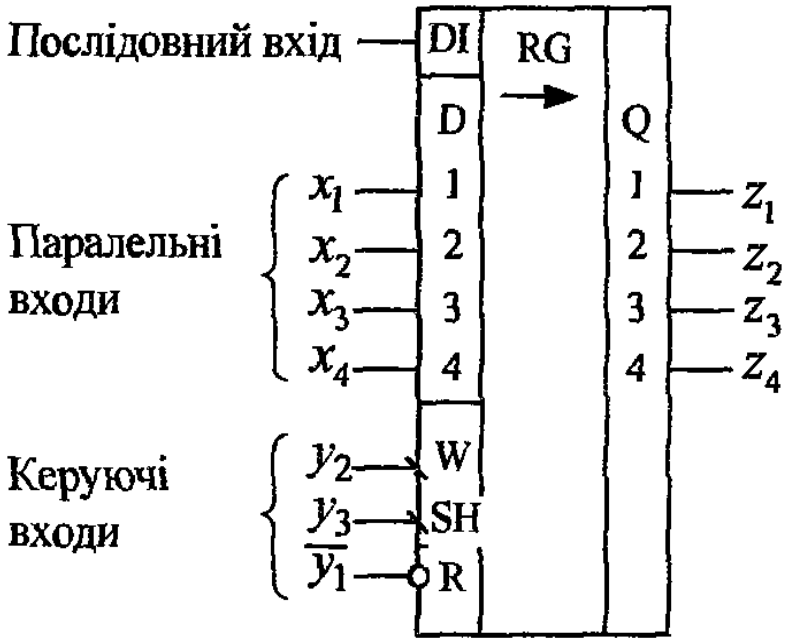
Якщо на вхід демультиплексора подати константу D=1, то на вибраному відповідно із заданою адресою виході буде логічна 1, на інших виходах – логічний 0.

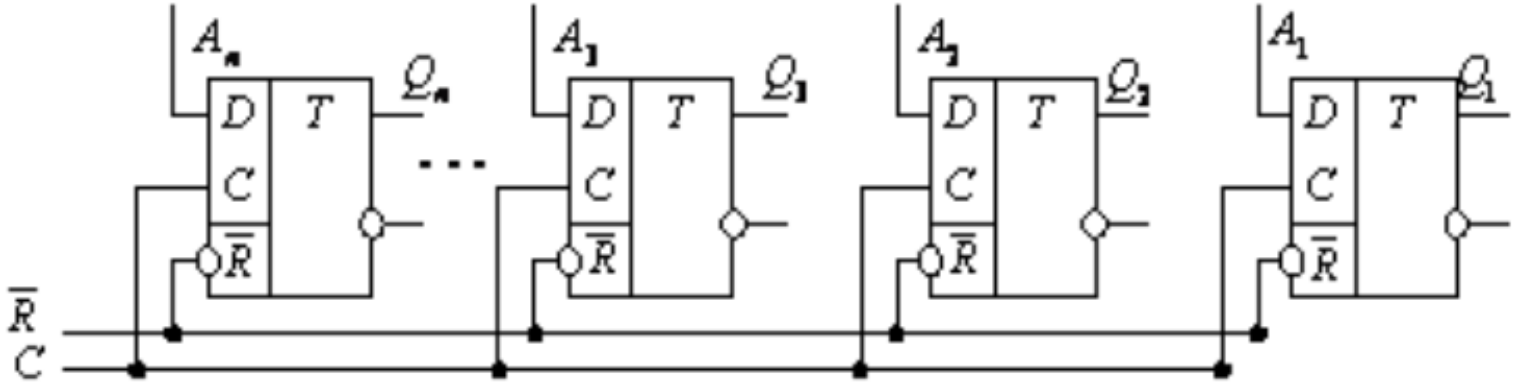
Використання демультиплексора може суттєво спростити побудову логічного пристрою, що має декілька виходів, на яких формуються різні логічні функції одних й тих самих змінних. За необхідності отримати більшу кількість виходів може бути побудоване демультиплексорне дерево.

Регістри

Регістром називається типовий функціональний вузол комп'ютера, призначений для приймання, зберігання, перетворення і видачі **n-розрядного двійкового слова**.

Це **набір однотипних тригерів**, в кожному з яких зберігається значення **одного двійкового розряду** слова.





Регістри, призначені **тільки для приймання (записування), зберігання і передачі інформації**, називаються **елементарними** або **фіксаторами**. Регістри, в яких зберігання даних **поєднується з мікрооперaціями зсуву**, називаються **зсувовими**.

Найчастіше використовують тригери типів *RS*, *JK* і *D* Елементарні регістри будують на одноступеневих тригерах, а зсувові – на двоступеневих або *D*-тригерах з динамічним керуванням.

Логічна функція регістра позначається буквами *RG* (register). Регістри забезпечують зберігання команд, адреси пам'яті, результатів операцій, індексів та ін.

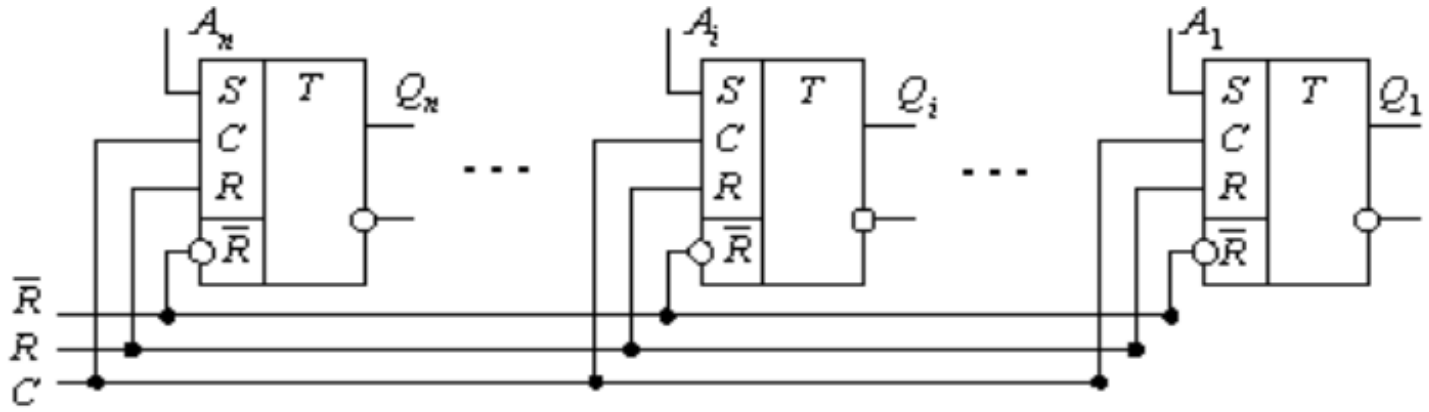
Регістри класифікують за такими ознаками:

- способом керування записуванням – **асинхронні** та **синхронні**;
- способом записування і видачі двійкових слів – **паралельні, послідовні** й **універсальні**; у паралельних записування і видача слів виконується одночасно всіма розрядами, а в послідовних – розряд за розрядом в напрямку від молодших розрядів до старших або навпаки; універсальні забезпечують обидва процеса обміну інформацією;
- числом ліній для представлення значення одного розряду слова (біта інформації) – **однофазні** й **парафазні**; при однофазному поданні значення кожного розряду слова передається по одній лінії зв'язку, а при парафазному – по двох лініях (одночасно відображається пряме та інверсне значення розряду);

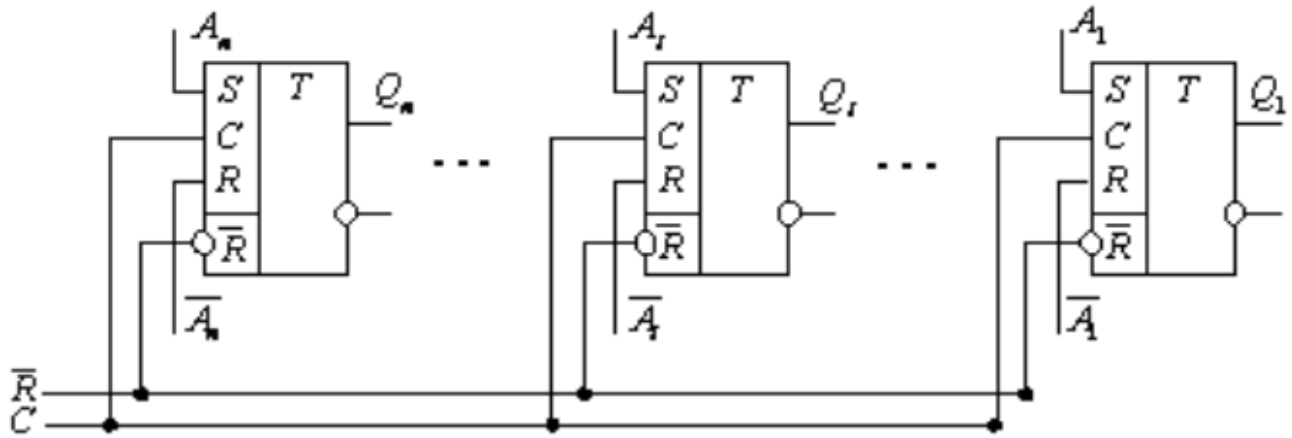
- числом тактів для записування слова – **одно-**, **дво-** і **багатотактні**;
- складом мікрооперацій, які виконуються: установлювальні, записування, читання, порозрядні логічні й зсуву, а також перетворення послідовного коду в паралельний і навпаки;
- напрямом зсуву – **односторонні** (лівий або правий зсув) і **двосторонні** (реверсивні);
- типом тригерів, що використовуються;
- елементною структурою – потенціальні, імпульсні й потенціально-імпульсні.

Установлювальні мікрооперації

Установлювальні мікрооперації служать для переключення регістрів у певний стан ("0" або "1"; установлення парних розрядів у стан "0", а непарних – у стан "1"; записування в регістр деякої константи та ін.) Установлювальні мікрооперації переважно використовують асинхронні входи регістра. У регістрах на *RS*- або *JK*- тригерах можливий **однофазний** або **парафазний** спосіб записування інформації. При однофазному записуванні значення кожного розряду слова $A=A_nA_{n-1}...A_i\ ...A_1$ надходить по одній лінії зв'язку на вхід *S* (або *J*) відповідних тригерів. Після зчитування записаної інформації регістр має обнулятися по спільному *R* входу. При однофазному записуванні частота обміну інформацією зменшується, оскільки процеси введення і скидання чергуються.



При парафазному записуванні значення A передається по 2 лініях зв'язку: пряме значення A_i надходить на вхід S (або J), а інверсне значення $\overline{A_i}$ – на вхід R (або K), тому не потрібне попереднє скидання регістра в стан "0".



У регістрах на *D*-тригерах, які мають один інформаційний вхід, можливий тільки однофазний спосіб записування інформації.

При необхідності збереження інформації на декілька тактів у регістрах на *D*-тригерах потрібен дозволяючий *V*-вхід, або блокування *C*-входу.

Мікрооперації зсуву

Зсув – це одночасне просторове переміщення двійкового слова в розрядній сітці із збереженням порядку слідування нулів і одиниць. Регістри, призначені для виконання мікрооперацій зсуву, називаються регістрами зсуву або зсувовими. Мікрооперації зсуву використовують у процесі виконання команд множення, ділення і нормалізації.

$$Q_i^{t+1} = Q_{i-j}^t \text{ — за зсуву вліво;}$$

$$Q_i^{t+1} = Q_{i+j}^t \text{ — за зсуву вправо.}$$

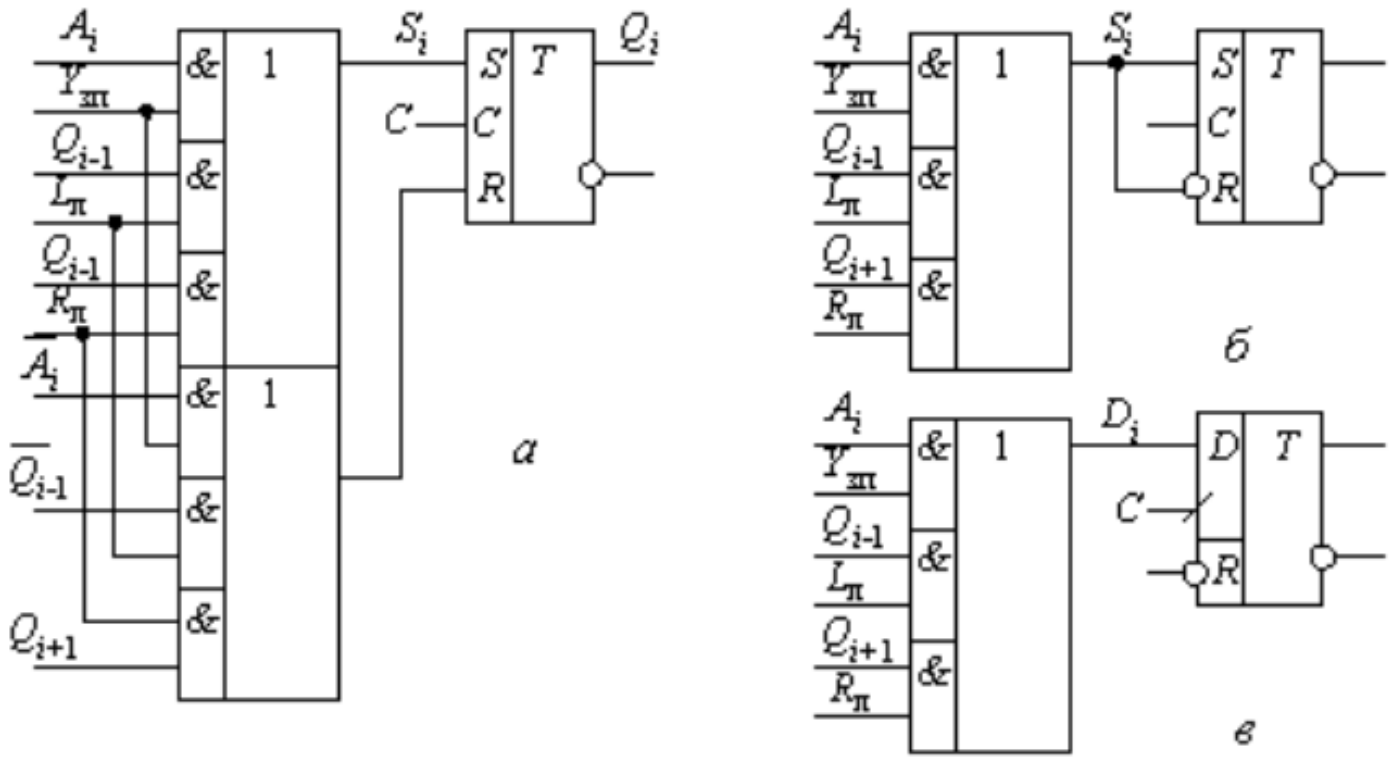


Схема розряду регістра зсуву: а – на RS-тригері; б – з інвертором на вході R; в – на D-тригері

Зсувні регістри проектують на **двоступеневих** *RS*- (або *JK*-) або *D*-тригерах з динамічним керуванням по фронту для **розділення процесів** приймання нової інформації в кожному розряді та видачі (зсуву) старої.

Кодоперетворювач – комбінаційна схема, яка перетворює n -елементний код в m -елементний код та використовується для шифрації та дешифрації цифрової інформації. ($m=n$, $m>n$,$m<n$).

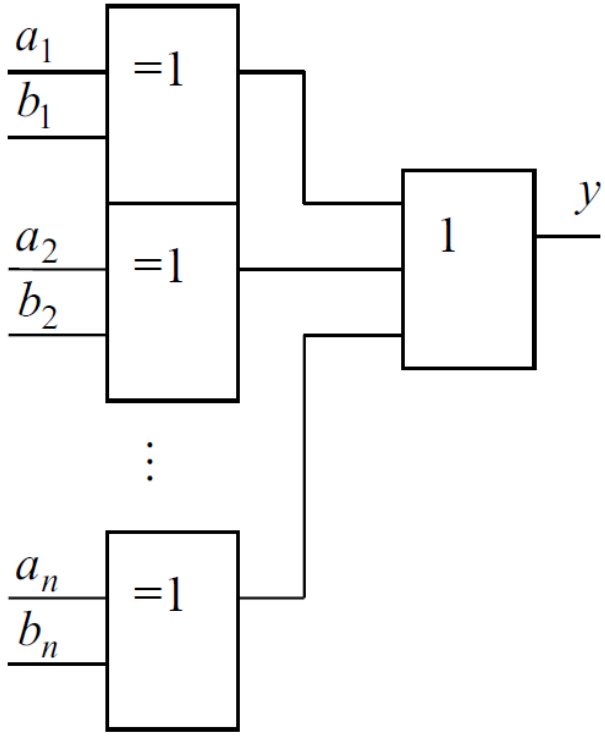
Кодоперетворювачі можна побудувати двома методами:

1) кодоперетворювач реалізується як система булевих функцій групи аргументів. Схему перетворювача кодів в такому випадку будують шляхом синтезу схеми з декількома виходами.

2) кодоперетворювач подається у вигляді пари дешифратор-шифратор.

Використовують **скорочену форму закону функціонування**: двійкові коди заміняють їх десятковими еквівалентами. Кількість входів дешифратора дорівнює кількості входів перетворювача кодів, кількість виходів шифратора дорівнює кількості виходів перетворювача кодів.

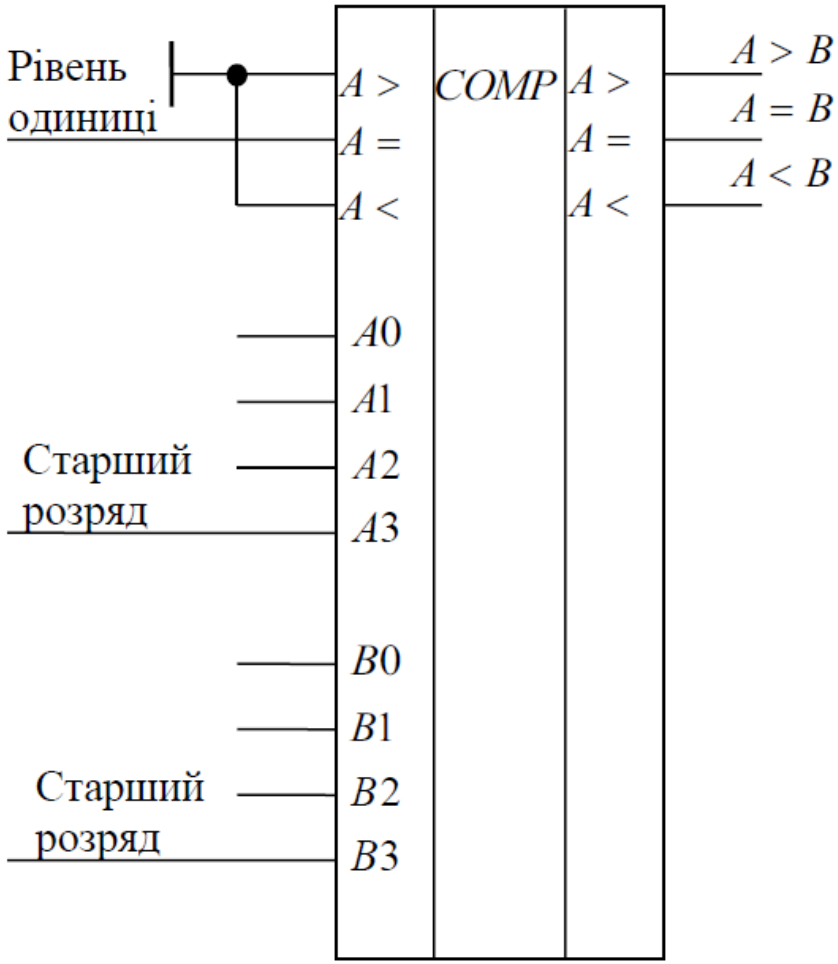
Пристрій порівняння (цифровий компаратор) – комбінаційна схема, що призначена для порівняння **двох багаторозрядних двійкових чисел** та формування результату порівняння у вигляді цифрових сигналів. Порівняння є **на рівність** та **на нерівність**. Часто необхідно встановити рівність постійного числа *A* з числом *B*, яке в кожен такт збільшується або зменшується на 1. Для визначення, коли *B*=*A*, проводиться **порозрядне додавання за модулем 2**. При *n*-розрядних числах пристрій складається з *n* суматорів за модулем 2, виходи яких з'єднані з елементом АБО. при співпадінні значень всіх розрядів чисел *A* та *B* на виходах всіх суматорів буде 0.



Якщо числа відрізняються в одному певному розряді, на виході відповідного суматора та на загальному виході буде 1 (при застосуванні елемента АБО рівності відповідає 1).

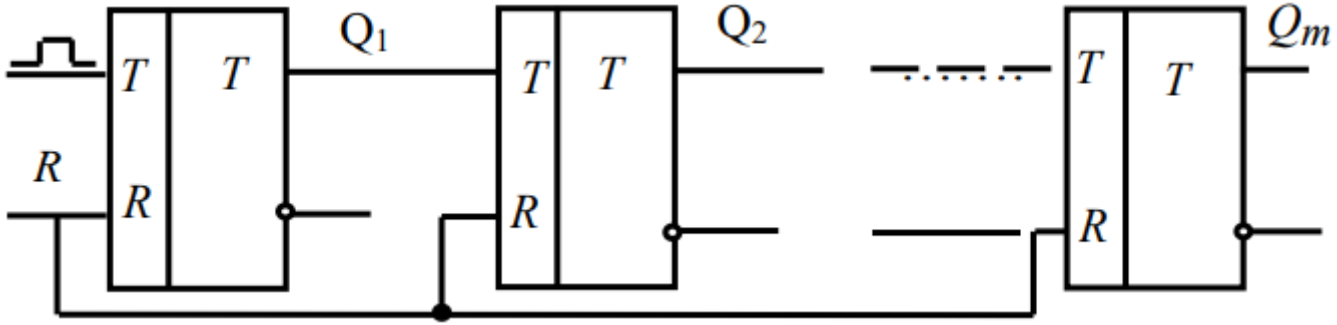
Порівняння на нерівність реалізується у вигляді мікросхеми та має окрім інформаційних входів **3 виходи для стикування з попередньою мікросхемою та 3 виходи, які є виходами всієї багаторозрядної схеми порівняння**.

Умовне позначення компаратора чотирирозрядних двійкових чисел



Двійковий лічильник

Двійковий лічильник – послідовнісна сжема (регистр), призначений **для підрахунку вхідних сигналів**. Лічильник є зв'язаним ланцюгом Т-тригерів, які створюють пам'ять із заданим числом сталих станів. Вхідні імпульси можуть поступати на лічильник як періодично, так і довільно розподіленими в часі. Розрядність лічильника п рівна числу *T*-тригерів.



Кожен вхідний імпульс **змінює стан лічильника**, **який зберігається до надходження наступного сигналу**. Значення виходів Q_n, $Q_{n-1},\ldots,Q_1$ відображають результат рахунку в прийнятій системі числення.

Літерами **СТ (counter)** позначається логічна функція лічильника. Список мікрооперацій лічильника включає попередню **установку в початковий стан**, **інкремент** або **декремент** слова, що зберігається, **видачу слів паралельним кодом** тощо. Є одним з основних функціональних вузлів комп'ютера, різних керуючих і інформаційно-вимірювальних систем.

Основне застосування лічильників:

- утворення послідовності адрес команд програми (лічильник команд або програмний лічильник);
- підрахунок числа циклів при виконанні операцій ділення, множення, зрушення (лічильник циклів);
- отримання сигналів мікрооперацій і синхронізації; аналогоцифрові перетворення і побудова електронних таймерів (годинника реального часу).

Лічильник характеризується **модулем** і **ємкістю рахунку**. **Модуль рахування** $K_{лч}$ – число станів лічильника. Для n-розрядного лічильника $K_{лч} = 2^n$. Після підрахування $K_{лч}$ імпульсів лічильник повертається в початковий стан. **Ємкість рахування** N_{max} визначає максимальну кількість імпульсів, яку може зафіксувати лічильник при одному циклі роботи. $N_{max} = K_{лч}–1$, якщо робота починається з нульового початкового стану.

Використовуються 3 режими роботи: **управління**, **накопичення** і **ділення**. Зчитування інформації в режимі управління проводиться **після кожного вхідного рахункового імпульсу**, наприклад, в лічильнику адреси команд. У режимі накопичення головним є **підрахунок заданого числа імпульсів** або рахування протягом певного часу. У режимі ділення (перерахування) основним є **зменшення частоти надходження імпульсів** в $K_{лч}$ разів.

ТАБЛИЦЯ СТАНІВ ЛІЧИЛЬНИКА З ПРИРОДНИМ ПОРЯДКОМ РАХУНКУ

Кількість рахункових сигналів	Стан лічильника	
	підсумо-вуючого	віднімаючого
0	0000	0000
1	0001	1111
2	0010	1110
3	0011	1101
4	0100	1100
5	0101	1011
6	0110	1010
7	0111	1001
8	1000	1000
9	1001	0111
10	1010	0101
11	1011	0101
12	1100	0100
13	1101	0011
14	1110	0010
15	1111	0001
16	0000	0000
...	...	...

ТАБЛИЦЯ СТАНІВ ЛІЧИЛЬНИКА ЗІ ШТУЧНИМ ПОРЯДКОМ РАХУНКУ

Кількість рахункових сигналів	Стан лічильника
0	0000
1	0001
2	0011
3	0010
4	0110
5	0111
6	0101
7	0100
8	1100
9	1101
10	1111
11	1110
12	1010
13	1011
14	1001
15	1000
16	0000
...	...

Лічильники **класифікують** за такими ознаками:

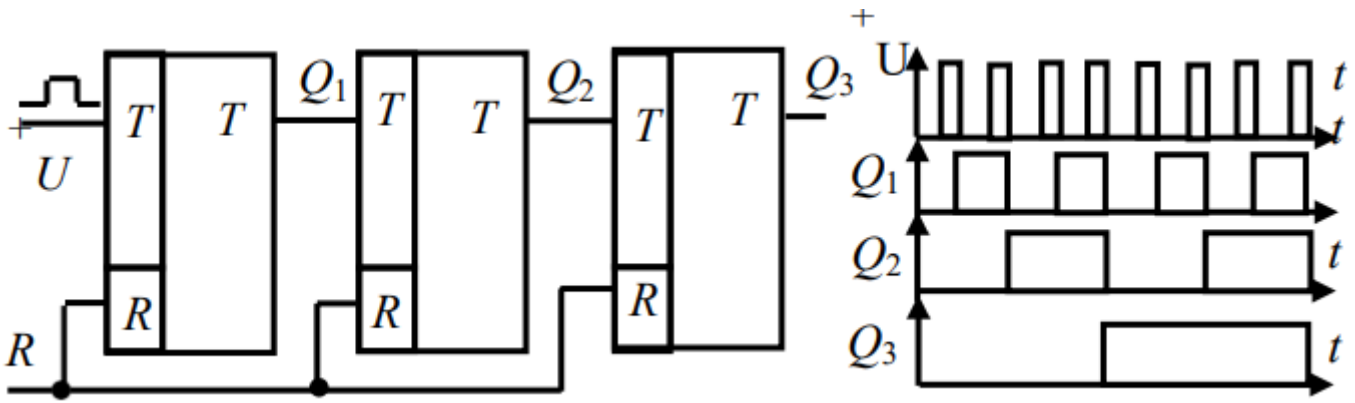
- способом кодування – **позиційні** і **непозиційні**;
- модулем рахування – **двійкові**, **десяткові**, з довільним постійним або **змінним (програмованим) модулем**;
- напрямком рахування – **прості** (що підсумовують, віднімають) і **реверсивні**;
- способом організації міжрозрядних зв'язків – з **послідовним, наскрізним, паралельним і комбінованим перенесеннями** (позикою);
- типом використовуваних тригерів – *T, JK,D* в **рахунковому режимі**;
- елементним базисом – **потенційні, імпульсні і потенційно-імпульсні**.

Двійкові лічильники реалізують підрахунок вхідних імпульсів в двійковій системі числення. Число розрядів n двійкового лічильника, що підсумовує, для заданого модуля M знаходять з виразу $n = \log_2 M$. Значення поточного числа N^+ вхідних імпульсів n- розрядного підсумовувального лічильника при відліку з нульового початкового стану визначають по формулі

$$N^+ = \sum_{i=1}^{n} 2^{i-1} Q_i = 2^{n-1} Q_n + 2^{n-1} Q_{n-1} + \ldots + 2^0 Q .$$

У двійковому лічильнику, що підсумовує, перенесення P_i у старший сусідній розряд Q_{i+1} виникає в тому випадку, якщо у момент надходження чергового рахункового імпульсу U^+ всі молодші розряди знаходяться в одиничному стані, тобто Pi ,$=U^+ Q_i Q_{i-1} \ldots Q_1 = 1$. Після перенесення старший розряд перемикається в стан "1", а всі молодші розряди – в стан "0".

Асинхронні підсумовувальні лічильники на двоступеневих *T*-тригерах будуються так, щоб вхідні імпульси U^+ поступали на рахунковий вхід тільки першого (молодшого) розряду. Сигнали перенесення передаються асинхронно (послідовно в часі) з прямих виходів молодших розрядів на *T* входи сусідніх старших



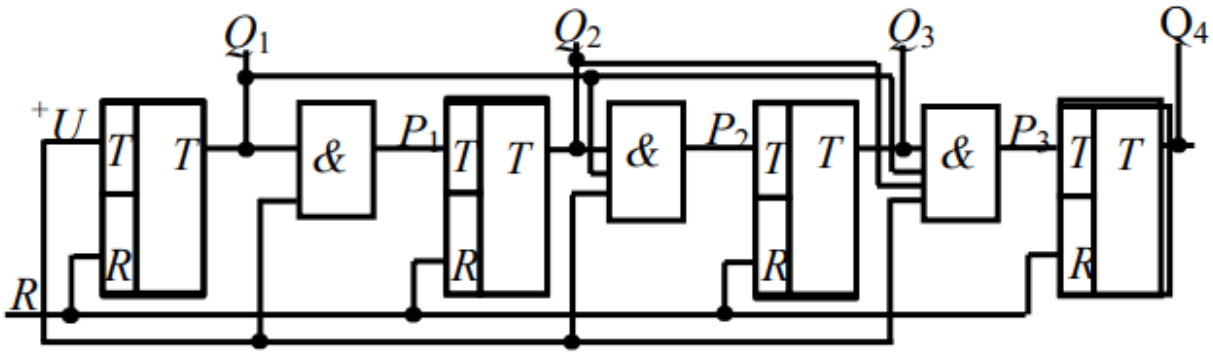
У режимі управління швидкодія асинхронного лічильника приблизно в *n* разів менша, ніж в режимі ділення.

Перевагою асинхронних лічильників є **простота схеми**: збільшення розрядності проводиться підключенням необхідного числа тригерів.

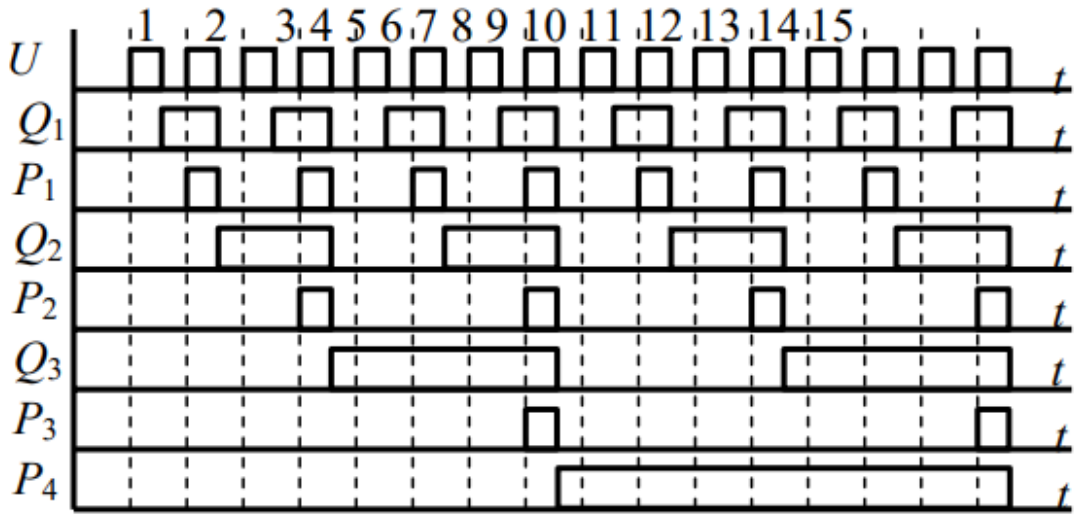
До **недоліків** асинхронних лічильників відносять порівняно **низька швидкодія в режимі управління** та її залежність від числа розрядів, а також **поява проміжних вихідних двійкових кодів** у процесі послідовного перемикання тригерів у новий стан.

Для отримання мінімального часу перемикання лічильника використовують паралельні перенесення: в кожному розряді синхронного лічильника є схема збігу, за допомогою якої аналізуються стани всіх попередніх молодших тригерів і виробляються функції перенесення згідно наступним логічним співвідношенням

$$P_1 = UQ_1; P_2 = UQ_2Q_1; P_3 = UQ_3Q_2Q_1; P_4 = Q_4Q_3Q_2Q_1$$



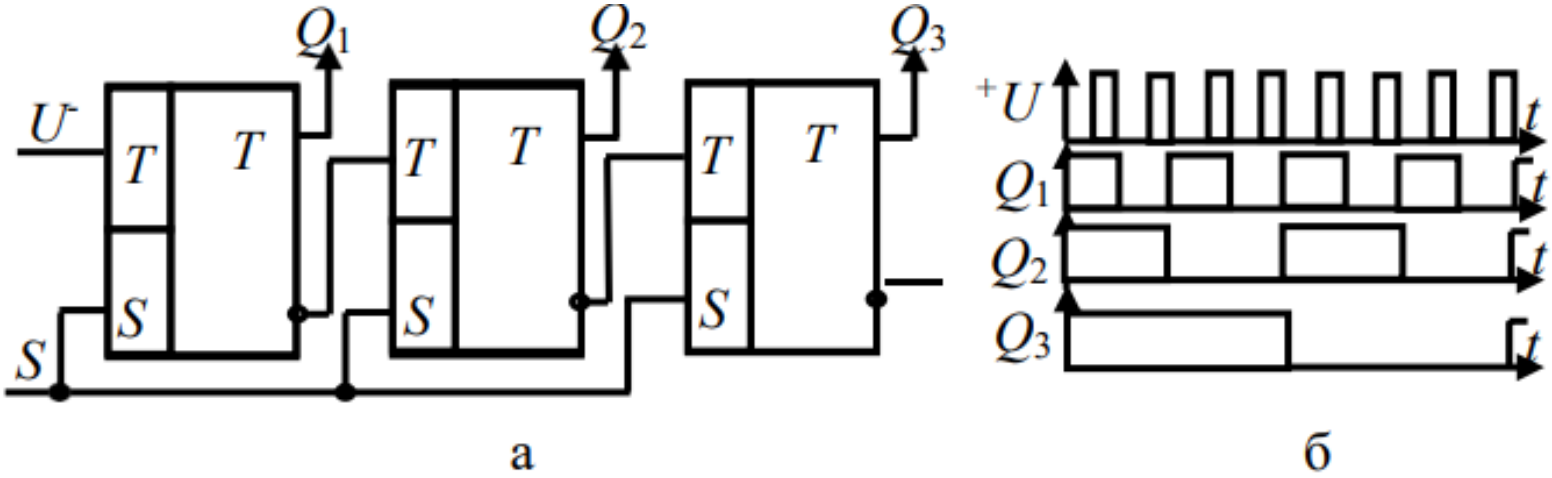
а



б

Підсумовувальний лічильник з паралельними перенесеннями: а – схема, б – часова діаграма роботи

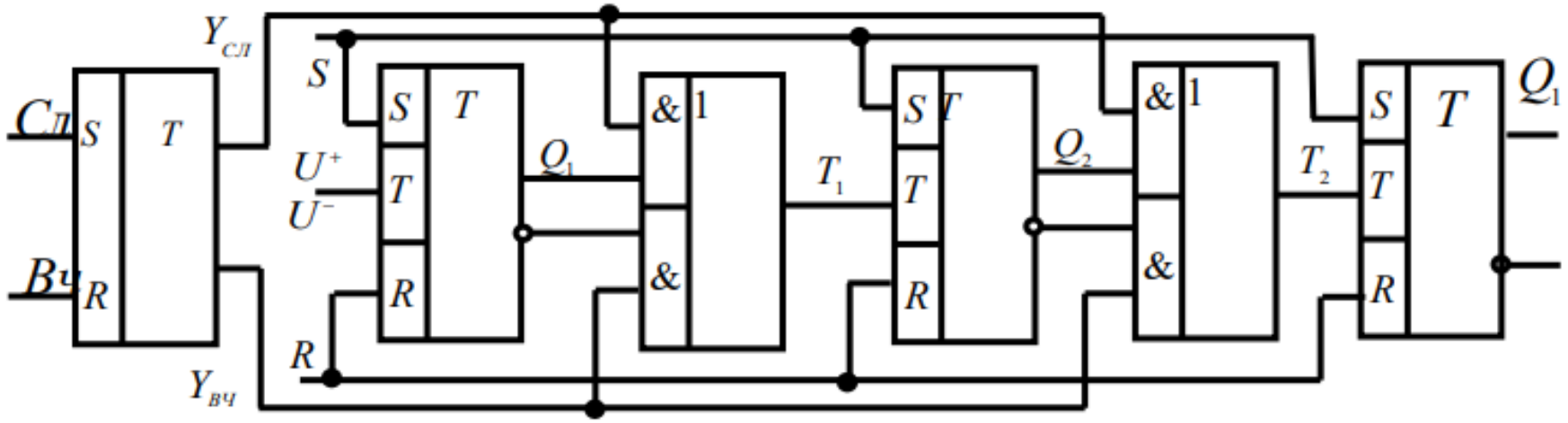
У віднімальних лічильниках сигнали віднімального зв’язку називаються позиками. За правилом двійкового віднімання у момент надходження рахункового імпульсу U^- позичка зі старшого розряду з одиничним значенням виникає за умови, що всі молодші тригери знаходяться в нульовому стані. Після цього всі вони перемикаються в стан “1”, а старші – в стан «0».



а б

Асинхронний віднімаючий лічильник на двоступеневих тригерах: а – схема; б – часова діаграма роботи

Двійкові реверсивні лічильники мають переходи в двох напрямках: у прямому (за рахунок підсумовувальних сигналів U^+) і в оберненому (за рахунок сигналів U^-, що віднімаються).



Для завдання напрямку рахування використовують віднімальний *RS* тригер: з його прямого виходу знімається сигнал управління додаванням $Y_{ск}$ (включає ланцюги перенесення), а з інверсного виходу – сигнал управління відніманням $Y_{від}$ (включає ланцюги позики).

Чим більше розрядів має лічильник, тим більший час йому потрібен на повне перемикання всіх розрядів. Затримка перемикання кожного розряду приблизно рівна затримці трігера, а повна затримка встановлення коду на виході лічильника рівна затримці одного розряду, помноженій на число розрядів лічильника. При періоді вхідного сигналу, меншому ніж повна затримка встановлення коду лічильника, правильний код на виході лічильника просто не встигне встановитися, тому така ситуація не має сенсу. Це створює **обмеження на період (частоту) вхідного сигналу**, причому збільшення, наприклад, удвічі кількості розрядів лічильника автоматично зменшує у двічі гранично допустиму частоту вхідного сигналу.

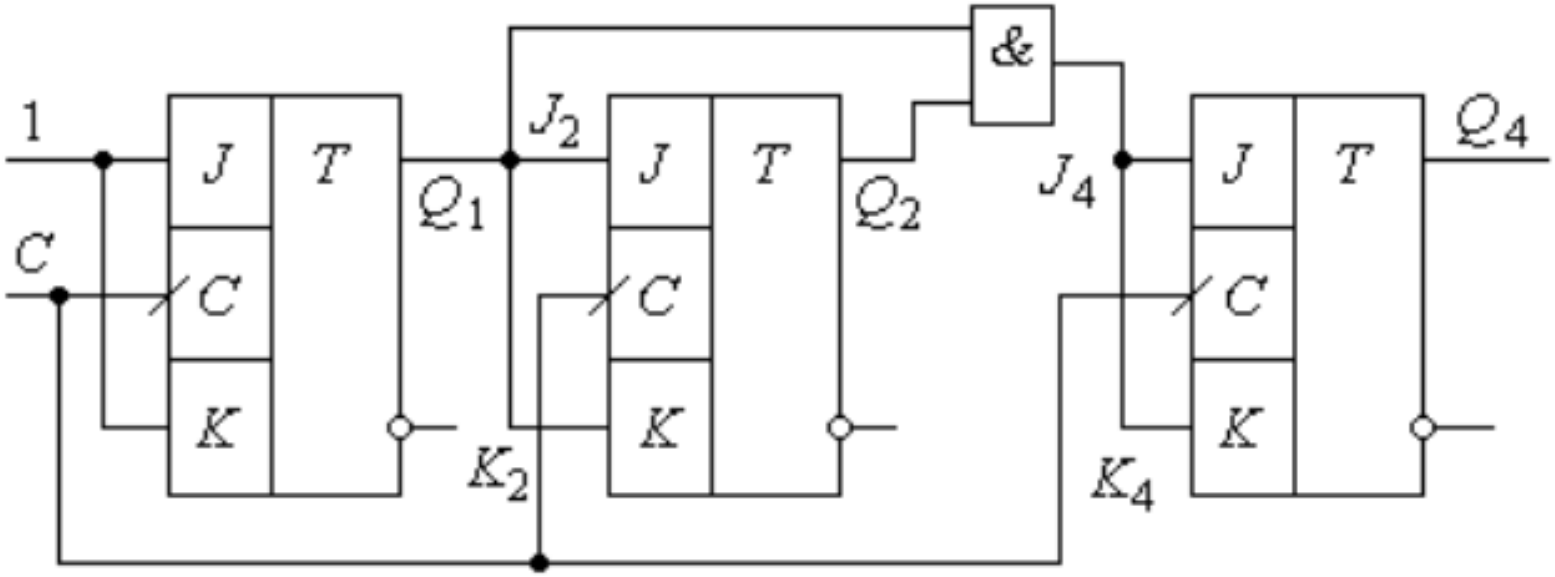
Треба ще врахувати, що за періодом вхідного сигналу повинен встигнути спрацювати пристрій (вузол), на який поступає вихідний код лічильника,

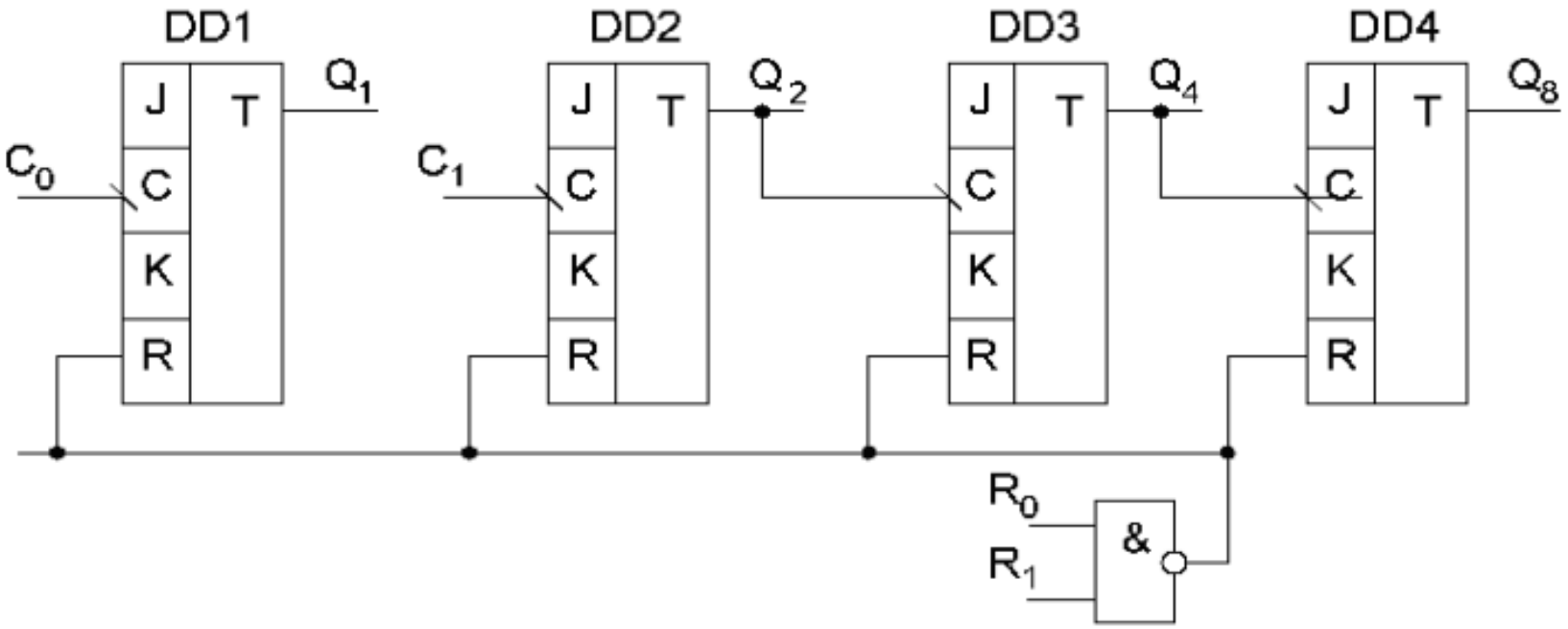
інакше лічильник просто не потрібен, тому обмеження на частоту вхідного сигналу звичайно буває ще жорсткіше.

Синхронні (або паралельні) лічильники характеризуються тим, що всі їх розряди в межах однієї мікросхеми перемикаються **одночасно, паралельно**. Це досягається **істотним ускладненням внутрішньої структури** мікросхеми в порівнянні з простими асинхронними лічильниками. В результаті повна затримка перемикання синхронного лічильника приблизно рівна затримці одного трігера, тобто синхронні лічильники набагато швидше за асинхронні, причому їх швидкодія не падає із зростанням кількості розрядів вихідного коду (звичайно, до певних меж).

Управління роботою синхронного лічильника набагато складніше, ніж у асинхронного лічильника, а кількість розрядів синхронних лічильників

звичайно не перевищує чотирьох. Тому синхронні лічильники не завжди можуть успішно конкурувати з асинхронними лічильниками, особливо при невисоких вимогах до швидкодії.




DD1
DD2
DD3
DD4
J
T
C
K
R
C_0
C_1
Q_1
Q_2
Q_4
Q_8
R_0
R_1
&

Лекція 8

- ЕТАПИ СИНТЕЗУ АВТОМАТІВ
- СТРУКТУРНИЙ СИНТЕЗ ЦИФРОВИХ АВТОМАТІВ
- КАНОНІЧНИЙ МЕТОД СТРУКТУРНОГО СИНТЕЗУ

ЕТАПИ СИНТЕЗУ АВТОМАТІВ

I (попередній) **етап блочного синтезу** автомата.

II етап (**абстрактного синтезу) -**визначають функції переходів та виходів.

На III етапі мінімізу**ють кількість** внутрішніх станів автомата.

На IV етапі синтезу проводиться кодування внутрішніх станів автомата **(розміщення внутрішніх станів),** вхідні і вихідні сигнали.

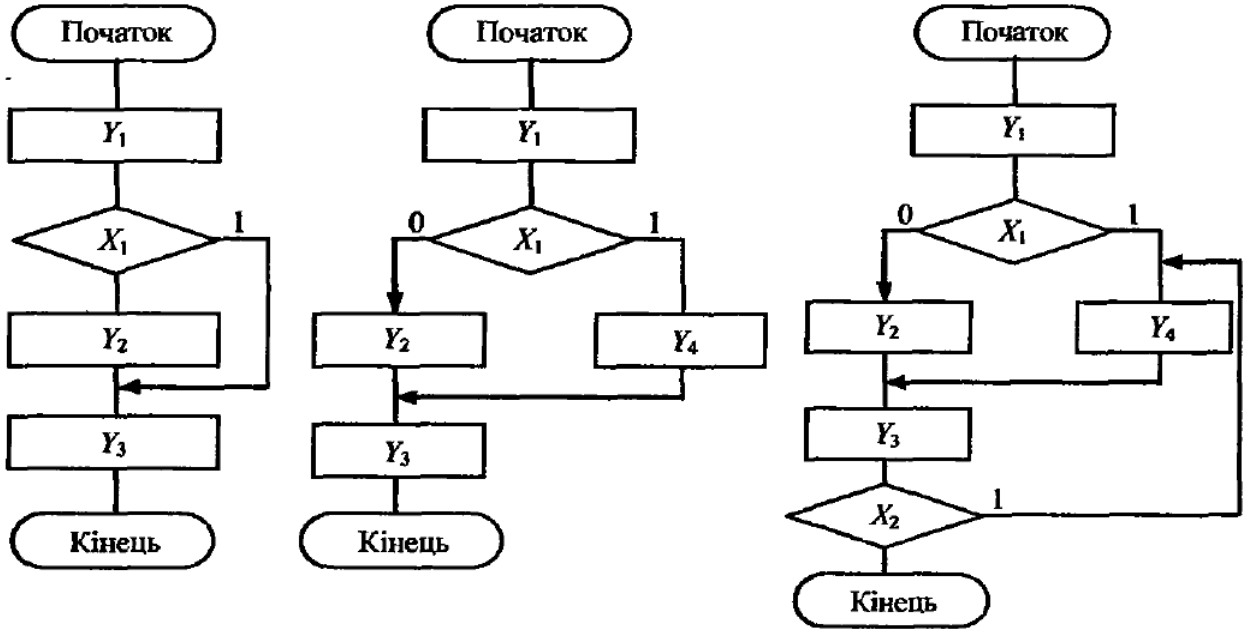
На V етапі (**структурного** синтезу) будується **функціональна схема з комбінаційної частини й автоматів пам'яті**.

VI етап проведення електричного та інших розрахунків елементів схем.

На **VII** етапі складаються монтажні схеми та технічна документація.

Структурна схема — схема, яка визначає основні функціональні частини виробу, їх взаємозв'язки та призначення.

Приклади графічних систем алгоритмів (ГСА)



Під функціональною частиною розуміють складову частину схеми:елемент, пристрій, функціональну групу, функціональну ланку. Із структурної схеми повинно бути зрозуміло, навіщо потрібний даний пристрій і як він працює в основних режимах роботи, як взаємодіють його частини.

СТРУКТУРНИЙ СИНТЕЗ ЦИФРОВИХ АВТОМАТІВ

Після абстрактного синтезу автомата, що закінчується мінімізацією числа внутрішніх станів, слідує етап структурного синтезу. Метою його є **побудова схеми автомата із логічних елементів заданого типу.** Якщо абстрактний автомат був лише математичною моделлю дискретного пристрою, то структурному автоматі необхідно враховувати структуру вхідних і вихідних сигналів, а також внутрішню будова автомата на рівні структурної чи функціональної (логічної) схеми. Основним завданням структурної теорії автоматів є **побудова композиції автоматів, тобто методів побудови складних автоматів із простіших автоматів.**

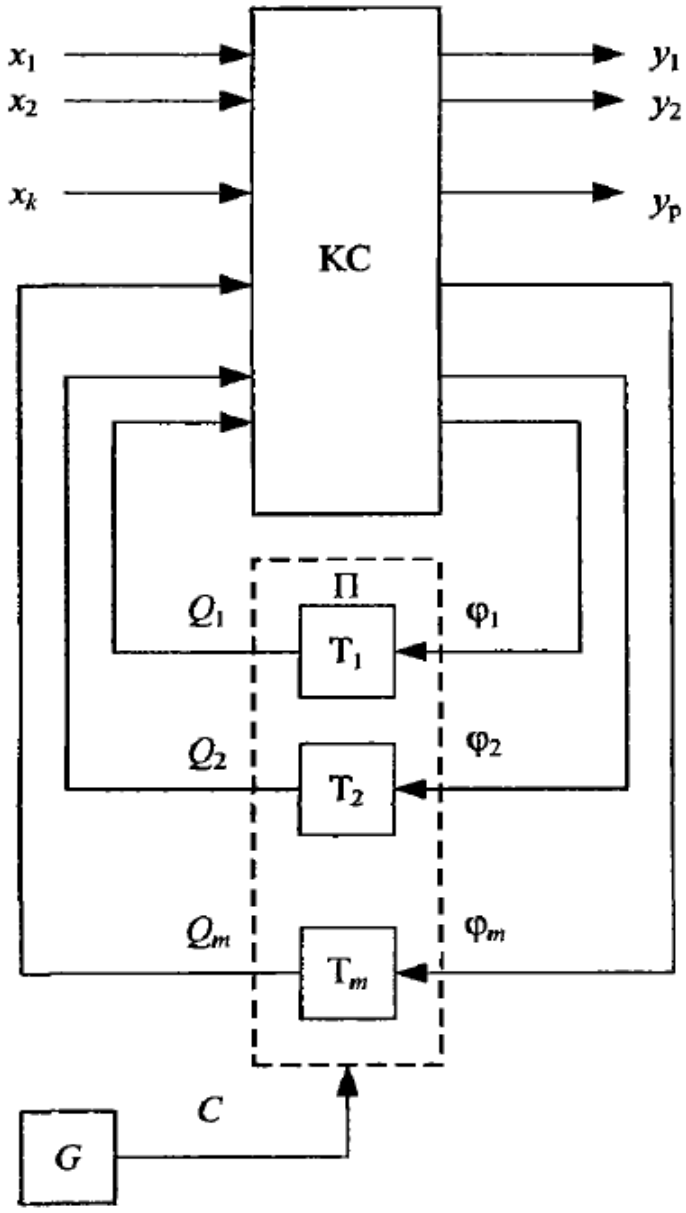
Елементарні автомати поділяють на **2** класи.

1-й –автомати з пам’яттю (**елементи пам’яті, запам’ятовуючі елементи**).

2-й - автомати без пам’яті (**комбінаційні**).

Тригери

Комбінаційна схема зі зворотними зв’язками, що має **два стійких стани** та **призначена для записування та зберігання одного біта інформації**, називається елементарним автоматом або **тригером**. Під дією вхідних сигналів тригер переходить з одного стійкого стану в інший.



КАНОНІЧНИЙ МЕТОД СТРУКТУРНОГО СИНТЕЗУ

Приклад 1. Синтез повністю визначеного автомата Мілі на *D*-тригерах.

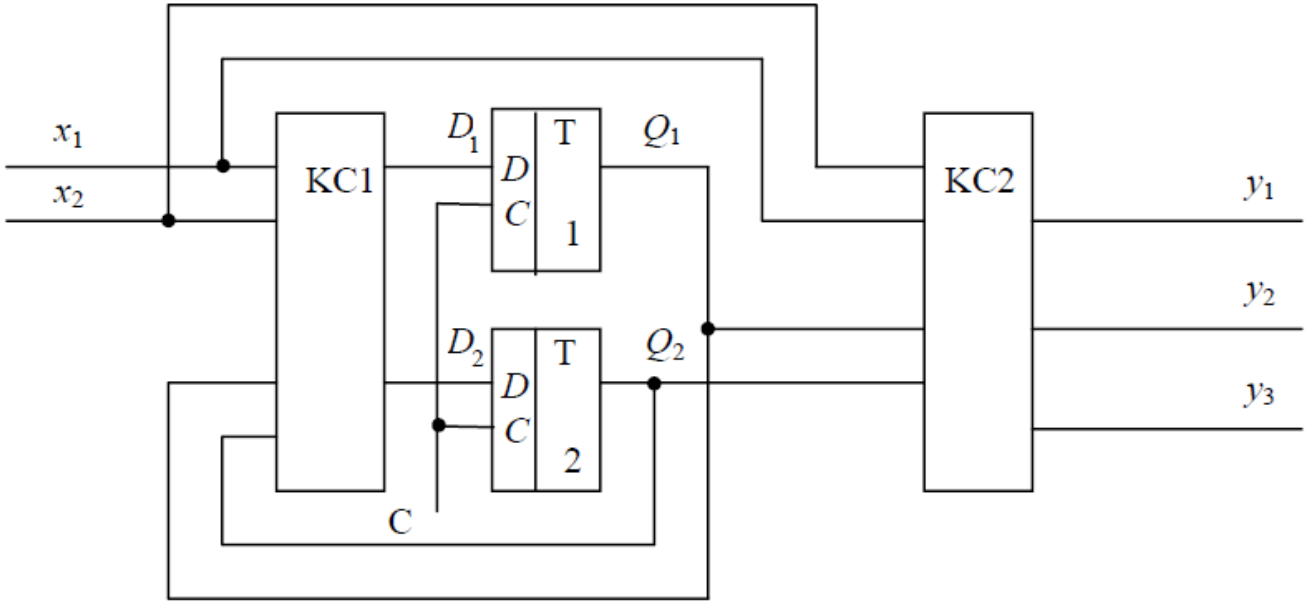
	a_1	a_2	a_3	a_4
z_1	a_1	a_3	a_1	a_2
z_2	a_3	a_2	a_4	a_1
z_3	a_2	a_4	a_3	a_2
z_4	a_3	a_4	a_1	a_4

	a_1	a_2	a_3	a_4
z_1	w_1	w_4	w_3	w_6
z_2	w_3	w_2	w_2	w_4
z_3	w_5	w_1	w_5	w_1
z_4	w_1	w_4	w_6	w_2

1. Визначення кількості структурних сигналів та елементів пам’яті.

	Кількість абстрактних сигналів чи станів	Кількість структурних сигналів	Найменування бітів
Вхідних	F=4	$L \geq \log_2 F = \log_2 4 = 2$	x_1, x_2
Вихідних	G=6	$N \geq \log_2 G = \log_2 6 = 3$	y_1, y_2, y_3
Пам’ять	M=4	$R \geq \log_2 M = \log_2 4 = 2$	Q_1, Q_2

2. Структурна схему цифрового автомата



3. Кодування вхідних, вихідних сигналів та внутрішніх станів автомата.

В загальному випадку, здійснюється довільно: кожному з абстрактних сигналів можна поставити у відповідність будь-яку комбінацію відповідних йому структурних сигналів. Необхідно тільки, щоб різні абстрактні сигнали кодувалися різними комбінаціями структурних.

Кодування вхідних сигналів

Вхідні сигнали	Код вхідних сигналів	
	x_1	x_2
z_1	0	0
z_2	0	1
z_3	1	0
z_4	1	1

Кодування станів автомата

Стани автомата	Код стану	
	Q_1	Q_2
a_1	0	0
a_2	0	1
a_3	1	0
a_4	1	1

Кодування вихідних сигналів

Вихідні сигнали	Код вихідних сигналів		
	y_1	y_2	y_3
w_1	0	0	0
w_2	0	0	1
w_3	0	1	0
w_4	0	1	1
w_5	1	0	0
w_6	1	0	1

4. Будуємо кодовані таблиці переходів та виходів структурного автомата

Q_1Q_2 / x_1x_2	00	01	10	11
00	00	10	00	01
01	10	01	11	00
10	01	11	10	01
11	10	11	00	11

Q_1Q_2 / x_1x_2	00	01	10	11
00	000	011	010	101
01	010	001	001	011
10	100	000	100	000
11	000	011	101	001

5. Будуємо структурну таблицю автомату

Входи	**Стан *t***	**Стан *t*+1**	**Виходи**	**Функції збудження**
$x_1\ x_2$	$Q_2^t\ Q_1^t$	$Q_2^{t+1}\ Q_1^{t+1}$	$y_1\ y_2\ y_3$	$D_2\ D_1$
0 0	0 0	0 0	0 0 0	0 0
0 1	0 0	0 1	0 1 0	0 1
1 0	0 0	1 0	1 0 0	1 0
1 1	0 0	0 1	0 0 0	0 1
0 0	1 0	0 1	0 1 1	0 1
0 1	1 0	1 0	0 0 1	1 0
1 0	1 0	1 1	0 0 0	1 1
1 1	1 0	1 1	0 1 1	1 1
0 0	0 1	0 0	0 1 0	0 0
0 1	0 1	1 1	0 0 1	1 1
1 0	0 1	0 1	1 0 0	0 1
1 1	0 1	0 0	1 0 1	0 0
0 0	1 1	1 0	1 0 1	1 0
0 1	1 1	0 0	0 1 1	0 0
1 0	1 1	1 0	0 0 0	1 0
1 1	1 1	1 1	0 0 1	1 1

Таблиця входів

Стан		*D*- тригер
Q_t	Q_{t+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

6. Знаходимо функції виходів з структурної таблиці (або таблиці виходів)

для Мілі

$$w(t) = \lambda(a(t), z(t))$$

ДДНФ

$$y_1 = x_1\bar{x}_2\bar{Q}_1\bar{Q}_2 + x_1\bar{x}_2Q_1\bar{Q}_2 + x_1x_2Q_1\bar{Q}_2 + \bar{x}_1\bar{x}_2Q_1Q_2,$$

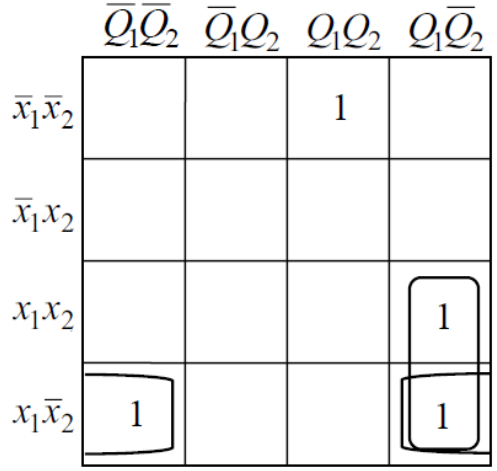
$$y_2 = \bar{x}_1x_2\bar{Q}_1\bar{Q}_2 + \bar{x}_1\bar{x}_2\bar{Q}_1Q_2 + x_1x_2\bar{Q}_1Q_2 + \bar{x}_1\bar{x}_2Q_1\bar{Q}_2 + \bar{x}_1x_2Q_1Q_2,$$

$$y_3 = \bar{x}_1\bar{x}_2\bar{Q}_1Q_2 + \bar{x}_1x_2\bar{Q}_1Q_2 + x_1x_2\bar{Q}_1Q_2 + \bar{x}_1x_2Q_1\bar{Q}_2 + \bar{x}_1\bar{x}_2Q_1Q_2 +$$

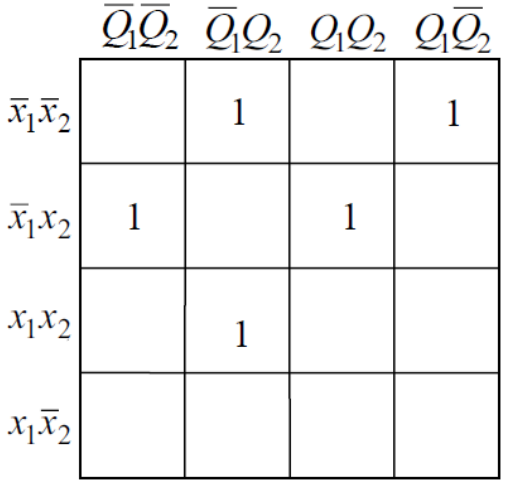
$$+ \bar{x}_1x_2Q_1Q_2 + x_1x_2Q_1\bar{Q}_2 + x_1x_2Q_1Q_2.$$

Спрощуємо функції за допомогою карт Карно

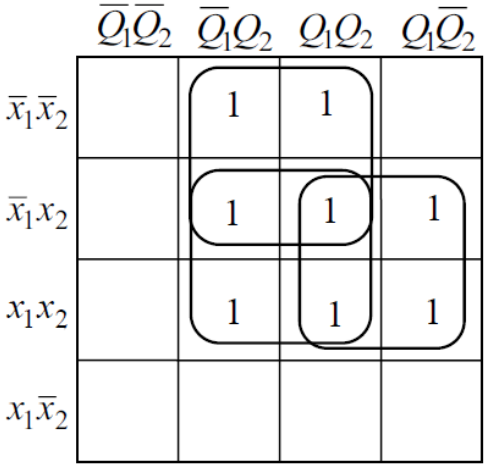
y_1



y_2



y_3



МКНФ

$$y_1 = x_1\bar{x}_2\bar{Q}_2 + x_1Q_1\bar{Q}_2 + \bar{x}_1\bar{x}_2Q_1Q_2,$$

$$y_2 = \bar{x}_1x_2\bar{Q}_1\bar{Q}_2 + \bar{x}_1\bar{x}_2\bar{Q}_1Q_2 + x_1x_2\bar{Q}_1Q_2 + \bar{x}_1\bar{x}_2Q_1\bar{Q}_2 + \bar{x}_1x_2Q_1Q_2,$$

$$y_3 = \bar{x}_1Q_2 + x_2Q_2 + x_2Q_1.$$

7. Знаходимо функції входів для двох тригерів D_1 та D_2, використовуючи **таблицю функцій збудження (входів)** тригерів відповідного типу.

$$D_1 = \bar{x}_1 x_2 \bar{Q}_1 \bar{Q}_2 + x_1 x_2 \bar{Q}_1 \bar{Q}_2 + \bar{x}_1 \bar{x}_2 \bar{Q}_1 Q_2 + x_1 \bar{x}_2 \bar{Q}_1 Q_2 + x_1 x_2 \bar{Q}_1 Q_2 + \bar{x}_1 x_2 Q_1 \bar{Q}_2 + x_1 \bar{x}_2 Q_1 \bar{Q}_2 + x_1 x_2 Q_1 Q_2,$$

$$D_2 = x_1 \bar{x}_2 \bar{Q}_1 \bar{Q}_2 + \bar{x}_1 x_2 \bar{Q}_1 Q_2 + x_1 \bar{x}_2 \bar{Q}_1 Q_2 + x_1 x_2 \bar{Q}_1 Q_2 + \bar{x}_1 x_2 Q_1 \bar{Q}_2 + \bar{x}_1 \bar{x}_2 Q_1 Q_2 + x_1 \bar{x}_2 Q_1 Q_2 + x_1 x_2 Q_1 Q_2.$$

D_1

	$\bar{Q}_1\bar{Q}_2$	$\bar{Q}_1 Q_2$	$Q_1 Q_2$	$Q_1\bar{Q}_2$
$\bar{x}_1\bar{x}_2$		1		
$\bar{x}_1 x_2$	1			1
$x_1 x_2$	1	1	1	
$x_1\bar{x}_2$		1		1

D_2

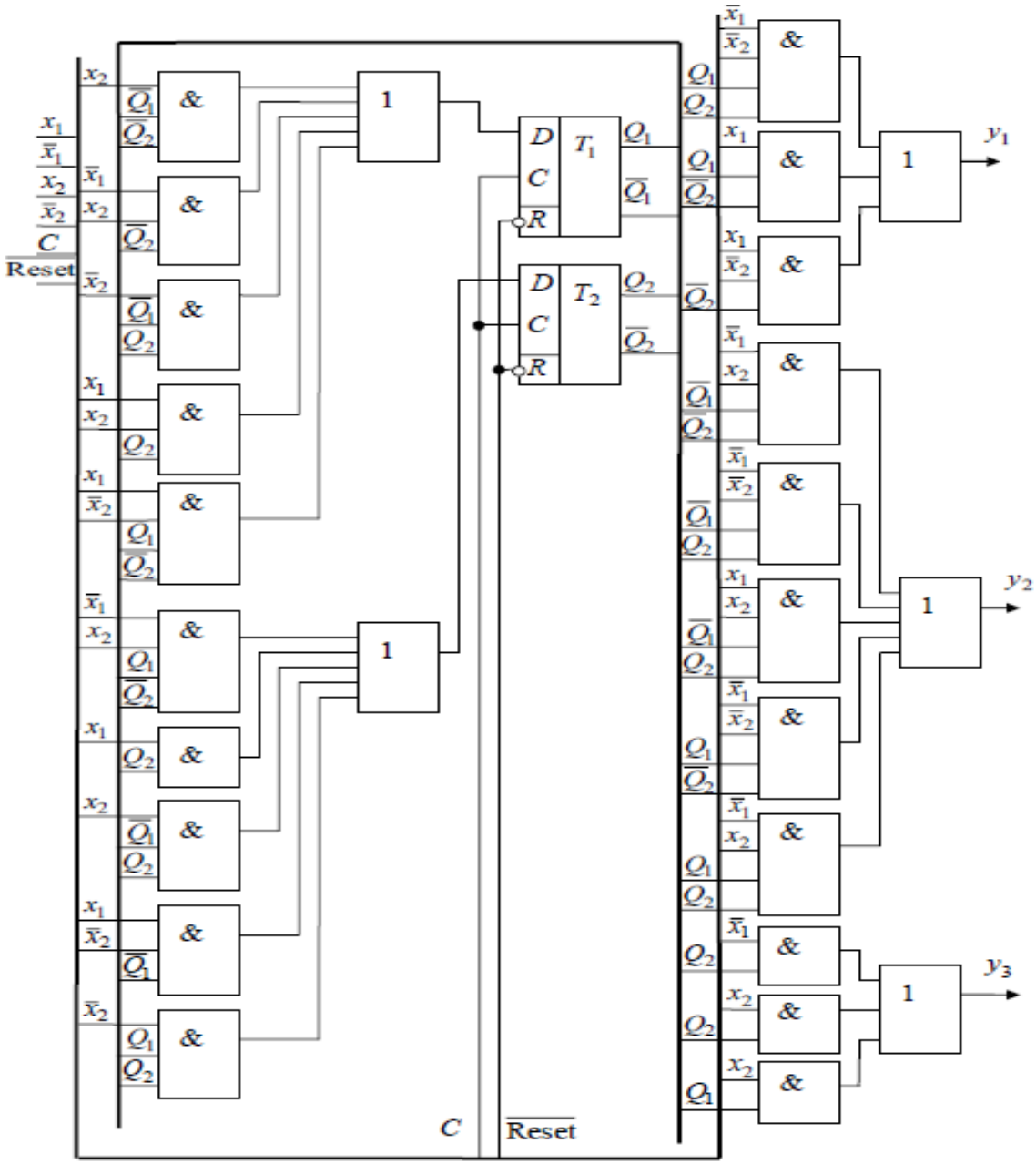
	$\bar{Q}_1\bar{Q}_2$	$\bar{Q}_1 Q_2$	$Q_1 Q_2$	$Q_1\bar{Q}_2$
$\bar{x}_1\bar{x}_2$			1	
$\bar{x}_1 x_2$		1		1
$x_1 x_2$		1	1	
$x_1\bar{x}_2$	1	1	1	

$$D_1 = x_2 \bar{Q}_1 \bar{Q}_2 + \bar{x}_1 x_2 \bar{Q}_2 + \bar{x}_2 \bar{Q}_1 Q_2 + x_1 x_2 Q_2 + x_1 \bar{x}_2 Q_1 \bar{Q}_2,$$

$$D_2 = x_1 Q_2 + x_2 \bar{Q}_1 Q_2 + x_1 \bar{x}_2 \bar{Q}_1 + \bar{x}_1 x_2 Q_1 \bar{Q}_2 + \bar{x}_2 Q_1 Q_2.$$

На підставі отриманих виразів будуємо функціональну схему автомата

Додатково показаний сигнал Reset , що встановлює автомат у початковий стан (у данному випадку 00).



Приклад 2. Синтез частково визначеного автомата Мілі на *RS*-тригерах.

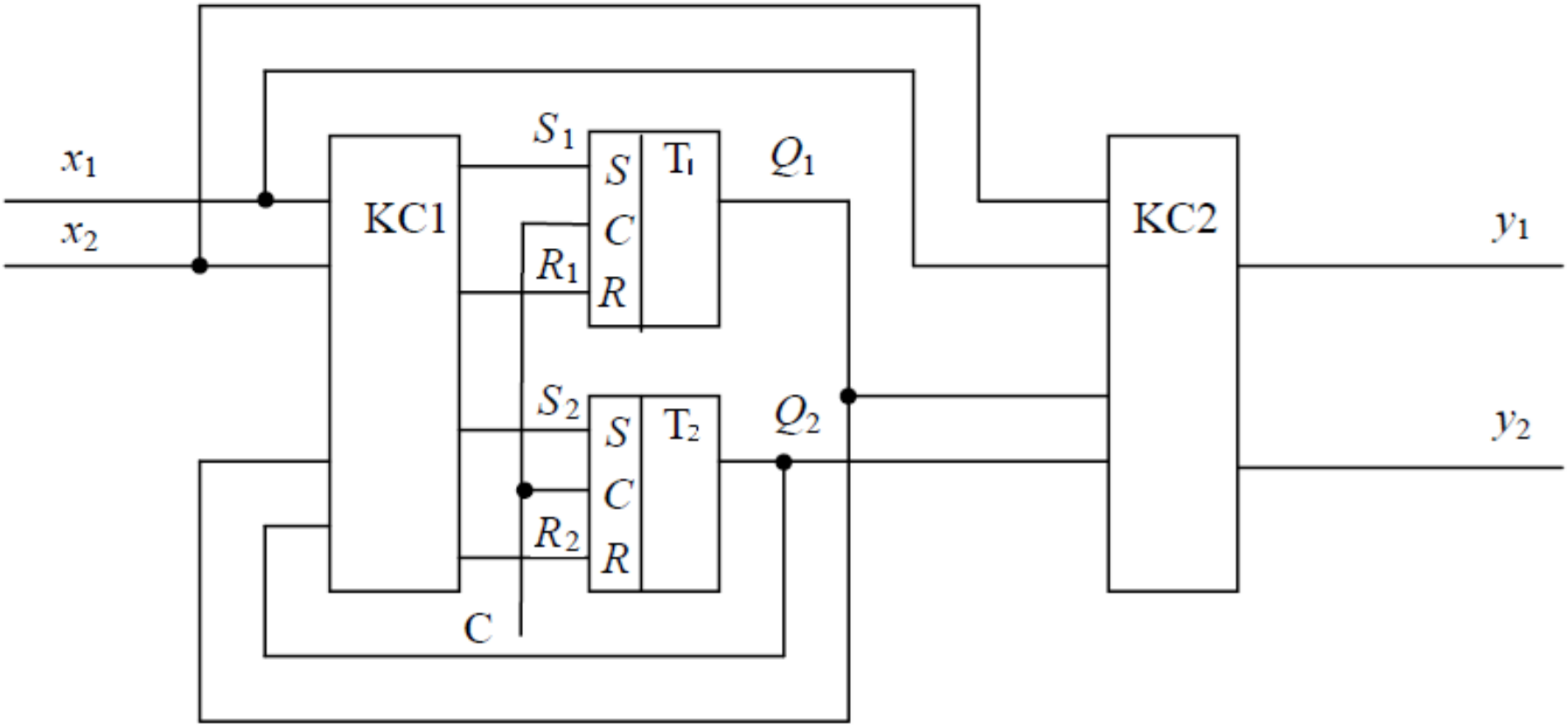
	a_1	a_2	a_3	a_4
z_1	a_2	-	a_4	-
z_2	a_3	a_2	-	a_2
z_3	a_4	a_1	a_1	a_3
z_4	-	a_3	a_2	-

	a_1	a_2	a_3	a_4
z_1	w_1	-	w_2	-
z_2	w_3	w_2	-	w_1
z_3	w_4	w_4	w_1	w_2
z_4	-	w_3	w_4	-

1. Визначення кількості структурних сигналів та элементів пам’яті.

	Кількість абстрактних сигналів чи станів	Кількість структурних сигналів	Найменування бітів
Вхідних	F=4	$L \geq \log_2 F = \log_2 4 = 2$	x_1, x_2
Вихідних	G=4	$N \geq \log_2 G = \log_2 4 = 2$	y_1, y_2
Пам’ять	M=4	$R \geq \log_2 M = \log_2 4 = 2$	Q_1, Q_2

2. Структурна схему цифрового автомата



3. Кодування вхідних, вихідних сигналів та внутрішніх станів автомата.

Вхідні сигнали	Код вхідних сигналів	
	x_1	x_2
z_1	0	0
z_2	0	1
z_3	1	0
z_4	1	1

Стани автомата	Код стану	
	Q_1	Q_2
a_1	0	0
a_2	0	1
a_3	1	0
a_4	1	1

Вихідні сигнали	Код вихідних сигналів	
	y_1	y_2
w_1	0	0
w_2	0	1
w_3	1	0
w_4	1	1

4. Будуємо кодовані таблиці переходів та виходів структурного автомата

Закодована таблиця переходів частково визначеного автомата Мілі

x_1x_2 \ Q_1Q_2	00	01	10	11
00	01	-	11	-
01	10	01	-	01
10	11	00	00	10
11	-	10	01	-

Закодована таблиця виходів частково визначеного автомата Мілі

x_1x_2 \ Q_1Q_2	00	01	10	11
00	00	-	01	-
01	10	01	-	00
10	11	11	00	01
11	-	10	11	-

5. Будуємо структурну таблицю автомату

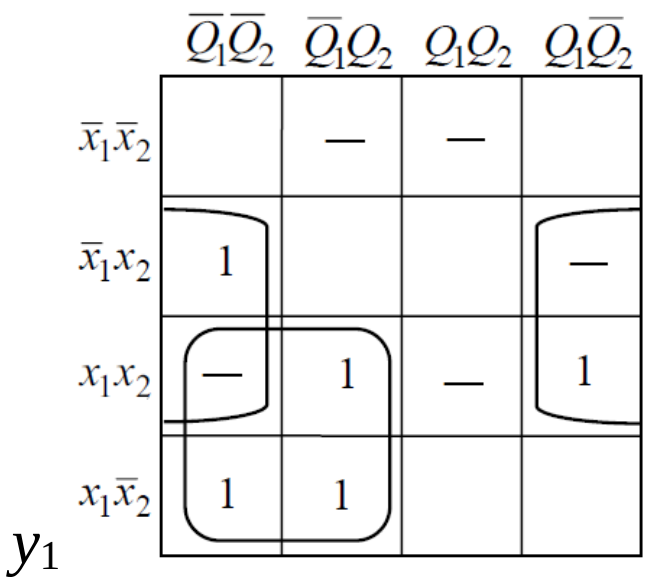
Входи	**Стан *t***	**Стан *t*+1**	**Виходи**	**Функції збудження**	
$x_1\ x_2$	$Q_2^t\ Q_1^t$	$Q_2^{t+1}\ Q_1^{t+1}$	$y_1\ y_2$	$S_2\ R_2$	$S_2\ R_1$
0 0	0 0	1 0	0 0	1 0	0 *
0 1	0 0	0 1	1 0	0 *	1 0
1 0	0 0	1 1	1 1	1 0	1 0
0 1	1 0	1 0	0 1	* 0	0 *
1 0	1 0	0 0	1 1	0 1	0 *
1 1	1 0	0 1	1 0	0 1	1 0
0 0	0 1	1 1	0 1	1 0	* 0
1 0	0 1	0 0	0 0	0 *	0 1
1 1	0 1	1 0	1 1	1 0	0 1
0 1	1 1	1 0	0 0	* 0	0 1
1 0	1 1	0 1	0 1	* 0	* 0

Стан		*SR*- тригер	
Q_t	Q_{t+1}	S	R
0	0	0	*
0	1	1	0
1	0	0	1
1	1	*	0

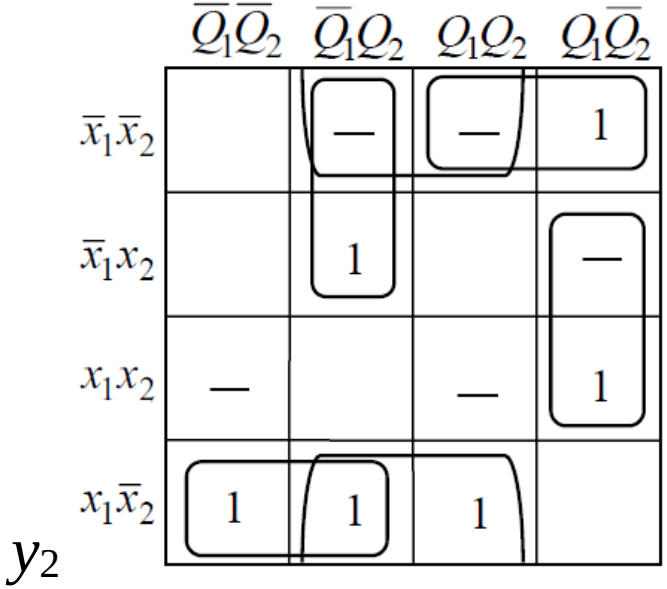
6. Знаходимо функції виходів з структурної таблиці (або таблиці виходів)

$$y_1 = \bar{x}_1 x_2 \bar{Q}_1 \bar{Q}_2 + x_1 \bar{x}_2 \bar{Q}_1 \bar{Q}_2 + x_1 \bar{x}_2 \bar{Q}_1 Q_2 + x_1 x_2 \bar{Q}_1 Q_2 + x_1 x_2 Q_1 \bar{Q}_2,$$
$$y_2 = x_1 \bar{x}_2 \bar{Q}_1 \bar{Q}_2 + \bar{x}_1 x_2 \bar{Q}_1 Q_2 + x_1 \bar{x}_2 \bar{Q}_1 Q_2 + \bar{x}_1 \bar{x}_2 Q_1 \bar{Q}_2 + x_1 x_2 Q_1 \bar{Q}_2 + x_1 \bar{x}_2 Q_1 Q_2.$$

Спрощуємо функції за допомогою карт Карно



y_1



y_2

$$y_1 = x_2 \bar{Q}_2 + x_1 \bar{Q}_1,$$
$$y_2 = x_1 \bar{x}_2 \bar{Q}_1 + \bar{x}_1 \bar{Q}_1 Q_2 + \bar{x}_1 \bar{x}_2 Q_1 + x_2 Q_1 \bar{Q}_2 + \bar{x}_2 Q_2.$$

7. Знаходимо функції входів для двох *RS*-тригерів T_1 та T_2, використовуючи **таблицю функцій збудження (входів)** тригерів відповідного типу.

$$R_1 = x_1\bar{x}_2Q_1\bar{Q}_2 + x_1x_2Q_1\bar{Q}_2 + \bar{x}_1x_2Q_1Q_2, \qquad S_1 = \bar{x}_1x_2\bar{Q}_1\bar{Q}_2 + x_1\bar{x}_2\bar{Q}_1\bar{Q}_2 + x_1x_2\bar{Q}_1Q_2,$$

$$R_2 = x_1\bar{x}_2\bar{Q}_1Q_2 + x_1x_2\bar{Q}_1Q_2 + x_1\bar{x}_2Q_1Q_2, \qquad S_2 = \bar{x}_1\bar{x}_2\bar{Q}_1\bar{Q}_2 + x_1\bar{x}_2\bar{Q}_1\bar{Q}_2 + \bar{x}_1\bar{x}_2Q_1\bar{Q}_2 + x_1x_2Q_1\bar{Q}_2.$$

R_1

x_1x_2 \ Q_1Q_2	$\bar{Q}_1\bar{Q}_2$	$\bar{Q}_1Q_2$	Q_1Q_2	$Q_1\bar{Q}_2$
$\bar{x}_1\bar{x}_2$	*	—	—	
$\bar{x}_1x_2$		*	1	—
x_1x_2	—		—	1
$x_1\bar{x}_2$		*		1

S_1

x_1x_2 \ Q_1Q_2	$\bar{Q}_1\bar{Q}_2$	$\bar{Q}_1Q_2$	Q_1Q_2	$Q_1\bar{Q}_2$
$\bar{x}_1\bar{x}_2$		—	—	*
$\bar{x}_1x_2$	1			—
x_1x_2	—	1	—	
$x_1\bar{x}_2$	1		*	

R_1

x_1x_2 \ Q_1Q_2	$\bar{Q}_1\bar{Q}_2$	$\bar{Q}_1Q_2$	Q_1Q_2	$Q_1\bar{Q}_2$
$\bar{x}_1\bar{x}_2$		—	—	
$\bar{x}_1x_2$	*			—
x_1x_2	—	1	—	
$x_1\bar{x}_2$		1	1	*

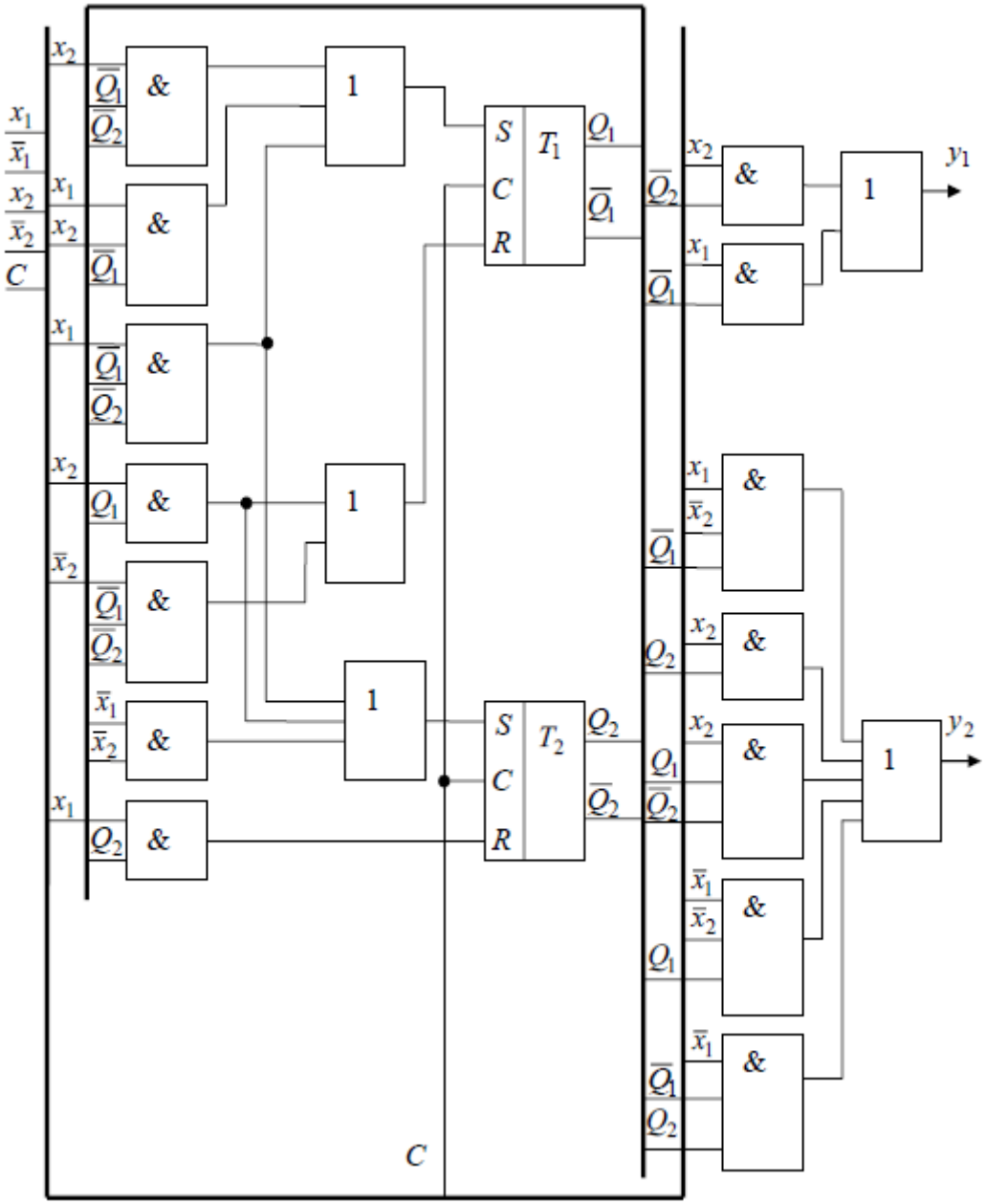
S_1

x_1x_2 \ Q_1Q_2	$\bar{Q}_1\bar{Q}_2$	$\bar{Q}_1Q_2$	Q_1Q_2	$Q_1\bar{Q}_2$
$\bar{x}_1\bar{x}_2$	1	—	—	1
$\bar{x}_1x_2$		*	*	—
x_1x_2	—		—	1
$x_1\bar{x}_2$	1			

$$R_1 = x_2Q_1 + x_1Q_1\bar{Q}_2,$$
$$S_1 = x_2\bar{Q}_1\bar{Q}_2 + x_1\bar{Q}_1\bar{Q}_2 + x_1x_2\bar{Q}_1,$$

$$R_2 = x_1Q_2,$$
$$S_2 = \bar{x}_1\bar{x}_2 + \bar{x}_2\bar{Q}_1\bar{Q}_2 + x_2Q_1.$$

На підставі отриманих виразів будуємо функціональну схему автомата



Приклад 3. Синтез автомата Мура на *T*- тригерах.

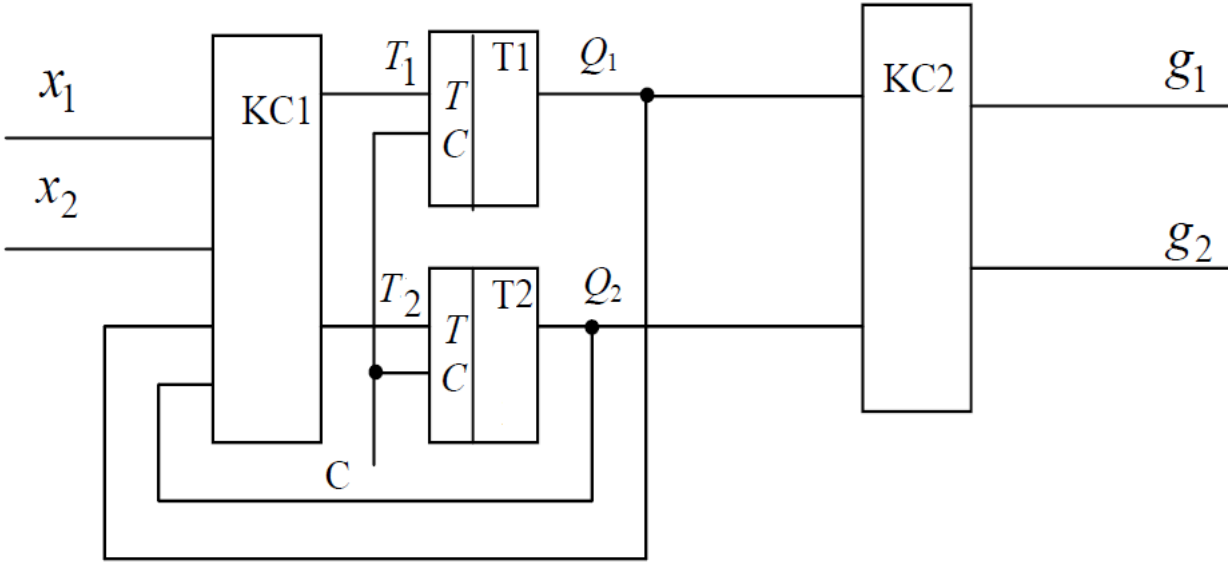
	u_1	u_2	u_3	u_4
	a_1	a_2	a_3	a_4
z_1	a_2	a_3	a_2	a_1
z_2	a_1	a_1	a_4	a_2
z_3	a_4	a_4	a_1	a_4

1. Визначення кількості структурних сигналів та елементів пам’яті.

	Кількість абстрактних сигналів чи станів	Кількість структурних сигналів	Найменування бітів
Вхідних	F=3	$L = 2 \geq \log_2 F = \log_2 3 = 1{,}58$	x_1, x_2
Вихідних	G=4	$N \geq \log_2 G = \log_2 4 = 2$	y_1, y_2
Пам’ять	M=4	$R \geq \log_2 M = \log_2 4 = 2$	Q_1, Q_2

Для автомата Мура вихідних сигналів **не може бути більше, ніж станів**

2. Структурна схему цифрового автомата



3. Кодування сигналів та внутрішніх станів автомата.

Вхідні сигнали	Код вхідних сигналів	
	x_1	x_2
z_1	0	0
z_2	0	1
z_3	1	0

Стани автомата	Код стану	
	Q_1	Q_2
a_1	0	0
a_2	0	1
a_3	1	0
a_4	1	1

Вихідні сигнали	Код вихідних сигналів	
	g_1	g_2
u_1	0	0
u_2	0	0
u_3	0	1
u_4	0	1

Таблиця переходів

x_1x_2 \ Q_1Q_2	00	01	10	11
00	01	10	01	00
01	00	00	11	01
10	11	11	00	11

4. Будуємо структурну таблицю автомату

Входи	**Стан *t***	**Стан *t*+1**	**Виходи**	**Функції збудження**	
$x_1\ x_2$	$Q_2^t\ Q_1^t$	$Q_2^{t+1}\ Q_1^{t+1}$	$g_1\ g_2$	T_2	T_1
0 0	0 0	1 0	0 0	1	0
0 1	0 0	0 0	0 0	0	0
1 0	0 0	1 1	0 0	1	1
0 0	1 0	0 1	0 1	1	1
0 1	1 0	0 0	0 1	1	0
1 0	1 0	1 1	0 1	0	1
0 0	0 1	1 0	1 0	1	1
0 1	0 1	1 1	1 0	1	0

Таблиця функції входів Т-тригера

Q^t	Q^{t+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

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

5. Знаходимо функції виходів автомата Мура $w(t) = \lambda(a(t))$ (**не залежить від *x***)

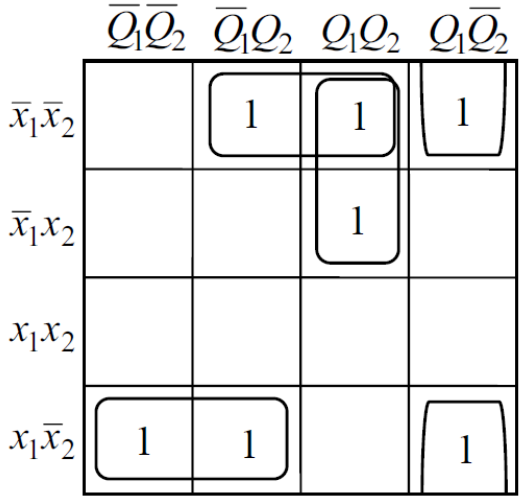
$$g_1 = Q_1\overline{Q}_2 + Q_1Q_2 = Q_1, \quad g_2 = \overline{Q}_1Q_2 + Q_1Q_2 = Q_2.$$

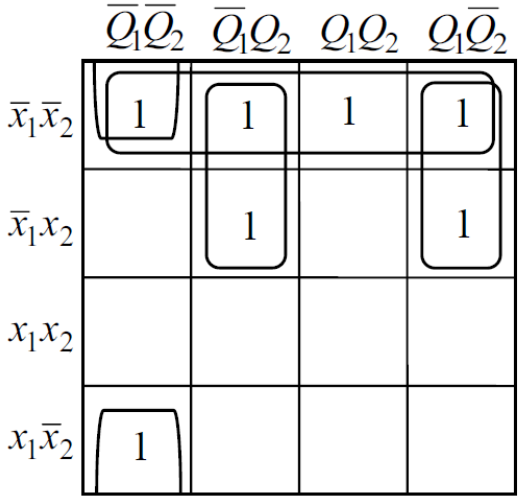
6. Знаходимо функції входів для двох *T*-тригерів T_1 та T_2, використовуючи **таблицю функцій збудження (входів)** тригерів відповідного типу.

$$T_1 = x_1\bar{x}_2\overline{Q}_1\overline{Q}_2 + \bar{x}_1\bar{x}_2\overline{Q}_1Q_2 + x_1\bar{x}_2\overline{Q}_1Q_2 + \bar{x}_1\bar{x}_2Q_1\overline{Q}_2 + x_1\bar{x}_2Q_1\overline{Q}_2 + \bar{x}_1\bar{x}_2Q_1Q_2 + \bar{x}_1x_2Q_1Q_2,$$

$$T_2 = \bar{x}_1\bar{x}_2\overline{Q}_1\overline{Q}_2 + x_1\bar{x}_2\overline{Q}_1\overline{Q}_2 + \bar{x}_1\bar{x}_2\overline{Q}_1Q_2 + \bar{x}_1x_2\overline{Q}_1Q_2 + \bar{x}_1\bar{x}_2Q_1\overline{Q}_2 + \bar{x}_1x_2Q_1\overline{Q}_2 + \bar{x}_1\bar{x}_2Q_1Q_2.$$

T_1 T_2

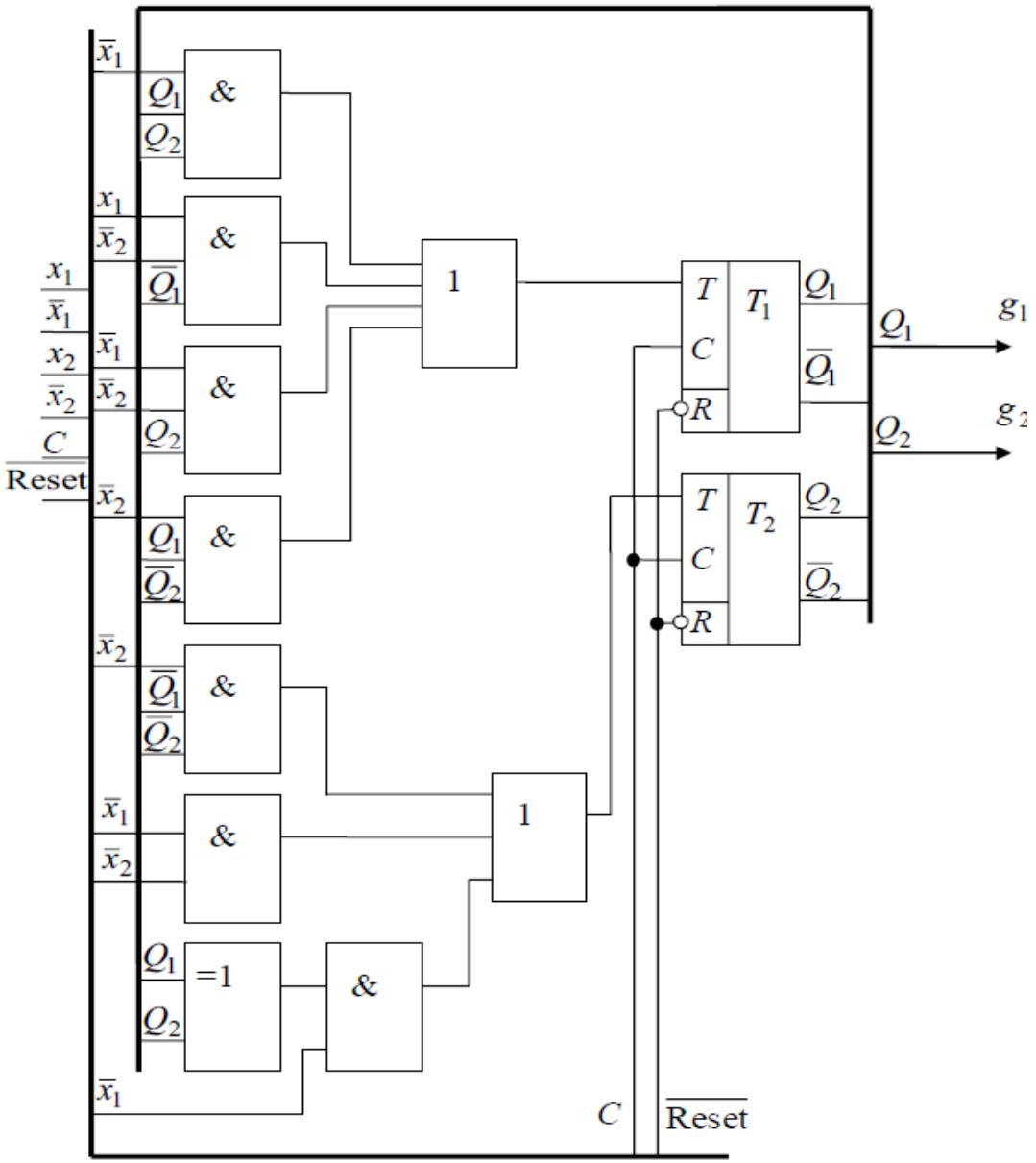




$$T_1 = x_1\bar{x}_2\bar{Q}_1 + \bar{x}_1\bar{x}_2Q_2 + \bar{x}_2Q_1\bar{Q}_2 + \bar{x}_1Q_1Q_2,$$

$$T_2 = \bar{x}_2\bar{Q}_1\bar{Q}_2 + \bar{x}_1\bar{x}_2 + \bar{x}_1\bar{Q}_1Q_2 + \bar{x}_1Q_1\bar{Q}_2 = \bar{x}_2\bar{Q}_1\bar{Q}_2 + \bar{x}_1\bar{x}_2 + \bar{x}_1(Q_1 \oplus Q_2).$$

На підставі отриманих виразів будуємо функціональну схему автомата



Лекція 9

КОМП’ЮТЕРНА АРИФМЕТИКА. ЗАГАЛЬНІ ПОНЯТТЯ ПРО СИСТЕМИ ЧИСЛЕННЯ

ПЕРЕВЕДЕННЯ ЧИСЕЛ З ОДНИХ СИСТЕМ ЧИСЛЕННЯ В ІНШІ

ПОДАННЯ ЧИСЕЛ В ЗАСОБАХ ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

КОМП’ЮТЕРНА АРИФМЕТИКА

КОМП’Ю́ТЕРНА АРИФМЕ́ТИКА – сукупність принципів і форм подання числової інформації, методів і алгоритмів виконання арифметичних операцій та обчислення елементарних функцій, що розглядаються на рівні внутрішньої структурної організації технічних засобів **комп’ютерів і комп’ютерних систем (КіКС)**. Є частиною **обчислювальної математики**, яка орієнтована на логічний рівень описання обчислювальних структур і процесів у них, і вивчає комп’ютерні аспекти теорії систем числення, комп’ютерно-орієнтовані алгоритми переводу чисел з однієї системи числення в інші, форми та способи комп’ютерного подання чисел, алгоритми виконання арифметичних операцій і обчислення елементарних функцій у КіКС.

ЗАГАЛЬНІ ПОНЯТТЯ ПРО СИСТЕМИ ЧИСЛЕННЯ

Системою числення називається сукупність заходів і правил для найменування і позначення чисел. Умовні знаки, що використовуються для позначення чисел називаються **цифрами**. Далі будемо припускати, що кількість цифр кінцева, тобто абетка, на основі якої складаються числа в деякій системі числення, складається з кінцевого числа елементів (цифр).

Системи числення поділяються на два класи: **непозиційні** і **позиційні**.

Непозиційною називають систему числення, в якій **значенню кожної цифри у будь–якому місці запису числа існує один і той же кількісний еквівалент**. Такі системи з’явилися раніше в історичному плані, наприклад, римська нумерація. У непозиційних системах дуже складниі і громіздки алгоритми подання чисел і виконання арифметичних операцій.

Системи, в яких **значення кожної цифри залежить від місця (позиції) у послідовності цифр при записі числа**, носять назву **позиційних**. Позиційною системою числення є звичайна десяткова система числення. Позиційні системи числення мають найменування, яке співпадає з кількістю цифр, що в ній використовуються. Наприклад, в десятковій системі числення використовується десять цифр від 0 до 9. а число 123 визначається як $1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0$. З цього прикладу ми бачимо, що позиція кожної цифри має свій ваговий коефіцієнт, і перша позначає кількість сотень, а остання – кількість одиниць.

У сучасних цифрових ЕОМ **використовуються** головним чином **позиційні системи числення**, тому що в них простіше реалізуються правила, ніж в непозиційних системах

Основними характеристиками позиційних систем числення є:

- основа системи числення – P;
- абетка цифр системи числення (кількість використовуваних в системі цифр);
- вага розрядів – R_i, де i=0, ..., n-1 – розрядність числа.

Основа позиційної системи числення є число, яке виявляє у скільки разів одиниця старшого розряду більша одиниці сусіднього молодшого розряду.

Абетка цифр позиційної системи числення характеризує значення цифр, які використовуються для зображення чисел у даній системі числення. Цифри обираються таким чином, щоб вони складали відрізок натурального ряду чисел, включаючи число “0”. У цьому випадку максимальне значення цифри, яка використовується у системі числення, дорівнює P-1.

Вага розряду визначає кількісне значення цифри у зображенні числа, відношення кількісного еквівалента цифри, яка стоїть в *i*-му розряді a_i, до кількісного еквіваленту тієї ж цифри, яка стоїть у нульовому розряді.

Число 34043,9145

Десяткове число	3	4	0	4	3	9	1	4	5
Номери розрядів	4	3	2	1	0	-1	-2	-3	-4
Вага розрядів	10^{4}	10^{3}	10^{2}	10^{1}	10^{0}	10^{-1}	10^{-2}	10^{-3}	10^{-4}

$$A_{(p)} = a_{n-1}p^{n-1} + a_{n-2}p^{n-2} + \cdots + a_1p^1 + a_0p^0 + a_{-1}p^{-1} + a_{-2}p^{-2} + \cdots$$

$$+ a_{-m}p^{-m} = \sum_{i=-m}^{n-1} a_i\ p^i$$

В якості основи систем числення може бути взяте **будь-яке відмінне від одиниці ціле число**. Розрізняють: двійкову, трійкову, п’ятіркову, вісімкову, шістнадцяткову та ін. системи. Їх можна побудувати незліченну кількість, але для використання у ЕОМ необхідно враховувати вимоги:

- однозначність подання чисел;
- можливість подання будь – якого числа із заданого діапазону чисел;
- простота виконання арифметичних і логічних операцій;
- зручність вводу початкових даних і виводу результатів обчислень;
- зручність відтворення кожної цифри стійкими станами декотрої фізичної системи. Кількість стійких станів фізичної системи повинно дорівнювати кількості цифр в системі числення, яку передбачають використовувати у цифровій ЕОМ.

Перерахованим вище вимогам у великій мірі задовольняє **двійкова** і **двійково–десяткова** системи числення. Вони потребує тільки два стійких стани для подання цифр **0** і **1**. Основа двійкової системи числення записується як 10, тобто $2_{(10)} = 10_{(2)}$ і читається: “один, нуль”.

$$A_{(2)} = \sum_{i=-m}^{n-1} a_i \cdot 10^i_{(2)}$$

$$A_{(2)} = 1101{,}101 = 1 \cdot 10^{011}_{(2)} + 1 \cdot 10^{010}_{(2)} + 0 \cdot 10^{001}_{(2)} +$$

$$+ 1 \cdot 10^{000}_{(2)} + 1 \cdot 10^{-001}_{(2)} + 0 \cdot 10^{-010}_{(2)} + 1 \cdot 10^{-011}_{(2)}.$$

$$A_{(10)} = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} =$$

$$= 8 + 2 + 0 + 1 + \frac{1}{2} + 0 + \frac{1}{8} = 13\frac{5}{8} = 13{,}625.$$

$$1101{,}101_{(2)} = 13{,}625_{(10)}$$

Вісімкова система числення

В цій системі числення для запису будь – якого числа застосовується перші вісім цифр десяткової абетки: 0, 1, 2, 3, 4, 5, 6, 7. $A_{(8)} = 3726,145$ і читається таким чином: три, сім, два, шість, кома, один, чотири, п'ять.

$$175,36_{(8)} = 1 \cdot 8^2 + 7 \cdot 8^1 + 5 \cdot 8^0 + 3 \cdot 8^{-1} + 6 \cdot 8^{-2} = 125\frac{15}{32}_{(10)}$$

Шістнадцяткова система числення

Використовують десяткові числа, а для зображення додаткових символів – початкові букви латинської абетки: **A, B, C, D, E, F**. $A_{(16)} = A5D,2B$

$$A_{(10)} = 10 \cdot 16^2 + 5 \cdot 16^1 + 13 \cdot 16^0 + 2 \cdot 16^{-1} + 11 \cdot 16^{-2} =$$

$$= 2560 + 80 + 13 + \frac{2}{16} + \frac{11}{256} = 2653\frac{43}{256}$$

Двійково – десяткова система числення

Кожна десяткова цифра записується у двійовій системі числення **«8-4-2-1»**.

$$849{,}05_{(10)} = 1000\ \ 0100\ \ 1001,\ \ 0000\ \ 0101_{(2\text{-}10)}$$

Всі цифри можуть бути поставлені і рядом.

Але ця система **незручна для виконання арифметичних операцій над десятковими числами**. Це пов’язано з труднощами виявлення перенесення в наступний десяткових розряд (більш старшу тетраду). Крім того, код прямого заміщення характеризується складністю переходу до **зворотних і додаткових кодів** для десяткових чисел, що полегшують виконання алгебраїчного додавання. Це пояснюється тим, що код прямого заміщення не є тим, що самодоповнюється, тобто інверсія його двійкових цифр не дає коду доповнення десяткової цифри до 9.

Двійково – десяткова система числення з надлишком

Формування цифр в цій системи числення відбувається методом **додавання до десяткової цифри числа 3 і подальшим поданням результату у вигляді двійкових тетрад.** Цей код самодоповнюється – **додавання до 9 отримується заміною 1 на 0 і навпаки 0 на 1**.

Перевагою кода є те, що **легко визначається перенесення в старшу двійкову тетраду**. Але код незручний для перетворення чисел з однієї системи числення в іншу.

Разом з двійково-десятковою системою числення з надлишком 3 в обчислювальних машинах широке застосування знаходять і **системи числення з надлишком 6**. Побудова цих систем відбувається по тій же схемі, що і система числення з надлишком 3.

	Код “8 – 4 – 2 – 1”	*Код з надлишком 3*	*Код з надлишком 6*	*Доповнення коду з надлишком 3 до 9*
0	0000	0011	0110	1100
1	0001	0100	0111	1011
2	0010	0101	1000	1010
3	0011	0110	1001	1001
4	0100	0111	1010	1000
5	0101	1000	1011	0111
6	0110	1001	1100	0110
7	0111	1010	1101	0101
8	1000	1011	1110	0100
9	1001	1100	1111	0011

ПЕРЕВЕДЕННЯ ЧИСЕЛ З ОДНИХ СИСТЕМ ЧИСЛЕННЯ В ІНШІ

Перевести число з одної системи числення в іншу – це означає знайти зображення цифр числа заданої системи в необхідній системі числення.

$$A_{(P_1)} \to B_{(P_2)}$$

$$A_{(P_1)} = \sum_{i=-m}^{n-1} a_i \cdot P_1^i, \quad \text{а} \quad B_{(P_2)} = \sum_{j=-l}^{K-1} b_j \cdot P_2^j,$$

$$\sum_{i=-m}^{n-1} a_i \cdot P_1^i = \sum_{j=-l}^{K-1} b_j \cdot P_2^j,$$

$$\{a_i\} = \overline{0, P_1 - 1}; \{b_j\} = \overline{0, P_2 - 1}.$$

Переведення <u>цілих чисел</u> діленням на основу нової системи числення

$$B_{(P_2)} = b_K \cdot P_2^K + b_{K-1} \cdot P_2^{K-1} + \ldots + b_1 \cdot P_2^1 + b_0 \cdot P_2^0$$

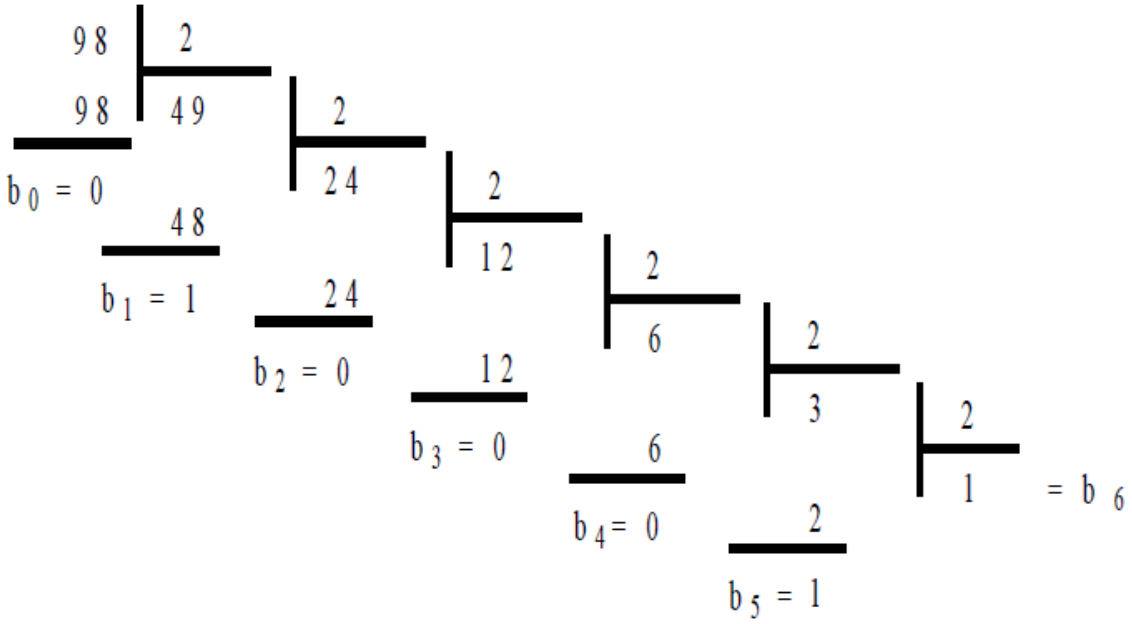
За схемою Горнера

$$B_{(P_2)} = (\ldots((b_K \cdot P_2 + b_{K-1}) \cdot P_2 + b_{K-2}) \cdot P_2 + b_{K-2}) \cdot P_2 + \ldots + b_1)P_2 + b_0.$$

Для переведення цілого числа з одної системи числення в іншу необхідно **послідовно ділити** це число і проміжкові частки **на основу нової системи** числення до тих пір, доки проміжкова частка не буде **менше основи нової системи числення**. **Остання частка і залишки у порядку, зворотному їх отриманню** є зображеннями цифр числа в новій системі числення

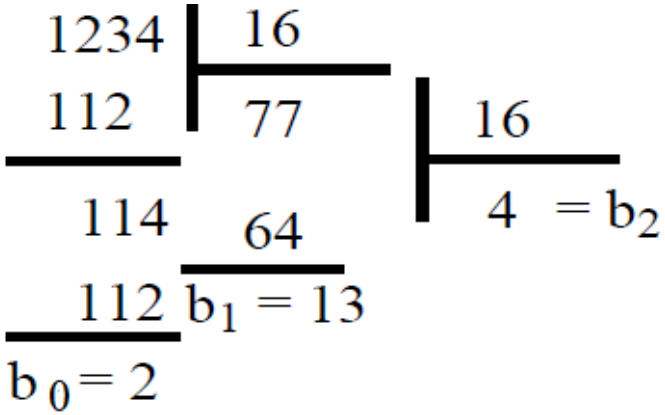
Для кінцевого запису числа $A_{(P1)}$ в P_2-річній системі числення необхідно кожний з отриманих коефіцієнтів bi записати одною P_2-річною цифрою.

Перевести десяткове число $A_{(10)} = 98$ у двійкову систему числення



$B_{(2)}=1100010$

Перевести число A(10)=1234 в шістнадцяткову систему числення.



$b_2=4_{(10)}$, $b_1=13_{(10)}$, $b_0=2_{(10)}$ $B_{(16)}=4D2$

Переведення дробових чисел множенням на основу нової системи

$$A_{(P_1)} = a_{-1}P_1^{-1} + a_{-2}P_1^{-2} + \ldots + a_{-m}P_1^{-m} \qquad B_{(P_2)} = b_{-1}P_2^{-1} + b_{-2}P_2^{-2} + \ldots + b_{-l}P_2^{-l}$$

За схемою Горнера $B_{(P_2)} = P_2^{-1}(b_{-1} + P_2^{-1}(b_{-2} + \ldots + P_2^{-1}(b_{-(l-1)} + P_2^{-1}b_{-l}))\ldots)$

Якщо праву частину виразу помножити на P_2, то знайдемо новий десятковий дріб, в цілій частині якого буде число b_{-1}. Потім помножити дробову частину, яка залишилася, знову на P_2, отримаємо дріб, в цілій частині якої буде b_{-2}. Повторюючи процес множення l раз, знайдемо все l цифр числа в новій системі числення. При цьому всі дії повинні виконуватися за правилами P_1-річної арифметики і, відповідно, в цілій частині отриманих дробів будуть проявлятися еквіваленти цифр нової системи числення, які записані в початковій системі числення.

При переведенні можна отримати дроби у вигляді нескінченності, або рядка, який розходиться. Процес можна закінчити, якщо появиться дробова частина, яка має у всіх розрядах нулі, або буде досягнута задана точність переведення (необхідна кількість розрядів). При переведенні дробів **необхідно вказувати кількість розрядів числа в новій системі числення**.

Для переведення дробу з однієї позиційної системи числення в іншу його необхідно **послідовно множити на основу нової системи числення** до того часу, поки в новому дробі не буде **потрібної кількості цифр**, яка визначається потрібною точністю представлення дробу.

Дріб в новій системі числення записується з цілих частин добутків, які отримані при послідовному множенні, причому перша ціла частина буде старшою цифрою нового дробу

$A_{(10)}$ = 0,625 в двійкову систему з точністю до четвертого знаку

0,	625
x	2
b_{-1}=1,	250
x	2
b_{-2}=0,	500
x	2
b_{-3}=1,	000
x	2
b_{-4}=0	000

$B_{(2)}$ = 0,1010

$A_{(10)}$=0,12 у вісімкову систему з точністю до шостого знака

0,	12
x	8
b_{-1}=0,	96
x	8
b_{-2}=7,	68
x	8
b_{-3}=5,	44
x	8
b_{-4}=3,	52
x	8
b_{-5}=4,	16
x	8
b_{-6}=1,	28

$B_{(8)}$ = 0,075341

Якщо $P_2<P_1$, то коефіцієнти **є цифрами P_2-річної системи** числення. Якщо $P_2>P_1$, то коефіцієнти які **необхідно замінити цифрами P_2-річної**.

Перевести $A_{(10)}=0,87$ у шістнадцяткову систему числення. $P_2>P_1$, тому

1. P_2 подається в початковій системі: $10_{(16)}=16_{(10)}$;
2. Виконується послідовне множення на основу $P_2=16_{(10)}$
3. Цілі частини переводяться у шістнадцяткову систему

$13_{(10)} = D_{(16)}$; $14_{(10)} = E_{(16)}$; $11_{(10)} = B_{(16)}$; $8_{(10)} = 8_{(16)}$

$B_{(16)}$ = 0,DEB8 с точністю до 4 знаку

0,	87
x	16
b_{-1}=13,	22
x	16
b_{-2}=14,	72
x	16
b_{-3}=11,	52
x	16
b_{-4}=8,	32

Для переведення **змішаного дробу** з однієї системи числення в іншу необхідно **окремо переводити цілу і дробову частини числа відповідно до методу ділення і методу множення** на основу нової системи числення.

На практиці метод частіше всього використовується для переведення чисел **з десяткової системи числення в інші системи**, так як арифметичні дії в цьому випадку робляться **у звичній для нас десятковій системі числення**.

Переведення **звичайних дробів** в двійкову систему числення робиться за розглянутим вище правилом **після переведення їх в десяткові дроби.**

Для переведення в двійкову систему числення **звичайних дробів,** знаменники яких є **ціла ступінь двійки**, потрібно перевести їх чисельник як ціле число, відділивши комою, починаючи з молодшого розряду, кількість цифр, що дорівнює **ступеню двійки в знаменнику дробу**.

$$A_{(10)} = \frac{13}{16}$$

13	2				
12	6	2			
$b_0 = 1$	6	3	2		
	$b_1 = 0$	2	1	$= b_3$	
		$b_2 = 1$			

$$\frac{13}{16_{(10)}} = 0{,}1101_{(2)} \qquad B_{(2)} = 0{,}1101$$

Метод безпосереднього заміщення

$$A_{(P1)} = a_{n-1}P_1^{n-1} + a_{n-2}P_1^{n-2} + \cdots + a_1P_1^1 + a_0P_1^0 + a_{-1}P_1^{-1} + \cdots + a_{-m}P_1^{-m} =$$
$$= b_{k-1}P_2^{k-1} + b_{k-2}P_2^{k-2} + \cdots + b_1P_2^1 + b_0P_2^0 + b_{-1}P_2^{-1} + \cdots + b_{-l}P_2^{-l} = B_{(P2)}$$

Правило переводу чисел цим методом полягають у наступному:

1. Задане число $A_{(P1)}$ подається у вигляді **зваженої суми**.

2. Цифри a_i і основа P_1 в правій частині виразу **записуються (заміщуються) в системі з основою** P_2. Якщо $P_2 > P_1$, то зображення цифр в P_2-річній системі співпадає з їх зображенням в P_1-річній.

3. Виконуються всі арифметичні операції в системі числення з основою P_2.

$$A_{(10)} = 35{,}25 \quad 35{,}25_{(10)} = 3 \cdot 10^1 + 5 \cdot 10^0 + 2 \cdot 10^{-1} + 5 \cdot 10^{-2}$$

$$B_{(2)} = 11110 + 101 + \frac{10}{1010} + \frac{101}{1010 \cdot 1010} \quad B_{(2)} = 100011{,}01$$

Переведення з вісімкової (шістнадцяткової) системи у двійкову та навпаки

*Якщо основа одної системи числення є цілим степенем двійки $P=2^K$, то при перетворенні символи вхідної інформації **замінюються відповідними двійковими еквівалентами**. При зворотньому перетворенні двійковий код розбивається на **групи по *K*-двійкових розрядів** в кожній. Ці групи **замінюються відповідними символами *P*-річної системи числення**. Для переведення вісімкового (шістнадцяткового) числа у двійкову систему числення достатньо кожну вісімкову (шістнадцяткову) цифру замінити рівною їй двійковою триадою (тетрадою).*

$A_{(8)}=765,163$ $B_{(2)} = \underbrace{111}_{7}\ \underbrace{110}_{6}\ \underbrace{101}_{5},\ \underbrace{001}_{1}\ \underbrace{110}_{6}\ \underbrace{011}_{3}$ $B_{(2)} = 111110101,001110011$

$A_{(16)}=3B7E,5A6$ $B_{(2)} = \underbrace{0011}_{3}\ \underbrace{1011}_{B}\ \underbrace{0111}_{7}\ \underbrace{1110}_{E},\ \underbrace{0101}_{5}\ \underbrace{1010}_{A}\ \underbrace{0110}_{6}$ $B_{(2)} = 11101101111110, 01011010011$

Для переведення двійкового числа у вісімкову (шістнадцяткову) систему числення достатньо розбити його направо та наліво від коми на триади (тетради) і замінити кожну триаду (тетраду) відповідною їй вісімковою (шістнадцятковою) цифрою. Якщо при розбиванні крайні триади (тетради) виявляться неповними, то їх потрібно доповнити нулями.

$A_{(2)}$=11010011101,11001011 $A_{(2)} = \underbrace{011}_{3}\ \underbrace{010}_{2}\ \underbrace{011}_{3}\ \underbrace{101}_{5},\ \underbrace{110}_{6}\ \underbrace{010}_{2}\ \underbrace{110}_{6}$ $B_{(8)} = 3235,626$

$A_{(2)}$ = 11010011101,11001011 $A_{(2)} = \underbrace{0110}_{6}\ \underbrace{1001}_{9}\ \underbrace{1101}_{D},\ \underbrace{1100}_{C}\ \underbrace{1011}_{B}$ $B_{(16)} = 69D,CB$

Простоту переходу від вісімкової (шістнадцяткової) системи числення до двійкової **можна використовувати для скорочення кількості операцій** при ручному переведенні чисел у двійкову систему числення.

Арифметичні операції в двійковій системі числення

$a_i + a_j < P$, то $a_i + a_j = a_k$, $a_i + a_j \geq P$, то $a_k = a_i + a_j - P$ і

з'являється одиниця перенесення в наступний старший розряд.

$0 + 0 = 0$ $0 + 1 = 1$ $1 + 0 = 1$ $1 + 1 = 10$ ↑ одиниця перенесення в сусідній старший розряд $0 - 0 = 0$ $1 - 0 = 1$ $1 - 1 = 0$ $10 - 1 = 1$ ↑ одиниця позички з сусіднього старшого розряду $0 \times 0 = 0$ $0 \times 1 = 0$ $1 \times 0 = 0$ $1 \times 1 = 1$	Скласти $A_{(2)}=1100101$ і $B_{(2)}=1010011$ 1 1 1 1 A = 1 1 0 0 1 0 1 + B = 1 0 1 0 0 1 1 A + B = 1 0 1 1 1 0 0 0 $A+B=10111000_{(2)}$ Відняти $B_{(2)}=1010101$ з $A_{(2)}=1101011$

	$\begin{array}{r} \overset{1}{\nearrow}\ \overset{1}{\nearrow} \\ A = 1\,1\,0\,1\,0\,1\,1 \\ - \\ B = 1\,0\,1\,0\,1\,0\,1 \\ \hline A - B = 0\,0\,1\,0\,1\,1\,0 \end{array}$ $A-B=10110_{(2)}$

При множенні двійкових чисел, як і при множенні десяткових, спочатку отримують часткові добутки, після чого знаходять їх суму. Кожний частковий добуток дорівнює нулю, якщо у відповідному розряді множника стоїть нуль, або одне множене, зсунуте на відповідне число розрядів вліво, якщо в розряді множника стоїть одиниця. Таким чином, **операція множення багаторозрядних двійкових чисел зводиться до операцій зсуву і додавання**. Комою відділяється кількість розрядів добутку, яке дорівнює сумі розрядів дробових частин співмножників.

Ділення чисел в двійковій системі числення проводиться аналогічно діленню десяткових чисел.

Операції множення та ділення виконуються в комп'ютері за спеціальними алгоритмами.

ПОДАННЯ ЧИСЕЛ В ЗАСОБАХ ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

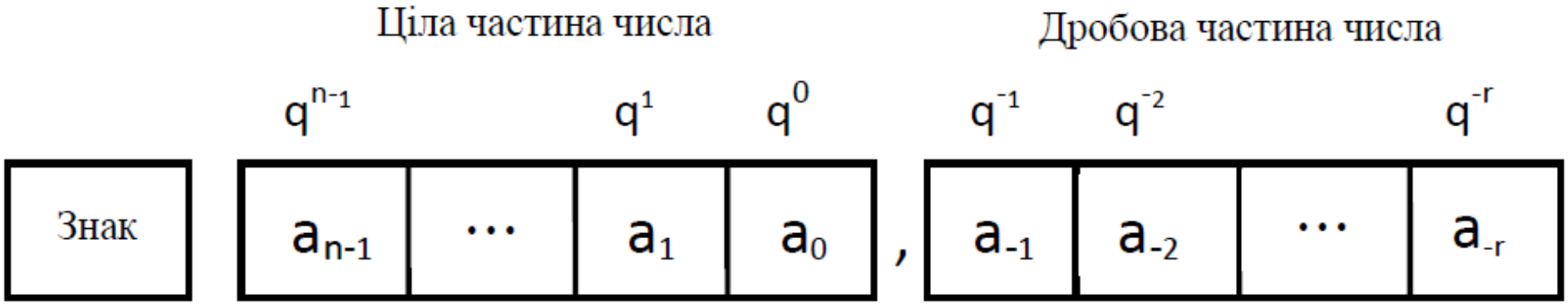
Для зберігання одного розряду коду двійкового числа використовується один фізичний елемент, який має два стійких і чітко виражених станів і володіє великою швидкістю переходу з одного стану в інший. Набір відповідної кількості таких елементів служить для подання багаторозрядного двійкового коду числа. **У цифрових ЕОМ розрядність числа завжди скінченна**.

Сукупність двійкових розрядів, призначених для зберігання і обробки чисел, являє розрядну сітку машини, що визначає формати чисел, якими можна оперувати. У **машині аппаратно не може бути представлено число, що містить більшу кількість двійкових розрядів, ніж їх є в розрядної сітці машини** (можливо лиш програмно).

Подання чисел у формі з фіксованою комою

Використовуються дві форми: **природна** (з фіксованою комою, ФК) і **нормальна** (з плаваючою комою, ПК). Подання X у формі ФК включає **знак** числа і його **модуль** в q-ічному коді. Числам з ФК відповідає запис $X=\pm a_{n-1}\ldots a_1 a_0 a_{-1} a_{-2}\ldots a_{-r}$. Розряд, в якому розміщується знак, називається **знаковим розрядом,** інші – **цифровими розрядами**. Положення коми однакове для

всіх чисел і в процесі вирішення завдань не міняється. Кома в коді числа ніяк не виділяється, а **тільки мається на увазі**.



n розрядів використовуються для запису цілої частини числа і ***r*** розрядів – для дробової частини. Якщо число є змішаним (містить цілу і дробову частини), воно обробляються як ціле, хоча і не є таким (в цьому випадку застосовують термін **масштабоване ціле**). **Обробка змішаних чисел у ОМ зустрічається украй рідко**. Як правило, використовуються ОМ з дробовою ($n = 0$) або цілочисельною ($r = 0$) арифметикою. Під час фіксації коми перед

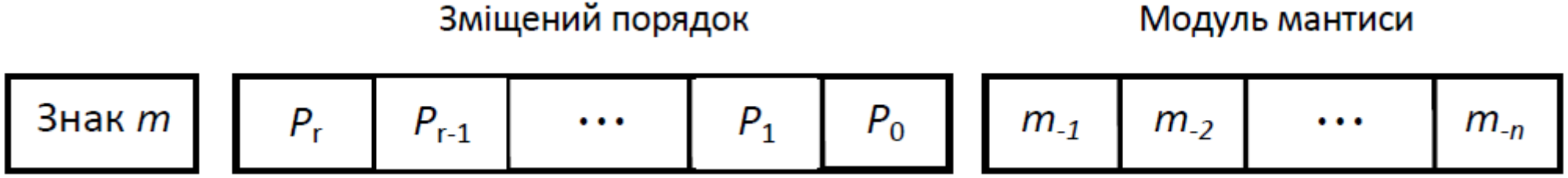
старшим цифровим розрядом можуть бути подані тільки правильні дроби. Під час фіксації коми після молодшого розряду подаються лише цілі числа. Це найбільш поширений спосіб, тому **надалі поняття ФК зв'язуватиметься виключно з цілими числами, а операції над числами у формі з ФК характеризуватимуться як цілочисельні**. Тут можливі числа із знаком і без знака.

Подання чисел у формі з плаваючою комою

Число подається у вигляді $X=\pm mq^{\pm p}$,

де m – мантиса числа X, p – порядок числа, q – основа системи числення.

У більшості обчислювальних машин для спрощення операцій над порядками останні приводять **до цілих позитивних чисел, застосовуючи так званий зміщений порядок.** Для цього до дійсного порядку додається ціле позитивне число – **зсув**.



Наприклад, у системі із зсувом 128 порядок –3 подається як 125 (–3 + 128). Звичайний зсув вибирається рівним половині уявного діапазону порядків. **Зміщений порядок займає всі біти поля порядку**, у тому числі і той, який раніше використовувався для запису знака порядку.

Мантиса в числах з ПЗ зазвичай подається в **нормалізованій формі**. Це означає, що на мантису накладаються такі умови, щоб вона по модулю була

менше одиниці ($|q| < 1$), а перша цифра після крапки відрізнялася від нуля. Отримана таким чином мантиса називається **нормалізованою**. Якщо перші i цифр мантиси рівні нулю, для нормалізації її потрібно зсунути відносно коми на i розрядів вліво з одночасним зменшенням порядку на i одиниць.

Машини з фіксованою комою мають **більшу швидкодію**, тому що не вимагають вирівнювання порядків чисел при арифметичних операціях. Вони мають **більш прості арифметико-логічні пристрої**, так як не вимагають обладнання як для операцій над мантиси, так і над порядками.

У машинах з фіксованою комою **складно проводити масштабування**.

У них **низька точність при виконанні арифметичних операцій над малими за абсолютною величиною числами**.

Подання чисел **в нормалізованому вигляді** в машинах з плаваючою комою **підвищує точність обчислень**. При однаковій довжині розрядної сітки діапазон представимих чисел в машинах з фіксованою комою значно вужче діапазону представимих чисел в машинах з плаваючою комою.

Лекція 10

- КОМП’ЮТЕРНА АРИФМЕТИКА
- СПОСОБИ КОДУВАННЯ ДВІЙКОВИХ ЧИСЕЛ
- АЛГОРИТМИ ВИКОНАННЯ АРИФМЕТИЧНИХ ОПЕРАЦІЙ У ЦІЛОЧИСЕЛЬНИХ ОПЕРАЦІЙНИХ ПРИСТРОЯХ, ЧАСТИНА І

КОМП’ЮТЕРНА АРИФМЕТИКА

КОМП’Ю́ТЕРНА АРИФМЕ́ТИКА – сукупність принципів і форм подання числової інформації, методів і алгоритмів виконання арифметичних операцій та обчислення елементарних функцій, що розглядаються на рівні внутрішньої структурної організації технічних засобів **комп’ютерів і комп’ютерних систем (КіКС)**. Є частиною **обчислювальної математики**, яка орієнтована на логічний рівень описання обчислювальних структур і процесів у них, і вивчає комп’ютерні аспекти теорії систем числення, комп’ютерно-орієнтовані алгоритми переводу чисел з однієї системи числення в інші, форми та способи комп’ютерного подання чисел, алгоритми виконання арифметичних операцій і обчислення елементарних функцій у КіКС.

СПОСОБИ КОДУВАННЯ ДВІЙКОВИХ ЧИСЕЛ

Під час виконання арифметичних операцій в арифметико-логічних пристроях необхідно враховувати можливість обробки як додатних, так і від'ємних чисел, а також виникнення переповнення розрядної сітки. Ця задача розв'язується застосуванням спеціальних кодів, за допомогою яких операція віднімання зводиться до арифметичного додавання, що призводить до спрощення арифметичних пристроїв комп'ютера. Взагалі для подання двійкових чисел в комп'ютерах застосовують **прямий, зворотний і доповняльний коди**.

В усіх цих кодах **додатні числа мають один і той же вигляд**, а **від'ємні – різний**. Двійкові коди чисел будемо записувати в квадратних дужках.

Прямий код двійкових чисел

$$[A]_п = \begin{cases} A, & \text{якщо } A \geq 0; \\ 1 + |A|, & \text{якщо } A \leq 0. \end{cases}$$

Знак додатного числа зображується цифрою **0,** а від'ємного – **1**. Знаки відділяються від мантиси крапкою.

Подати число $A_{(2)} = 0{,}101101$ і $B_{(2)} = -0{,}110111$ в прямому коді.

$$A_{(2)} = 0{,}101101 \rightarrow [A]_п = 0.101101;$$

$$B_{(2)} = -0{,}110111 \rightarrow [B]_п = 1.110111.$$

З формули видно, що нуль в прямому коді має два зображення:

$$A_{(2)} = +0{,}00\ldots0 \rightarrow [A]_п = 0.00\ldots0;$$

$$A_{(2)} = -0{,}00\ldots0 \rightarrow [A]_п = 1.00\ldots0.$$

Це треба мати на увазі при порівнюванні чисел.

Додавання чисел, які подані в прямому коді і мають однакові знаки, виконується досить просто. Мантиси чисел додаються і сумі привласнюється код знаку доданків. Операція додавання чисел з різними знаками (операція алгебраїчного додавання) є більш складною. В цьому випадку приходиться визначати більше за модулем число, з мантиси більшого числа виконати віднімання мантиси меншого числа, а потім різниці привласнити знак більшого (за модулем) числа.

Таким чином, для реалізації операції віднімання (алгебраїчного додавання) чисел в прямому коді необхідне додаткове обладнання, що конструктивно ускладнює арифметико-логічний пристрій комп'ютера. Це є недоліком прямого коду.

Зворотний код двійкових чисел

$$[A]_з = \begin{cases} A, & \text{якщо } A \geq 0; \\ 10^{n+1} - 10^{-m} + A, & \text{якщо } A \leq 0, \end{cases}$$

де n – кількість розрядів у цілій частині числа;

m – кількість розрядів дробової частини числа;

10^{-m} – одиниця молодшого розряду числа A;

10 – число 2 в двійковій системі числення.

Для чисел меньших одиниці

$$[A]_з = \begin{cases} A, & \text{якщо } A \geq 0; \\ 10 - 10^{-m} + A, & \text{якщо } A \leq 0. \end{cases}$$

Зворотний код **додатного числа збігається з його прямим кодом.**

Зворотний код **від'ємного числа утворюється заміною (інвертуванням) значущих цифр на зворотні і встановленням в знаковий розряд одиниці.**

$A_{(2)} = 0{,}1011\ [A]_з = 0.1011$, $B_{(2)} = -\ 0{,}1011\ [B]_з = 10 - 0.1011 = 1.0100$.

З формули видно, що нуль в зворотному коді має два зображення:

$$A_{(2)} = +\ 0{,}00\ldots0 \rightarrow [A]_з = 0.00\ldots0;\ A_{(2)} = -\ 0{,}00\ldots0 \rightarrow [A]_з = 1.11\ldots1.$$

Для переходу від зворотного коду до двійкового зображення числа необхідно замість знакового розряду записати мінус (якщо в знаковому розряді стоїть одиниця), а мантису числа інвертувати.

Для отримання зворотного коду **від'ємного числа з прямого коду** необхідно всі розряди мантиси про інвертувати, а в знаковому розряді залишити одиницю. Це ж правило справедливе і при переведенні від'ємних чисел із зворотного коду в прямий.

При алгебраїчному додаванні чисел, поданих в зворотному коді, виконується арифметичне додавання цих кодів, включаючи розряди знаків, які при цьому розглядаються як звичайні цифрові розряди. При виникненні одиниці, що вийшла за знаковий розряд, вона додається до молодшого розряду суми кодів. Таке перенесення називається циклічним.

Від'ємні числа $[A]_з = 10 - 10^{-m} - |A|$ і $[B]_з = 10 - 10^{-m} - |B|$

$$[A]_з + [B]_з = (10 - 10^{-m} - |A|) + (10 - 10^{-m} - |B|) = \underbrace{10}_{\text{одиниця переносу із знакового розряду}} + (10 - 10^{-m} - 10^{-m} - |A + B|).$$

$$(10 - 10^{-m} - \mathbf{10^{-m}} - |A + B|) = [A + B]_з - \mathbf{10^{-m}} = 10 - 10^{-m} - |A + B| - \mathbf{10^{-m}}$$

Щоб отримати правильний результат суми, необхідно **одиницю переносу із знакового розряду додати до молодшого розряду суми**. Тоді сума чисел A і B матиме вигляд: $[A + B]_з = 10 - 10^{-m} - |A + B|$

Додати два числа

$A_{(2)} = -0{,}011010$ і $B_{(2)} = -0{,}100011$

в зворотному коді.

$$\begin{array}{r} [A]_з = 1.100101 \\ + \\ [B]_з = 1.011100 \\ \hline 11.000001 \\ \curvearrowright 1 \\ \hline [A + B]_з = 1.000010 \end{array}$$

Додати два числа

$A_{(2)} = -0{,}110110$ і $B_{(2)} = 0{,}100101$

в зворотному коді.

Результат подати в прямому коді.

$$\begin{array}{r} [A]_з = 1.001001 \\ + \\ [B]_з = 1.100101 \\ \hline [A + B]_з = 1.101110 \end{array}$$

$$[A + B]_п = 1.010001$$

Перевагою зворотного коду є можливість звести операцію **віднімання до операції додавання**.

Недоліки зворотного коду:

− виникнення переповнення розрядної сітки в тому випадку, коли абсолютне значення суми більше одиниці, що призводить до зміни результатів обчислень;

− виникнення циклічного переносу збільшує час додавання чисел і тим самим зменшує швидкодію комп'ютерів.

Цей недолік відсутній при додаванні чисел в **доповняльному коді**.

Доповняльний код двійкових чисел

$$[A]_д = \begin{cases} A, & \text{якщо } A \geq 0; \\ 10^{n+1} + A, & \text{якщо } A < 0, \end{cases}$$

Де n – кількість розрядів у цілій частині числа;

10 – число 2 в двійковій системі числення.

Для чисел, менших одиниці,

$$[A]_д = \begin{cases} A, & \text{якщо } A \geq 0; \\ 10 + A, & \text{якщо } A < 0, \end{cases}$$

Доповняльний код **додатного числа** збігається з його **прямим кодом**.

Для від’ємного числа перетворимо вираз

$$[A]_д = 10 - |A| = 10 - 10^{-m} - |A| + 10^{-m} = [A]_з + 10^{-m}$$

Для отримання **доповняльного** коду **від’ємного числа** необхідно отримати його **зворотний код**, а потім **до молодшого розряду** числа, яке подане в зворотному коді, **додати одиницю**.

$$A_{(2)} = -0{,}11010 \quad [A]_з = 1.00101 \quad [A]_д = 1.00110$$

Якщо при виконанні арифметичних операцій в доповняльному коді виникає **одиниця**, яка **вийшла за знаковий розряд**, то вона **відкидається**.

Додати від’ємні числа A і B в доповняльному коді.

$[A]_д = 10 - |A|$, $[B]_д = 10 - |B|$

$$[A]_д + [B]_д = (10 - |A| + 10 - |B|) = \underbrace{10}_{\text{одиниця переносу відкидається}} + (10 - |A + B|).$$

Нуль в доповняльному коді має одне зображення

$$[0]_д = 0.00\ldots0,$$

$$[0]_д = [0]_з + 10^{-m} = 1.11\ldots1 + 0.00\ldots1 = \underset{\text{одиниця відкидається}}{\overset{v}{1}}0.00\ldots0 = 0.00\ldots0.$$

При алгебраїчному додаванні двійкових чисел, які подані в доповняльному коді, виконується додавання цих кодів, включаючи розряди знаків, які при цьому розглядаються як звичайні цифрові розряди. При виникненні переносу із знакового розряду одиниця переносу відкидається.

Сума отримується в прямому коді, якщо вона додатна, і в доповняльному коді, якщо вона від'ємна. Якщо сума від'ємна, то для отримання прямого коду необхідно взяти двійковий додаток від доповняльного коду.

Додати два числа $A_{(2)} = -0{,}10011$ і $B_{(2)} = -0{,}01001$ в доповняльному коді.

Результат подати в прямому коді.

$$\begin{array}{r} [A]_д = 1.01101 \\ + \\ [B]_д = 1.10111 \\ \hline [A + B]_д = 11.00100 \end{array}$$

одиниця відкидається

$$[A + B]_п = \left[[A + B]_д\right]_д = 1.11100.$$

Використання доповняльного коду дозволяє позбавитись від циклічної передачі одиниці переносу, що сприяє підвищенню швидкодії комп'ютера, але отримати доповняльні коди доданків важче, тому, що спочатку необхідно отримати зворотний код, а потім додати до нього одиницю молодшого розряду.

Модифіковані коди

В процесі виконання арифметичних операцій можливий варіант, коли модуль отриманого результату перевищує одиницю, тобто перевищує максимально допустиме число, яке може бути записане в розрядну сітку комп'ютера. Це явище, яке називається переповненням розрядної сітки, приводить до спотворення результатів.

Додати два числа $A_{(2)} = -0{,}10101$ і $B_{(2)} = -0{,}11001$ в зворотному і додатковому кодах. Модуль суми цих чисел більший 1: $|A + B| > 1$

$$
\begin{array}{rr}
+\ [A]_з = 1.01010 & +\ [A]_д = 1.01011 \\
\quad [B]_з = 1.00110 & \quad [B]_д = 1.00111 \\
\hline
\quad 10.10000 & \quad 10.10010 \\
\quad \gamma \longrightarrow 1 & \quad \text{v відкидається} \\
\hline
[A + B]_з = 0.10001 & [A + B]_д = 0.10010
\end{array}
$$

Отриманий результат є невірним з двох позицій.

По-перше, в знаковому розряді стоїть нуль, який показує, що отримана сума додатна, а насправді результат повинен бути від'ємним.

По-друге, в цифрових розрядах числа також отримане невірне значення.

Це виходить через те, що результат додавання не вкладається в розрядну сітку комп'ютера, тобто спостерігається переповнення розрядної сітки. Щоб результат обчислень вийшов правильним, необхідно виключити переповнення розрядної сітки.

Для цього використовують модифіковані коди.

Модифіковані коди відрізняються від розглянутих вище тим, що для зображення знака числа відводиться **два розряди**. Якщо число додатне, то в знаковому розряді записується **два нулі**, якщо від'ємне – **дві одиниці**

Число $A_{(2)} = -0{,}101011$ в **модифікованих** кодах

$$[A]^{м}_{п} = 11.101011; \quad [A]^{м}_{з} = 11.010100; \quad [A]^{м}_{д} = 11.010101$$

Алгебраїчне додавання двійкових чисел в модифікованих кодах виконується за такими ж правилами, як і в звичайних кодах, при цьому знакові розряди розглядаються **як розряди цілої частини числа.**

Виконати додавання чисел $A_{(2)} = -0{,}00101$ і $B_{(2)} = -0{,}10011$

$$[A]^{м}_{з} = 11.11010 \qquad [A]^{м}_{д} = 11.11011$$
$$[B]^{м}_{з} = 11.01100 \qquad [B]^{м}_{д} = 11.01101$$
$$111.00110 \qquad 111.01000$$
$$\downarrow \qquad \downarrow$$
$$\rightarrow 1 \qquad \text{відкидається}$$
$$[A+B]^{м}_{з} = 11.00111 \qquad [A+B]^{м}_{д} = 11.01000$$
$$[A+B]_{п} = 1.11000 \qquad [A+B]_{п} = 1.11000$$

Ознакою переповнення розрядної сітки є наявність в знакових розрядах різних цифр: 01 або 10. Якщо комбінація 01, то результат додатний, якщо комбінація 10, то результат від'ємний.

У випадку переповнення розрядної сітки в комп'ютерах з **фіксованою комою** він зупиняється. Для усунення переповнення розрядної сітки необхідно знову виконати масштабування.

Якщо здійснюється переповнення розрядної сітки в комп'ютері з **плаваючою комою**, то в цьому випадку комп'ютер здійснює так звану операцію **нормалізації вправо на один розряд**. В результаті виконання цієї операції зсувається вправо на один розряд код результату алгебраїчного додавання і додається одиниця до порядку числа. При цьому **модуль числа залишається без змін**.

Додати числа $A_{(2)} = -0{,}110110 \cdot 10^{011}$ і $B_{(2)} = -0{,}001101 \cdot 10^{011}$ в формі з плаваючою комою з використанням доповняльного модифікованого коду:

$$[A]_{д}^{м} = 11.001010 \quad 0.011$$
$$[B]_{д}^{м} = 11.110011 \quad 0.011$$
$$\overline{\qquad\qquad\qquad\qquad}$$
$$\text{✗}10.111101 \quad 0.011$$
$$\overline{\qquad\qquad\qquad\qquad}$$
$$[A + B]_{д}^{м} = 10.111101 \quad 0.011$$

Відбулося переповнення розрядної сітки. Виконуємо зсув мантиси і її знаку на один розряд вправо і збільшення порядку на одиницю

$$[A + B]_{д}^{м} = 11.0111101 \quad 0.100$$

Модифіковані коди застосовуються для скорочення часу на визначення переповнення розрядної сітки.

АЛГОРИТМИ ВИКОНАННЯ АРИФМЕТИЧНИХ ОПЕРАЦІЙ У ЦІЛОЧИСЕЛЬНИХ ОПЕРАЦІЙНИХ ПРИСТРОЯХ

Для більшості сучасних комп'ютерів у форматі з фіксованою комою (ФК) кома фіксується праворуч від молодшого розряду коду числа. Відповідні операційні пристрої називають цілочисельними ОПр. У формі з ФК можуть бути представлені як числа без знака, коли все *n* позицій числа відводяться під значущі цифри, так і зі знаком. У останньому випадку старший (n – 1)-й розряд числа займає знак числа (0 – плюс, 1 – мінус), а під значущі цифри відведені розряди з (*n* – 2)-го по 0-й. Під час запису негативних чисел використовується доповняльний код. Цілочисельний ОПр повинний забезпечувати виконання таких арифметичних операцій над числами без знака і зі знаком: **додавання/віднімання; множення; ділення**.

Алгоритми виконання операцій додавання і віднімання

Під час додавання *n*-розрядных двійкових чисел (біт знака і *n* – 1 значу-щих цифр) можливий результат, який містить *n* значущих цифр.

Ця ситуація відома як **переповнення**. «Зайвий» біт займає позицію знака, що приводить до некоректності результату. Природно, що ОПр повинний виявляти факт переповнення і сигналізувати про нього. Для цього використовується наступне правило: **якщо додаються два числа і вони обидва позитивні або обидва негативні, переповнення має місце тоді і тільки тоді, коли знак результату протилежний знаку доданків.**

Приклади додавання цілих чисел, поданих у доповняльному коді (нагадаємо, що під час додавання в доповняльному коді знаковий розряд бере участь в операції нарівні з цифровими).

$$\begin{array}{r} (+4)\ 0\,1\,0\,0 \\ +\ (+3)\ 0\,0\,1\,1 \\ \hline (+7)\ 0\,1\,1\,1 \end{array}$$

а

$$\begin{array}{r} (-7)\ 1\,0\,0\,1 \\ +\ (+4)\ 0\,1\,0\,0 \\ \hline (-3)\ 1\,1\,0\,1 \end{array}$$

б

$$\begin{array}{r} (+5)\ 0\,1\,0\,1 \\ +\ (-2)\ 1\,1\,1\,0 \\ \hline (+3)\ \not{1}\ 0\,0\,1\,1 \end{array}$$

в

$$\begin{array}{r} (-1)\ 1\,1\,1\,1 \\ +\ (-5)\ 1\,0\,1\,1 \\ \hline (-6)\ \not{1}\ 1\,0\,1\,0 \end{array}$$

г

$$\begin{array}{r} (+5)\ 0\,1\,0\,1 \\ +\ (+4)\ 0\,1\,0\,0 \\ \hline 1\,0\,0\,1 \end{array}$$

д

$$\begin{array}{r} (-6)\ 1\,0\,1\,0 \\ +\ (-5)\ 1\,0\,1\,1 \\ \hline \not{1}\ 0\,1\,0\,1 \end{array}$$

е

Приклади виконання операції додавання в доповняльному коді:

а, б, в, г – додавання без виникнення переповнення;

д, е – додавання з переповненням

Для віднімання одного числа (від'ємника) з іншого (зменшуваного) необхідно взяти доповнення від'ємника і додати його до зменшуваного.

Доповненням – від'ємник з протилежним знаком, поданий у доповняльному коді.

```
(+3)  0 0 1 1
(+7)  0 1 1 1
-------------
      0 0 1 1
    + 1 0 0 1
-------------
(-4)  1 1 0 0
```

а

```
(+5)  0 1 0 1
(+2)  0 0 1 0
-------------
      0 1 0 1
    + 1 1 1 0
-------------
(+3)⊥ 0 0 1 1
```

б

```
(-5)  1 0 1 1
(+2)  0 0 1 0
-------------
      1 0 1 1
    + 1 1 1 0
-------------
(-7)⊥ 1 0 0 1
```

в

```
(+6)  0 1 1 0
(- 1) 1 1 1 1
-------------
      0 1 1 0
    + 0 0 0 1
-------------
(-7)  0 1 1 1
```

г

```
(+7)  0 1 1 1
(-7)  1 0 0 1
-------------
      0 1 1 1
    + 0 1 1 1
-------------
      1 1 1 0
```

д

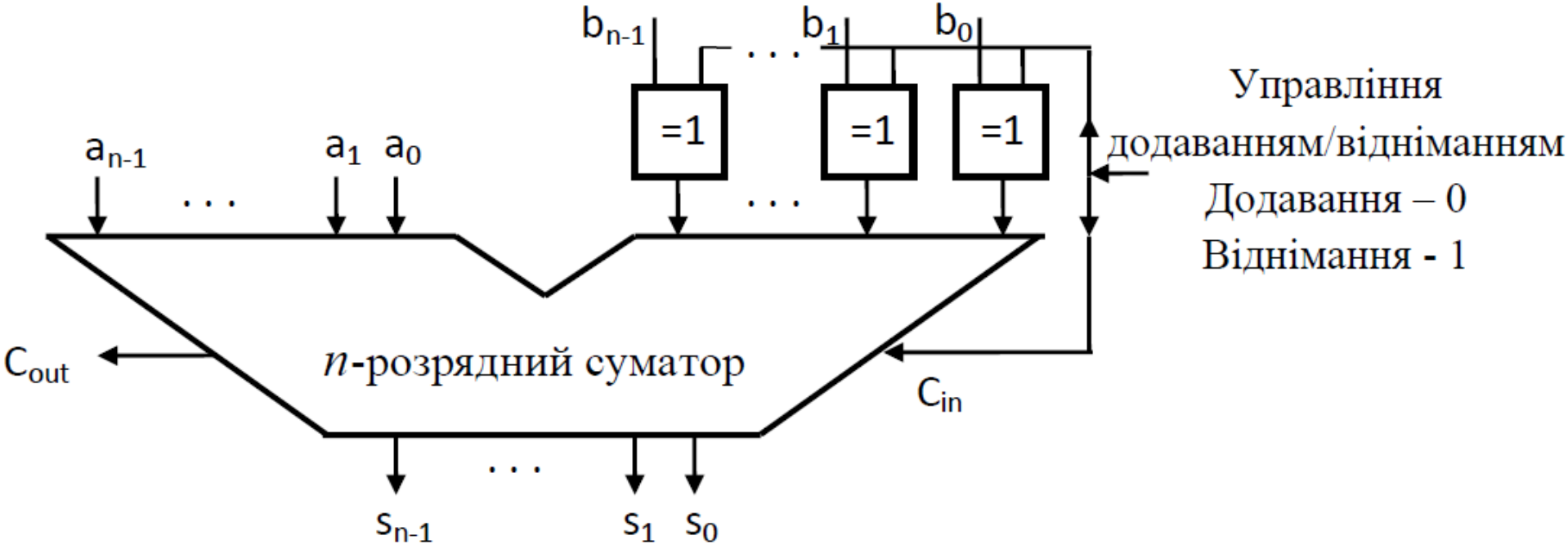
```
(-6)  1 0 1 0
(+4)  0 1 0 0
-------------
      1 0 1 0
    + 1 1 0 0
-------------
     ⊥0 1 1 0
```

е

Приклади виконання операції віднімання в доповняльному коді:
а, *б, в, г* - віднімання без виникнення переповнення;
д, е - віднімання з переповненням

Для виявлення переповнення застосовується **модифікований доповняльний код**: для зберігання знака відводяться два розряди, обидва беруть участь в арифметичній операції нарівні з цифровими розрядами.

Структура операційного блока (ОБ) для додавання і віднімання чисел із знаком у форматі з фіксованою комою



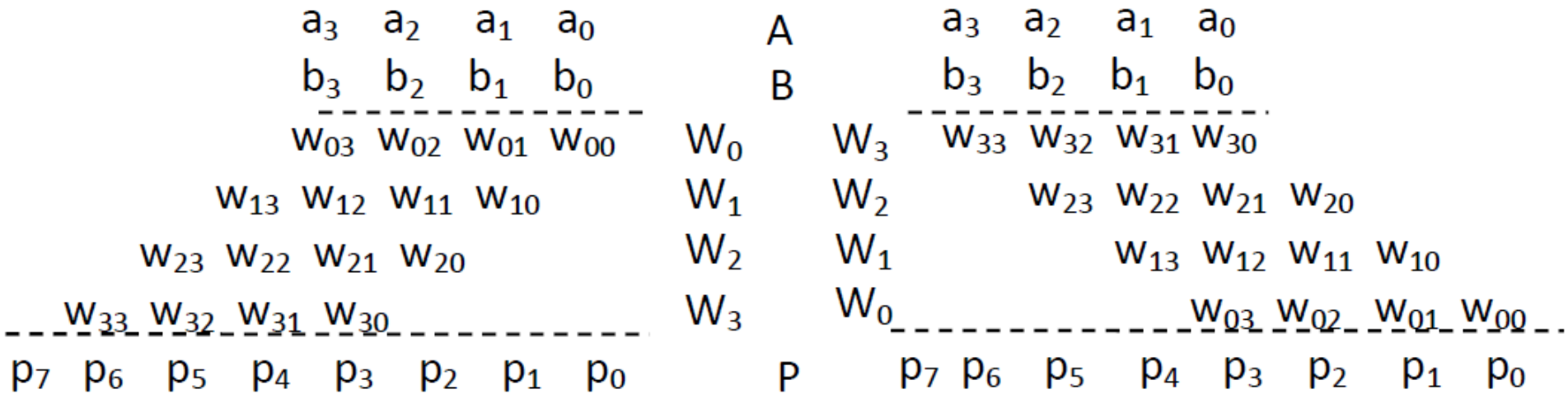
Центральною ланкою пристрою є *n*-разрядный двійковий суматор.

Операнд *A* поступає на вхід суматора без змін. Операнд *B* заздалегідь пропускається через схеми додавання по модулю 2, тому вид коду *B,* що поступає на інший вхід суматора, залежить від виконуваної операції. Якщо задана операція додавання (управляючий код – 0), то результат на виході ОБ визначається виразом S = *A* + *B*. Якщо задана операція віднімання (управляючий код – 1), на вхід суматора подаються інверсні значення всіх розрядів *B* і, крім того, на вхід перенесення в молодший розряд суматора C_{in} поступає 1. В результаті на виході ОБ буде $S = A + \overline{B} + 1$, що відповідає додаванню до *A* числа *B* з протилежним знаком, тобто відніманню.

Алгоритми множення кодів чисел з фіксованою комою

У порівнянні з додаванням і відніманням, множення – складніша операція як у разі програмного, так і в разі апаратного втілення. Застосовуються різні алгоритми її і, відповідно, декілька схем побудови операційних блоків, що забезпечують виконання операції. Традиційна схема множення схожа на відому процедуру запису «в стовпчик». Обчислення добутку $P(p_{2n-1}\, p_{2n-2} \ldots p_1\, p_0)$ двох ***n***-розрядних двійкових чисел без знака $A(a_{n-1}\, a_{n-2} \ldots a_1\, a_0)$ і $B(b_{n-1}\, b_{n-2} \ldots b_1\, b_0)$ зводиться до формування часткових добутків (ЧД) W_i, поодинці на кожну цифру множника, з подальшим підсумовуванням отриманих ЧД. Перед підсумовуванням кожен частковий добуток повинен бути зсунутим на один розряд щодо попереднього згідно з вагою цифри множника, якій цей ЧД відповідає.

Оскільки операндами є двійкові числа, обчислення ЧД спрощується – якщо цифра множника b_i дорівнює 0, то W_i теж дорівнює 0, а при b_i=1 частковий добуток дорівнює множеному (W_i=A). Результату множення двох ***n***-розрядних двійкових чисел $P=A\times B$ містить $2\boldsymbol{n}$ бітів.



$$W_0 = Ab_0 \times 2^0$$
$$W_1 = Ab_1 \times 2^1$$
$$W_2 = Ab_2 \times 2^2$$
$$W_3 = Ab_3 \times 2^3$$

$$W_3 = Ab_3 \times 2^0$$
$$W_2 = Ab_2 \times 2^{-1}$$
$$W_1 = Ab_1 \times 2^{-2}$$
$$W_0 = Ab_0 \times 2^{-3}$$

Підсумовування ЧД проводиться не на завершальному етапі, а в міру їх отримання, щоб уникнути зберігання всіх ЧД. Пристрій множення припускає наявність регістрів множеного, множника і суми часткових добутків, а також суматора ЧД і схем зсуву. Можливі чотири варіанти:

1. Множення починається з молодших розрядів множника, і зсув суми часткових добутків відбувається вправо у разі нерухомого множеного.

2. Множення починається із старших розрядів множника, і зсув суми часткових добутків відбувається вліво у разі нерухомого множеного.

3. Множення починається з молодших розрядів множника, і зсув множеного відбувається вліво у разі нерухомої суми часткових добутків.

4. Множення починається із старших розрядів множника, і зсув множеного відбувається вправо у разі нерухомої суми часткових добутків.

Множення чисел без знака

A	×	1 0 1 0
B		1 0 1 1
0		0 0 0 0
$W_0 = Ab_0$	+	1 0 1 0
$P_0 = 0 + W_0$		0 1 0 1 0
$P_0 \times 2^{-1}$	⇨	0 1 0 1 0
$W_1 = Ab_1$	+	1 0 1 0
$P_1 = P_0 \times 2^{-1} + W_1$		0 1 1 1 1 0
$P_1 \times 2^{-1}$	⇨	0 1 1 1 1 0
$W_2 = Ab_2$	+	0 0 0 0
$P_2 = P_1 \times 2^{-1} + W_2$		0 0 1 1 1 1 0
$P_2 \times 2^{-1}$	⇨	0 0 1 1 1 1 0
$W_3 = Ab_3$	+	1 0 1 0
$P_3 = P_2 \times 2^{-1} + W_3$		0 1 1 0 1 1 1 0
$P_3 \times 2^{-1}$	⇨	0 1 1 0 1 1 1 0

Множення із зсувом суми часткових добутків вправо

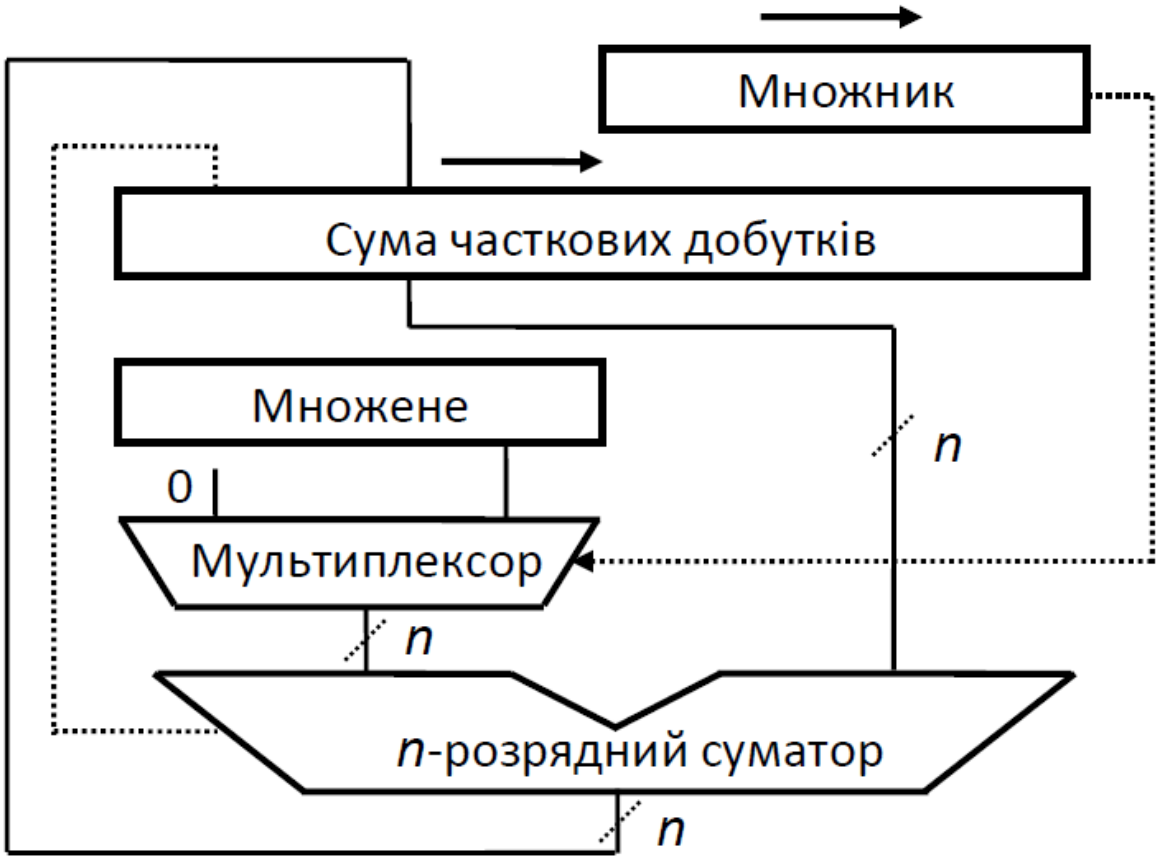


Схема пристрою множення за алгоритмом *A*

Спочатку множене і множник заносяться в ***n***-розрядні регістри множеного (РМн) і множника (РМк) відповідно, а всі розряди 2***n***-розрядного регістра суми часткових добутків (РЧД) встановлюються в 0. Множення відбувається

за ***n*** кроків. На кожному кроці, залежно від стану молодшого розряду регістра множника, який керує мультиплексором, на один з входів ***n***-разрядного суматора подається або множене, або 0. На другий вхід поступає вміст ***n*** старших розрядів РЧД. Новий частковий добуток з суматора пересилається в старші розряди РЧД. Далі вміст РЧД зсовується на один розряд вправо, причому в старший розряд регістра, що звільнився, заноситься значення переносу із старшого розряду суматора. Оскільки мультиплексор управляється молодшим розрядом РМк, то і вміст цього регістра також зсовується на один розряд вправо. Описана послідовність повторюється ***n*** разів.

Множення чисел із знаком

n-розрядні співмножники містять знак у старшому розряді слова і *n*–1 значущу цифру (знаковий розряд відокремлюється крапкою і бере участь в операції разом з цифровими розрядами). Найбільш очевидна думка – набути абсолютних значень операндів і перемножити їх як числа без знака. У ОМ числа із знаком подають у формі з фіксованою комою в **доповняльному коді**. **Додатні числа** в цьому уявленні **не відрізняються** від запису **в прямому коді**, **від'ємні** записуються у вигляді $2^n–x$, де x – фактичне значення числа. В двійковій системі **запис від'ємного числа** в доповняльному коді зводиться до **інвертування всіх цифрових розрядів** числа, представленого в прямому коді, і **додавання одиниці до молодшого розряду зворотного коду**, що вийшов після інвертування.

A	×	0. 1 1 0 1	+13
B		0. 1 0 1 0	+10
0		0. 0 0 0 0	
$W_0 = Ab_0$	+	0. 0 0 0 0	
$P_0 = 0 + W_0$		0. 0 0 0 0	
$P_0 \times 2^{-1}$	⇨	0. 0 0 0 0 0	
$W_1 = Ab_1$	+	0. 1 1 0 1	
$P_1 = P_0 \times 2^{-1} + W_1$		0. 1 1 0 1 0	
$P_1 \times 2^{-1}$	⇨	0. 0 1 1 0 1 0	
$W_2 = Ab_2$	+	0. 0 0 0 0	
$P_2 = P_1 \times 2^{-1} + W_2$		0. 0 1 1 0 1 0	
$P_2 \times 2^{-1}$	⇨	0. 0 0 1 1 0 1 0	
$W_3 = Ab_3$	+	0. 1 1 0 1	
$P_3 = P_2 \times 2^{-1} + W_3$		1. 0 0 0 0 0 1 0	
$P_3 \times 2^{-1}$	⇨	0. 1 0 0 0 0 0 1 0	+130

Множене довільного знаку, множник додатний

A	×	1.0011	−13
B		0.1010	+10
0		0.0000	
$W_0 = Ab_0$	+	0.0000	
$P_0 = 0 + W_0$		0.0000	
$P_0 \times 2^{-1}$	⇒	0.00000	
$W_1 = Ab_1$	+	1.0011	
$P_1 = P_0 \times 2^{-1} + W_1$		1.00110	
$P_1 \times 2^{-1}$	⇒	1.10011	
$W_2 = Ab_2$	+	0.0000	
$P_2 = P_1 \times 2^{-1} + W_2$		1.10011	
$P_2 \times 2^{-1}$	⇒	1.11001	
$W_3 = Ab_3$	+	1.0011	
$P_3 = P_2 \times 2^{-1} + W_3$		10.1111110	
$P_3 \times 2^{-1}$	⇒	1.0111111 0	−130

Під час зсуву вправо для суми часткових добутків цифрові позиції, що звільнилися у разі зсуву, повинні заповнюватися **не нулем, а значенням знакового розряду зсунутого числа**. Це правило починає діяти з моменту, коли **з'являється перша одиниця**.

Множене довільного знака, множник від'ємний

A	0. 1 1 0 1	+13
B	× 1. 0 1 1 0	−10
0	0. 0 0 0 0	
$W_0 = Ab_0$	+ 0. 0 0 0 0	
$P_0 = 0 + W_0$	0. 0 0 0 0	
$P_0 \times 2^{-1}$	⇨ 0. 0 0 0 0 0	
$W_1 = Ab_1$	+ 0. 1 1 0 1	
$P_1 = P_0 \times 2^{-1} + W_1$	0. 1 1 0 1 0	
$P_1 \times 2^{-1}$	⇨ 0. 0 1 1 0 1 0	
$W_2 = Ab_2$	+ 0. 1 1 0 1	
$P_2 = P_1 \times 2^{-1} + W_2$	1. 0 0 1 1 1 0	
$P_2 \times 2^{-1}$	⇨ 0. 1 0 0 1 1 1 0	
$W_3 = Ab_3$	+ 0. 0 0 0 0	
$P_3 = P_2 \times 2^{-1} + W_3$	0. 1 0 0 1 1 1 0	
$P_3 \times 2^{-1}$	⇨ 0. 0 1 0 0 1 1 1 0	
	+ 1. 0 0 1 1	Корекція
	1. 0 1 1 1 1 1 1 0	−130

A	1. 0 0 1 1	−13
B	× 1. 0 1 1 0	−10
0	0. 0 0 0 0	
$W_0 = Ab_0$	+ 0. 0 0 0 0	
$P_0 = 0 + W_0$	0. 0 0 0 0	
$P_0 \times 2^{-1}$	⇨ 0. 0 0 0 0 0	
$W_1 = Ab_1$	+ 1. 0 0 1 1	
$P_1 = P_0 \times 2^{-1} + W_1$	1. 0 0 1 1 0	
$P_1 \times 2^{-1}$	⇨ 1. 1 0 0 1 1 0	
$W_2 = Ab_2$	+ 1. 0 0 1 1	
$P_2 = P_1 \times 2^{-1} + W_2$	0. 1 1 0 0 1 0	
$P_2 \times 2^{-1}$	⇨ 1. 0 1 1 0 0 1 0	
$W_3 = Ab_3$	+ 0. 0 0 0 0	
$P_3 = P_2 \times 2^{-1} + W_3$	1. 0 1 1 0 0 1 0	
$P_3 \times 2^{-1}$	⇨ 1. 1 0 1 1 0 0 1 0	
	+ 0. 1 1 0 1	Корекція
	0. 1 0 0 0 0 0 1 0	+130

Множник від’ємний –записується в доповняльному коді:

$$[B]_{Д} = 2^{n} - |B|,$$

і в цифрових розрядах коду буде подане число

$$2^{n-1} - |B|.$$

У разі типового множення (як у випадку $B \geq 0$) отримаємо

$$P = A \times (2^{n-1} - |B|) = -|B| \times A + A \times 2^{n-1}.$$

Псевдодобуток P більше дійсного добутка P на величину

$$A \times 2^{n-1},$$

що і необхідно враховувати під час формування остаточного результату. Для цього перед останнім зсувом з отриманого псевдодобутку необхідно відняти надлишковий член. У приклади множення є згадана корекція результату – додавання протилежного А числа у допівняльному коді.

Лекція 11

АЛГОРИТМИ ВИКОНАННЯ АРИФМЕТИЧНИХ ОПЕРАЦІЙ У ЦІЛОЧИСЕЛЬНИХ ОПЕРАЦІЙНИХ ПРИСТРОЯХ, ЧАСТИНА ІІ

Прискорення цілочисельного множення

На практиці для перемножування застосовують алгоритм Бута, що прискорює процес множення. В його основі лежить таке співвідношення

$$2^m + 2^{m-1} + \cdots + 2^k = 2^{m+1} - 2^k ,$$

де m і k – номери крайніх розрядів у групі з послідовних одиниць.

Наприклад, $011110 = 2^5 - 2^1$. Це означає, що за наявності в множнику груп з декількох одиниць (комбінацій вигляду 011, 110), послідовне додавання до СЧД множеного з наростаючою вагою (від 2^k до 2^m) можна замінити відніманням з СЧД множеного з вагою 2^k і додаванням до СЧД множеного з вагою 2^{m+1}. Алгоритм припускає три операції: зсув, додавання і віднімання. Крім скорочення числа додавань (віднімань) у нього є ще одна перевага – він у рівній мірі застосовний до чисел без знаку і зі знаком.

Методи прискорення множення можна умовно розділити на **апаратні** і **логічні**. Ті та інші вимагають **додаткових витрат обладнання**, які під час використання апаратних методів зростають із збільшенням розрядності співмножників.

Апаратні способи приводять **до ускладнення схеми помножувача**, але не зачіпають схеми управління.

Додаткові витрати обладнання під час реалізації логічних методів не залежать від розрядності операндів, але **схема управління помножувача** при цьому **ускладняється**.

На практиці прискорення множення часто досягається комбінацією апаратних і логічних методів.

Логічні методи прискорення множення можна поділити на дві групи:

- методи, що дозволяють зменшити кількість додавань у ході множення;
- методи, що забезпечують обробку декількох розрядів множника за крок.

Реалізація вимагає введення додаткових ланцюгів зсуву в регістри.

Апаратні методи прискорення множення.

Один із способів полягає в зміні системи кодування співмножників, за рахунок чого можна скоротити кількість підсумовуваних часткових добутків. Приклад: алгоритм Бута. Апаратні методи зводяться:

- до паралельного обчислення часткових добутків;
- до скорочення кількості операцій додавання;
- до зменшення часу розповсюдження перенесень під час підсумовування часткових добутків.

Всі три підходи реалізуються за допомогою комбінаційних пристроїв.

Паралельне обчислення ЧД має багато різновидів. Відмінності виявляються в основному в способі підсумовування отриманих часткових добутків, і з цих позицій використовувані схеми множення можна поділити на **матричні** і з **деревовидною структурою**. В обох варіантах підсумовування здійснюється за допомогою масиву взаємозв'язаних однорозрядних суматорів. У матричних помножувачах суматори організовані у вигляді матриці, а в деревовидних вони реалізуються у вигляді дерева. Відмінності в рамках кожної з цих груп виражаються в кількості використовуваних суматорів, їх вигляді і способі розповсюдження переносів, що виникають у процесі додавання.

Алгоритми ділення кодів чисел з фіксованою комою

Ділення складніша операція, ніж множення, але базується на тих же принципах. Основу складає загальноприйнятий спосіб ділення за допомогою операцій віднімання або додавання і зсуву

Z ділене	$z_7\ z_6\ z_5\ z_4\ z_3\ z_2\ z_1\ z_0$	$d_3\ d_2\ d_1\ d_0$	D – дільник
$-\ D\ q_3 \times 2^3$	$r_3\ r_2\ r_1\ r_0$	$q_3\ q_2\ q_1\ q_0$	Q – частка
$-\ D\ q_2 \times 2^2$	$r_3\ r_2\ r_1\ r_0$		
$-\ D\ q_1 \times 2^1$	$r_3\ r_2\ r_1\ r_0$		
$-\ D\ q_0 \times 2^0$	$r_3\ r_2\ r_1\ r_0$		
	$s_3\ s_2\ s_1\ s_0$		S – остача

Задача зводиться до обчислення частки Q і залишку S:

$$Q = \text{int}\left(\frac{Z}{D}\right),\ S = Z - QD,\ S < D$$

Ділення – послідовність віднімань дільника спочатку з діленого, а потім з часткових залишків (ЧЗ), що утворюються в процесі ділення.

Ділене $Z(z_{2n-1}\ z_{2n-2}\ \ldots\ z_1\ z_0)$ записується подвійним словом ($2n$ розрядів), дільник $D(d_{2n-1}\, d_{2n-2}\ \ldots\ d_1\, d_0)$, частка $Q(q_{2n-1}\, q_{2n-2}\ \ldots\ q_1\ q_0)$ і залишок $S(s_{2n-1}\, s_{2n-2}\ \ldots\ s_1\ s_0)$ мають розрядність n. Операція виконується за n ітерацій і може бути описана таким чином:

$$S^{(i)} = 2S^{(i-1)} - q_{n-i}(2^n D),\ \text{якщо}\ S^{(0)} = Z\ \ \text{і}\ \ S^{(n)} = 2^n S;$$

$$q_{n-i} = \begin{cases} 1, \text{якщо } (2S^{(i-1)} - 2^n D) \geq 0, \\ 0, \text{якщо } (2S^{(i-1)} - 2^n D) < 0. \end{cases}$$

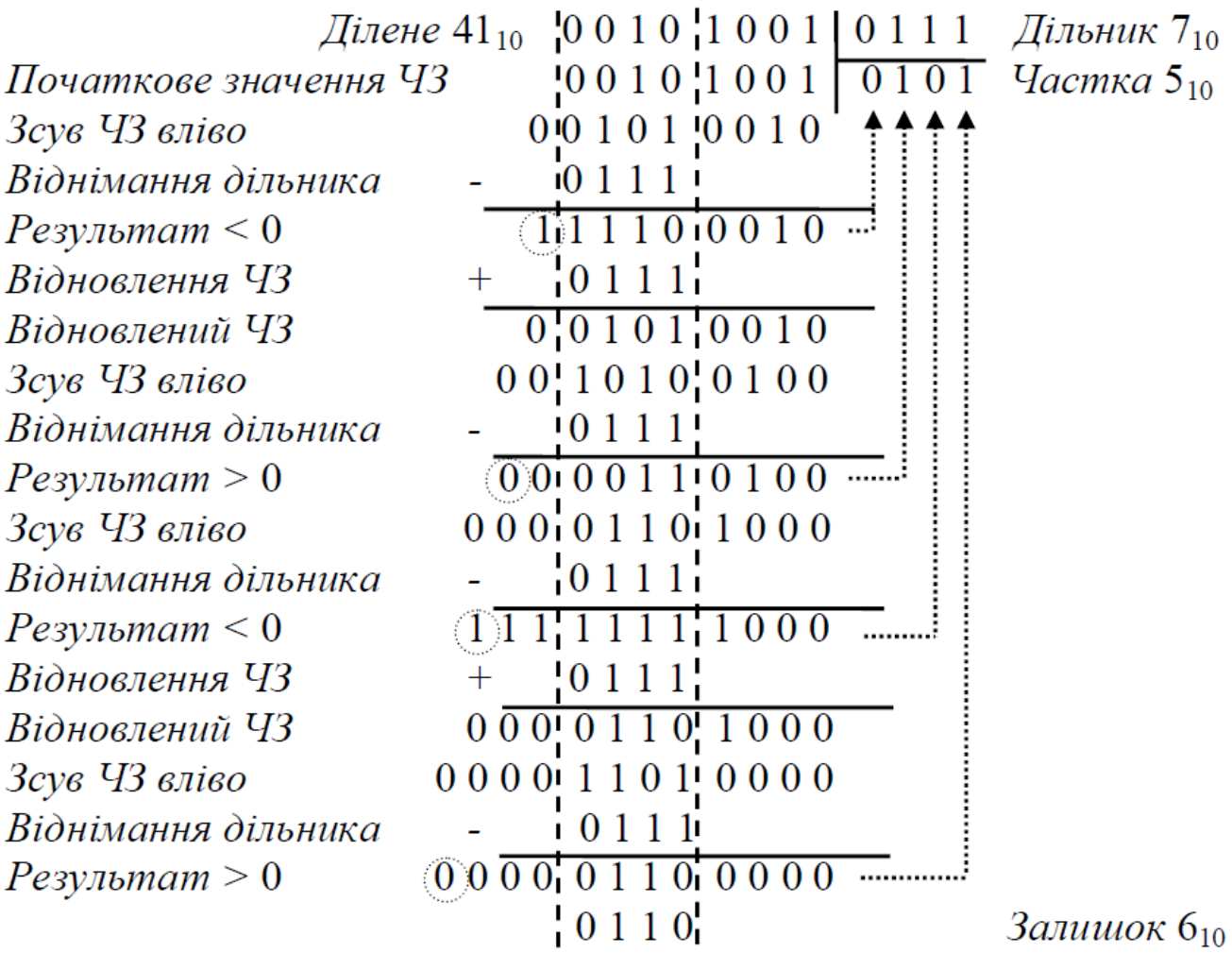
Після n ітерацій виходить $S^{(n)} = 2^n S^{(0)} - Q(2^n D) = 2^n[Z - (Q \times D)] = 2^n S.$

Частка від ділення $2n$-розрядного числа на n-розрядне може містити більше, ніж n розрядів. У цьому випадку виникає переповнення, через що перед виконанням ділення необхідна перевірка умови $Z < (2^n - 1)D + D = 2^n D$. переповнення не буде, якщо число, що міститься в старших n розрядах діленого, менше дільника. Перед початком операції необхідно виключити можливість ситуації ділення на 0. Реалізується двома способами:

- з нерухомим діленим і зсовуваним вправо дільником;
- з нерухомим дільником і зсовуваним вліво діленим.

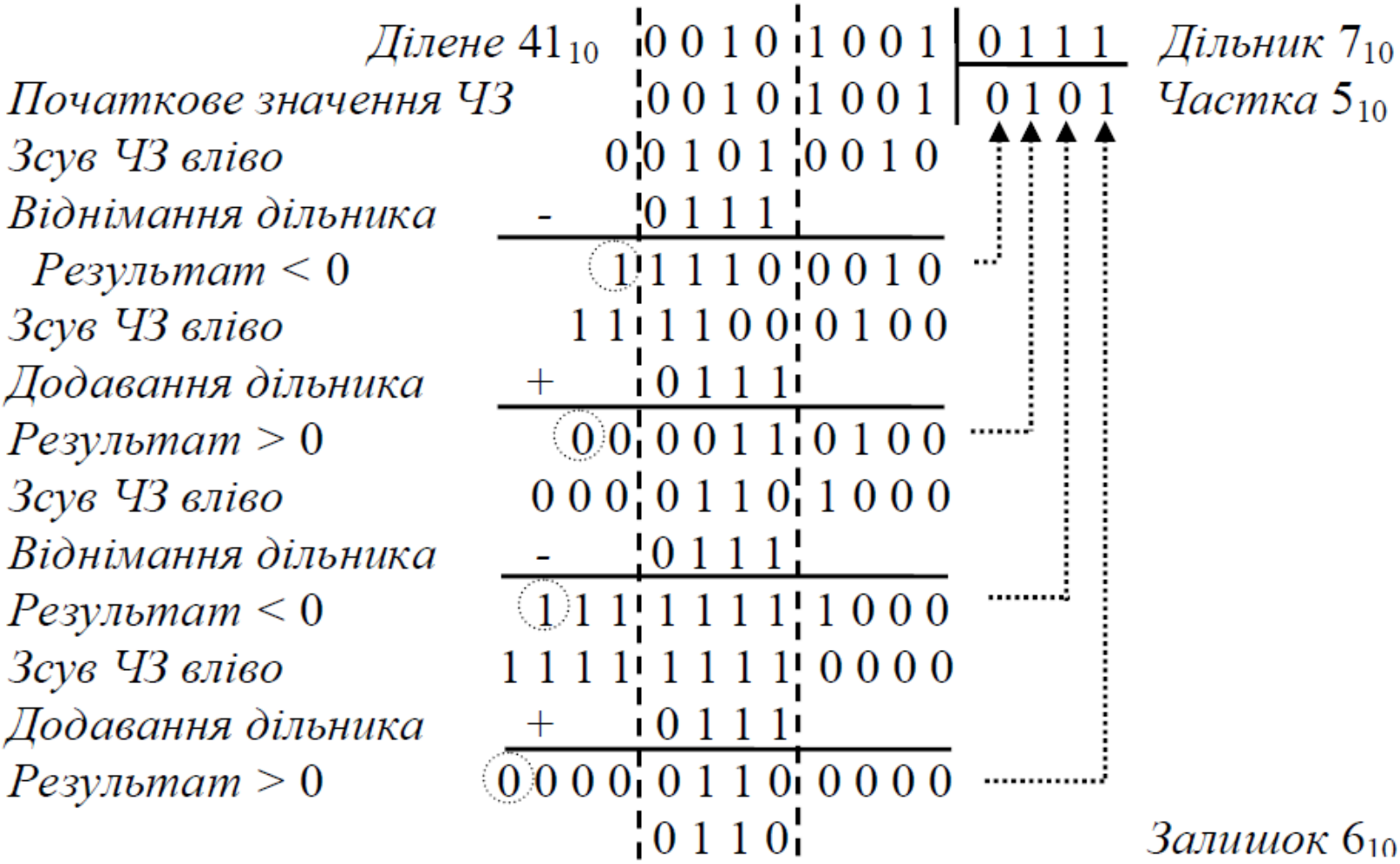
Недоліком першого способу є потреба мати в пристрої ділення суматор і регістр подвійної довжини. Другий спосіб дозволяє будувати пристрій ділення з суматором одинарної довжини.

Алгоритм ділення з нерухомим дільником і відновленням залишку. Він дуже схожий на ділення стовпчиком. Недолік: виконання на окремих



кроках додаткових операцій додавання для відновлення часткового залишку. Це збільшує час виконання ділення, яке в цьому випадку може мінятися залежно від конкретного поєднання кодів операндів.

Алгоритм ділення з нерухомим дільником без відновлення залишку



Опис алгоритму.

1. Початкове значення ЧЗ вважається рівним старшим розрядам діленого.

2. ЧЗ подвоюється шляхом зсуву на один розряд вліво. При цьому в молодший розряд ЧЗ, що звільняється під час зсуву, заноситься чергова цифра частки.

3. Із зсунутого ЧЗ віднімається дільник, якщо залишок додатний, і до зсунутого часткового залишку додається дільник, якщо залишок від'ємний.

4. Чергова цифра модуля частки дорівнює одиниці, коли результат віднімання додатний, і нулю, якщо він від’ємний.

5. Пункти 2…4 послідовно виконуються для отримання всіх цифр модуля частки.

Ділення чисел із знаком

Ділення чисел із знаком може бути виконане шляхом переходу до абсолютних значень діленого і дільника, з подальшим привласненням частки знака «+» у разі збігу знаків діленого і дільника або «–» – інакше. Ділення чисел, представлених у доповняльному коді, можна здійснювати не переходячи до модулів. Дії з частковим залишком (додавання або віднімання *D*) залежать від знаків залишку і дільника

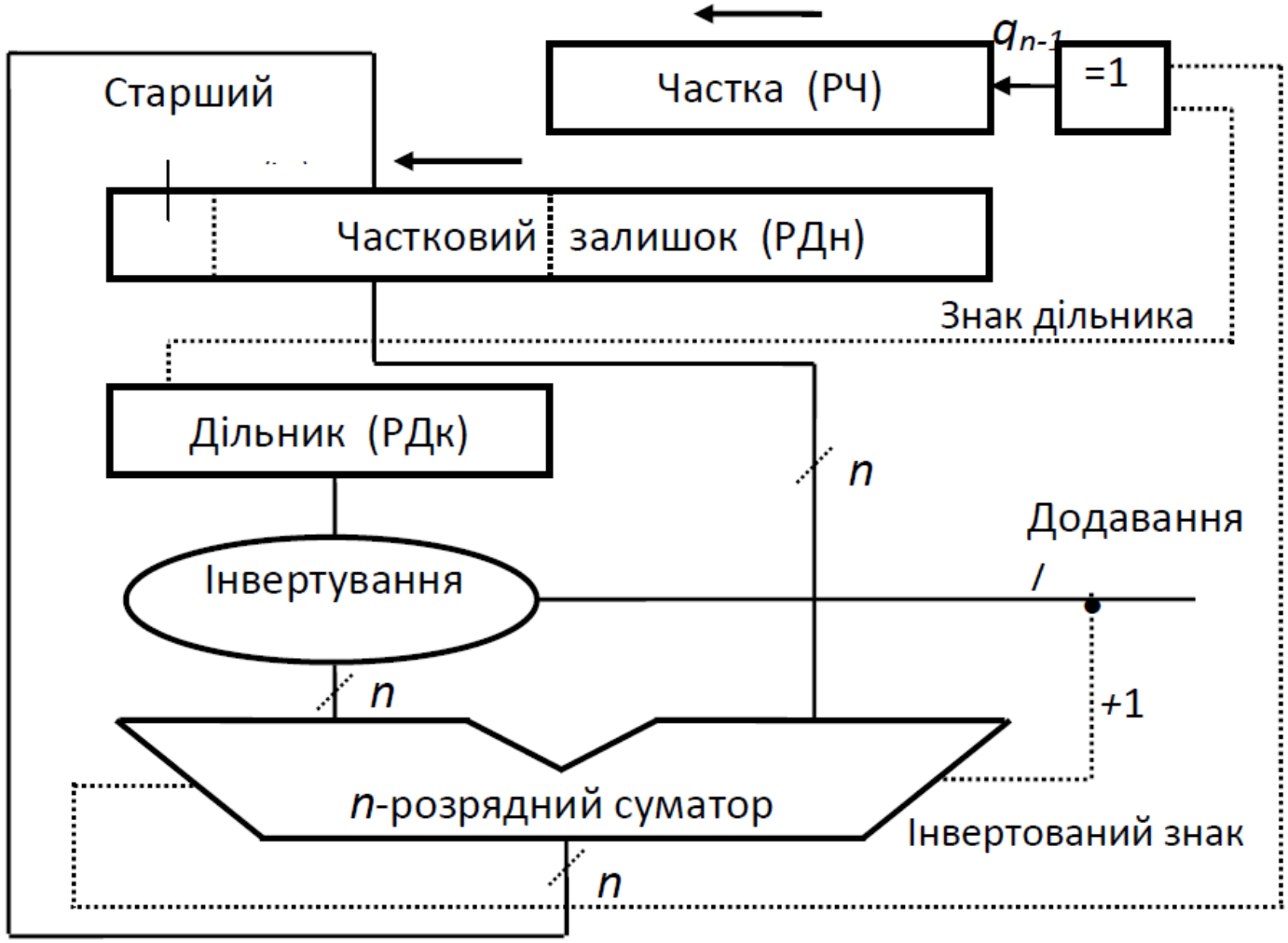
Операція, яка виконується в черговій ітерації ділення

Знак залишку	Знак дільника	Дія
+	+	Віднімання
+	–	Додавання
–	+	Додавання
–	–	Віднімання

- Якщо знак залишку збігається із знаком дільника, то чергова цифра частки − 1, інакше − 0.
- Якщо $Z > 0$ і $D < 0$, частку необхідно збільшити на 1.
- Якщо $Z < 0$ і $D > 0$, то при ненульовому залишку від ділення частку потрібно збільшити на одиницю.
- Якщо $Z < 0$ і $D < 0$, то у разі нульового залишку від ділення частку потрібно збільшити на одиницю.

Залишок завжди приводиться до додатного числа, тобто якщо після закінчення ділення він від'ємний, до нього слід додати модуль дільника.

Пристрій ділення



Процедура починається із занесення діленого в 2*n*-розрядний регістр діленого (РДн) і дільника в *n*-розрядний регістр дільника (РДк). В лічильник циклу (ЛчЦ – на схемі не показаний), який служить для підрахунку кількості отриманих цифр частки, поміщається початкове значення, яке дорівнює *n*.

На кожному кроці вміст регістра діленого (РДн) і регістра частки (РЧ) зсовується на один розряд вліво. Залежно від поєднання знаків часткового залишку і дільника визначається значення чергової цифри частки і необхідна дія: віднімання або додавання дільника. Віднімання дільника проводиться за допомогою додавання доповняльного коду дільника.

Перетворення в доповняльний код здійснюється за рахунок передачі дільника на вхід суматора зворотним (інверсним) кодом з подальшим

додаванням одиниці до молодшого розряду суматора. Описана процедура повторюється до вичерпання всіх цифр діленого, про що свідчить нульовий вміст лічильника циклів ЛчЦ. Після закінчення операції ділення частка розташовується в регістрі частки, а в регістрі діленого буде залишок від ділення.

На завершальному етапі, якщо це необхідно, проводиться коректування отриманого результату.

На практиці для накопичення і зберігання частки замість окремого регістра використовують молодші розряди регістра діленого, що звільняються в процесі зсувів

Прискорення цілочисельного ділення

- заміна дільника зворотною величиною і її множенням на ділене;
- скорочення часу обчислення часткових залишків за рахунок прискорення операцій підсумовування (віднімання);
- скорочення часу обчислення за рахунок зменшення кількості операцій підсумовування (віднімання) під час розрахунку ЧЗ;
- обчислення частки в надмірній системі числення.

Операцію множення можна проводити порівняно швидко, якщо взяти на озброєння комбінаційні схеми паралельного множення. Дану обставину можна використовувати, замінивши операцію ділення на D множенням

$$Q = \frac{1}{D} = Z \times \frac{1}{D}$$

У цьому випадку проблема зводиться до ефективного обчислення 1/*D*. Зазвичай задача вирішується одним з двох методів: за допомогою ряду Тейлора або методу Ньютона-Рафсона. В обох випадках основний час витрачається на множення, тому даний метод має сенс за наявності швидких схем множення. Для двійкового подання *D* можна записати

$$\frac{1}{D} = (1 - X) \times (1 + X^2) \times (1 + X^4) \times (1 + X^8) \times (1 + X^{16}) \ldots$$

Метод був використаний у моделі обчислювальної машини IBM 360 для обчислення 32-розрядної величини 1/*D*. Можливі значення співмножників у правій частині виразу витягувалися з таблиці ємністю 28 байт, яка зберігалася в пам'яті. Операція обчислення 1/*D* вимагає шести множень.

Обчислення величини $1/D$ методом Ньютона-Рафсона зводиться до знаходження кореня рівняння $f(X) = \frac{1}{X} - D = 0$. Рішення отримують із залученням рекурентного співвідношення: $X_{i+1} = X_i\ (2 - X_i\ D)$. Кількість ітерацій визначається точністю обчислення $1/D$, для n-розрядних чисел вимагає $2\ \text{int}\ (\log 2\ n) - 1$ операцій множення.

Загалом, заміна операції ділення на множення характерніша для чисел з плаваючою комою.

В основі **третьої групи методів прискорення операції ділення** лежить так званий алгоритм SRT (Sweeney, Robertson, Tocher), який є модифікацією ділення без відновлення залишку. В стандартній про- цедурі на кожному кроці крім зсуву часткового залишку проводиться додавання або віднімання

дільника. В SRT-алгоритмі зсув ЧЗ також є в кожній ітерації, проте додавання або віднімання, залежно від ЧЗ, що виходить, на окремих кроках може не виконуватися, що, природно, позитивно впливає на швидкодію ділення. Алгоритм був орієнтований на операції над мантисами чисел з плаваючою комою і спирається на ту обставину, що мантиси в таких числах нормалізовані. В даний час він широко застосовується в мікропроцесорах фірми Intel.

Найбільш поширені методи прискорення операції ділення засновані на застосуванні алгоритмів, де ділення виконується в **надмірних системах числення**, тобто в системах числення, відмінних від двійкової.

Виконання арифметичних операцій з плаваючою комою

Операції над числами у форматі з плаваючою комою (ПК) мають істотні відмінності від аналогічних операцій цілочисельної арифметики, тому їх зазвичай реалізують за допомогою самостійного операційного пристрою. Як і цілочисельний ОПр, операційний пристрій для чисел у форматі ПК як мінімум повинен забезпечувати виконання чотирьох арифметичних дій: додавання, віднімання, множення і ділення.

У стандарті IEEE 754 мантиси чисел *M* подаються у нормалізованому вигляді, при цьому діє прийом прихованого розряду, коли старша цифра мантиси, яка завжди дорівнює одиниці, в записі числа відсутня, тобто в полі мантиси старшою є друга старша цифра нормалізованої мантиси.

На відміну від загальноприйнятої умови нормалізації S = |*M*| < 1, в стандарті IEEE 754 використовується умова 1 ≤ |M| < 2. Запис числа містить зміщений порядок, збільшений на величину зміщення, яке в стандарті IEEE 754 для одинарного формату дорівнює 127, а для подвійного − 1023.

$$\pm Z_M \times 2^{Z_{СП}} = (\pm X_M \times 2^{X_{СП}}) \Diamond (\pm Y_M \times 2^{Y_{СП}}),$$

де X_M, Y_M, Z_M − нормалізовані мантиси операндів і результату;

$X_{СП}$, $Y_{СП}$, $Z_{СП}$ − зміщені порядки операндів і результату;

◊ − знак арифметичної операції.

При всіх відмінностях у виконанні різних арифметичних операцій підготовчий і завершальний етапи у всіх випадках збігаються, через що має сенс розглянути їх окремо.

Підготовчий етап. Для чисел з плаваючою комою операції над трьома складовими чисел з ПК (**знаками, мантисами і порядками операндів**) виконуються роздільно блоками **обробки знаків** (БОЗ), **обробки порядків** (БОП) і **обробки мантис** (БОМ). Для зберігання операндів і результату в ОП передбачені відповідні регістри. Хоч ці регістри можуть бути фізично реалізовані у вигляді єдиних пристроїв, кожен з них логічно розглядати як сукупність трьох регістрів: **знаку, порядку і мантиси**. На етапі завантаження операндів у регістри ОП здійснюється «**розпаковка**» чисел з ПК, їх розбиття на три складові: в старшому розряді регістра мантиси відновлюється одиниця, яка в записі числа була відсутня (була **прихована**).

Виконується перевірка на рівність нулю операндів (у IEEE 754 нулю дорівнюють усі розряди порядку): якщо нулю рівні множник, множене або ділене, результат відразу можна нульовим, **обійшовши інші дії**.

Завершальний етап зводиться до виявлення **нульового значення** мантиси (**втрати значимості** мантиси), **нормалізації** мантиси, виявлення **від'ємного переповнювання** порядку, "**упаковки**" складових результату.

Нульове значення мантиси може вийти в результаті час складання або віднімання мантис. Другою причиною може стати зсув мантиси вправо для усунення переповнення. В обох випадках має місце ситуація **втрати значимості мантиси**, і результат операції приймається рівним нулю. Для IEEE 754 це означає, що всі цифри порядку результату необхідно **обнулити**, а нормалізацію мантиси результату проводити **не потрібно**.

Нормалізація мантиси результату зводиться до послідовного її зсуву вліво до тих пір, поки старшу позицію не займе одиниця. Кожен зсув супроводжується зменшенням на одиницю порядку результату. В ході зменшення порядок може стати від'ємним, що для зміщених порядків свідчить про отримання числа, неуявного в даному форматі.

В такій ситуації результат приймається рівним нулю і одночасно формується ознака **втрати значимості порядку**.

На завершення мантиса результату округляється і, якщо це передбачено форматом ПК, з неї видаляється прихований розряд.

В останній фазі здійснюється «**упаковка**» всіх складових результату (знака, порядку і мантиси), після чого сформований результат заноситься у вихідний регістр ОП.

Додавання і віднімання

Алгоритм додавання і віднімання включає такі основні фази:

1. Підготовчий етап.

2. Визначення операнду, що має менший порядок, і зсув його мантиси вправо на число розрядів, яке дорівнює різниці порядків операндів.

3. Прирівнювання порядку результату більшому з порядків операндів.

4. Додавання або віднімання мантис і визначення знака результату.

5. Перевірку на переповнювання.

6. Завершальний етап.

Додавання і віднімання виконуються ідентично, але у разі віднімання необхідно змінити знак другого операнда на протилежний.

Множення

Спочатку проводиться перевірка **на рівність нулю** співмножників. Якщо один з операндів дорівнює нулю, як результат видається 0. Наступний крок – **додавання порядків**. У разі **переповнення** порядку або **втрати значимості** виконання операції припиняється і видається повідомлення. Якщо ні, проводиться **перемножування мантис** з урахуванням їх знаку (як для чисел з фіксованою комою) і розміщення добутку у розрядній сітці (мантиси подаються не цілими числами, а правильними дробами). Результат множення мантис має подвоєну, але він округляється до довжини поля мантиси. На останньому кроці проводиться **нормалізація** і **компоновка** результату, аналогічно тому, як це має місце під час додавання і віднімання.

Ділення

Спочатку також проводиться перевірка на 0. Якщо нулю дорівнює дільник, залежно від реалізації видається повідомлення про ділення на 0, або як результат приймається нескінченність. Коли нулю дорівнює ділене, результат також приймається рівним нулю.

Далі виконується віднімання порядку дільника з порядку діленого, що приводить до видалення зсуву з порядку результату. Отже, для отримання зміщеного порядку результату до різниці повинен бути доданий зсув. Після виконання цих дій необхідна перевірка на переповнювання порядків і втрату значимості.

Наступний крок − ділення мантис, за яким йдуть нормалізація, округлення і компоновка числа з мантиси і порядку.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. В. І. Жабін, І. А. Клименко, В. В. Ткаченко Комп’ютерна логіка: Практикум [Електронний ресурс]: навч. посіб. для студ. спеціальності 123 «Комп’ютерні системи та мережі», спеціалізацій «Комп’ютерні системи та мережі» та «Технології програмування для комп’ютерних систем та мереж». КПІ ім. Ігоря Сікорського, 2019. 97 с.
2. Комп’ютерна логіка. Прикладна теорія цифрових автоматів: комп’ютерний практикум [Електронний ресурс] : навч. посіб. для студ. спеціальності 121 «Інженерія програмного забезпечення», спеціалізації «Програмне забезпечення комп’ютерних та інформаційно-пошукових систем» / І. А. Дичка, В. П. Легеза, М. В. Онай ; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 3,85 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2018. – 88 с.
3. Прикладна теорія цифрових автоматів: навчальний посібник / уклад.: О. Я. Олар, Г. І. Воробець, Р. І. Макарчук, Ю. Ю. Блошко, Чернівці: ЧНУ, 2022. 200 с.
4. Elahi, Ata Computer systems: digital design, fundamentals of computer architecture and assembly language / Ata Elahi – Cham, Switzerland: Springer, 2018. – 251 p.

Олевський Віктор Ісаакович
Олевська Юлія Борисівна
Соколова Наталя Олегівна

Конспект лекцій з дисципліни «Комп’ютерна логіка»
для здобувачів ступеня бакалавра
спеціальності 123 Комп’ютерна інженерія

За редакцією авторів

Електронний ресурс

Національний технічний університет «Дніпровська політехніка»
49005, м. Дніпро, просп. Д. Яворницького, 19.