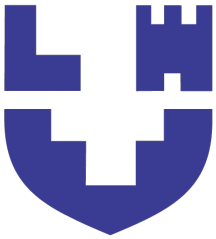
Міністерство освіти і науки України
Луцький національний технічний університет



КОДУВАННЯ ІНФОРМАЦІЇ ТА АРХІТЕКТУРА КОМП'ЮТЕРА

Конспект лекцій
для здобувачів першого (бакалаврського) рівня вищої освіти
освітньої програми «Професійна освіта (комп'ютерні технології)»
галузі знань А Освіта
спеціальності А5.39 Професійна освіта (Цифрові технології)
денної та заочної форм навчання

Луцьк 2026

УДК 004.2:004.3(07)
К-57

До друку

Голова вченої ради факультету цифрових, освітніх та соціальних технологій ЛНТУ _____ Г. ГЕРАСИМЧУК

Затверджено вченою радою факультету цифрових, освітніх та соціальних технологій ЛНТУ, протокол № ____ від «____»_____ 2026 року.

Електронна копія друкованого видання передана для внесення в репозитарій ЛНТУ.
Директор бібліотеки _____ Н. ПОЛІЩУК

Рекомендовано до видання на засіданні кафедри цифрових освітніх технологій ЛНТУ, протокол № _____ від «____»_____ 2026 року.
Завідувач кафедри цифрових освітніх технологій _____ В. КАБАК

Укладач: _____ П. САВАРИН, кандидат педагогічних наук, доцент кафедри цифрових освітніх технологій ЛНТУ.

Рецензент: _____В. КАБАК, кандидат педагогічних наук, доцент кафедри цифрових освітніх технологій ЛНТУ.

Відповідальний за випуск:_____ В. КАБАК, кандидат педагогічних наук, доцент, завідувач кафедри цифрових освітніх технологій ЛНТУ.

К-57 **Кодування інформації та архітектура комп'ютера** : Конспект лекцій для здобувачів першого (бакалаврського) рівня вищої освіти освітньої програми «Професійна освіта (комп'ютерні технології)» галузі знань А Освіта спеціальності А5.39 Професійна освіта (Цифрові технології) денної та заочної форм навчання / уклад. П. САВАРИН. Луцьк: ЛНТУ, 2026. – 192 с.

Методичне видання складене відповідно до діючої програми курсу «Кодування інформації та архітектура комп'ютера» і містить необхідний теоретичний матеріал, який дасть можливість здобувачам освіти опанувати необхідними компетентностями з мінімальними витратами часу.

Зміст

Тема 1. Коди та комбінації …… 4
Тема 2. Брайль та двійкові коди …… 8
Тема 3. Телеграфи та реле …… 13
Тема 4. Наші десять цифр …… 18
Тема 5. Альтернативи десятці …… 23
Тема 6. Логіка та перемикачі …… 37
Тема 7. Логічні вентилі …… 50
Тема 8. Двійковий суматор …… 78
Тема 9. Байти та шістнадцяткові числа …… 88
Тема 10. Складання пам'яті …… 97
Тема 11. Від рахівниць до мікросхем …… 111
Тема 12. Два класичні мікропроцесори …… 128
Тема 13. Набір символів ASCII …… 151
Тема 14. Шини …… 166
Тема 15. Фіксована крапка, плаваюча крапка …… 181
Список використаної літератури …… 191

Тема 1. Коди та комбінації

Абетку Морзе придумав Семюель Фінлі Бріз Морзе (Samuel Finley Breese Morse) (1791–1872). Цей винахід невіддільний від створення телеграфу, про роботу якого нам також доведеться дізнатися. Абетка Морзе послужила хорошим вступним матеріалом для знайомства з сутністю коду, а телеграф – такий самий зручний приклад, що ілюструє апаратне забезпечення комп'ютера.

Багатьом здається, що абетку Морзе простіше передавати, ніж приймати. Навіть якщо ви не знаєте її на пам‘ять, можете просто звірятися з таблицею, де літери для зручності розставлені за абеткою.

Приймати абетку Морзе і переводити її у звичайні слова значно складніше і довше, оскільки ви працюєте у зворотному порядку: з'ясовуєте, яка літера відповідає конкретній кодовій послідовності крапок та тире. Наприклад, якщо ви отримаєте сигнал «тире-крапка-тире-тире», доведеться заглянути в таблицю і переглянути майже всі літери одну за одною, доки не з'ясується, що перед вами літера *Y.*

Проблема в тому, що ми маємо таблицю для наступного перекладу:
буква алфавіту → послідовність абетки Морзе, що складається з крапок і тире.

Однак немає зворотної таблиці:
послідовність абетки Морзе, що складається з крапок та тире, → літера алфавіту.

На початку вивчення азбуки Морзе така таблиця, безумовно, стала б у нагоді. Щоправда, не зовсім зрозуміло, як її скласти. Крапки та тире не допускають жодної подоби алфавітного порядку.

Давайте забудемо про алфавіт. Мабуть, розумніше згрупувати коди таким чином, щоб їхнє розміщення залежало від кількості крапок і тире в тій чи іншій літері. Так, послідовність з абетки Морзе, що містить одну крапку та одне тире, може означати лише одну з двох літер: *E* або *T.*

E	•
T	—

Комбінації, в яких міститься по два знаки (або крапки, або тире), дають нам вже чотири

I	• •
A	• —
N	— •
M	— —

літери: *I, A, N* та *M.*

S	• • •	D	— • •
U	• • —	K	— • —
R	• — •	G	— — •
W	• — —	O	— — —

Паттерн із трьох символів, крапок або тире, дає нам вісім букв.

Нарешті (якщо ми хочемо припинити цю вправу, поки не перейшли до цифр і розділових знаків), чотиризначні послідовності крапок і тире дають нам ще 16 символів.

H	• • • •	B	— • • •
V	• • • —	X	— • • —
F	• • — •	C	— • — •
Ü	• • — —	Y	— • — —
L	• — • •	Z	— — • •
Ä	• — • —	Q	— — • —
P	• — — •	O	— — — •
J	• — — —	Ş	— — — —

Загалом у цих таблицях міститься 2 + 4 + 8 + 16 кодів сумарно для 30 букв; це на чотири коди більше, ніж потрібно для повної латиниці, що складається з 26 літер. Саме тому чотири коди в останній таблиці відведені під букви з діакритичними знаками.

Ці чотири таблиці допоможуть легко перекладати будь-які повідомлення, що передаються абеткою Морзе. Отримавши код конкретної літери, ви рахуєте, скільки в ньому крапок і тире, і вирішуєте, з якою таблицею звірятися. Кожна таблиця влаштована так, що код, який складається з одних крапок, розташовується у верхньому лівому куті, а код з одних тире — у правому нижньому куті.

Чи помічаєте закономірність у *розмірах* чотирьох таблиць? Зверніть увагу: у кожній таблиці вдвічі більше кодів, ніж у попередній. Це логічно: у наступній таблиці містяться всі коди з попередньої «плюс крапка», а також усі коди з попередньої «плюс тире».

Цю тенденцію можна резюмувати в такий спосіб.

Number of Dots and Dashes	Number of Codes
1	2
2	4
3	8
4	16

Кожна з чотирьох таблиць містить вдвічі більше кодів, ніж попередня таблиця, так що якщо в першій таблиці 2 коди, то в другій – 2 × 2 кодів, в третій – 2 × 2 × 2 кодів. Ось як ще можна це уявити.

Number of Dots and Dashes	Number of Codes
1	2
2	2 x 2
3	2 x 2 x 2
4	2 x 2 x 2 x 2

Зрозуміло, при множенні числа самого на себе можна використовувати степені. Так, 2×2×2×2 можна записати як 2^4 (2 у четвертому степені). Числа 2, 4, 8 і 16 є степенями двійки, оскільки їх можна отримати множенням двійки самої на себе. Отже, нашу таблицю можна переписати і так.

Number of Dots and Dashes	Number of Codes
1	2^1
2	2^2
3	2^3
4	2^4

Таблиця дуже спростилася. Кількість кодів дорівнює просто 2 у степені <кількість крапок і тире>. Можна резюмувати табличні дані у вигляді простої формули:

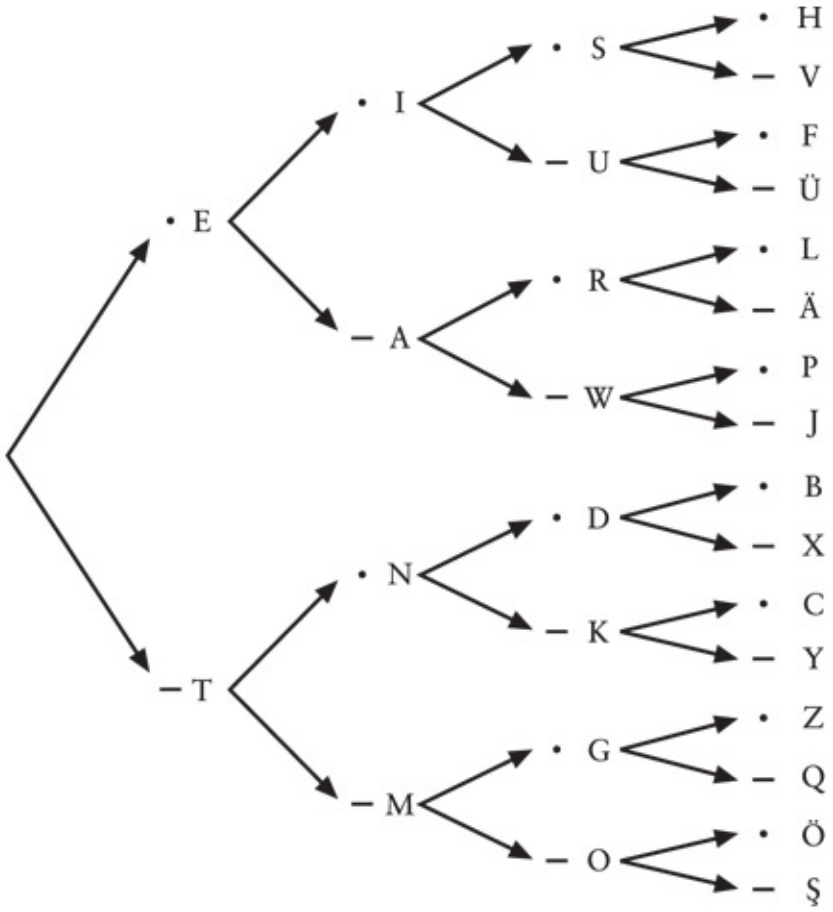
Кількість кодів = 2 кількість крапок і тире.

Степінь двійки часто використовується в різних кодах (інший приклад розглянемо в наступній темі).

Щоб ще більше спростити розшифровку коду Морзе, спробуємо побудувати велику деревоподібну схему на наступній сторінці.

На схемі показано, які літери виходять за умови поступового ускладнення послідовностей крапок і тире. Щоб розшифрувати конкретну послідовність, йдіть стрілками зліва направо. Припустимо, ми хочемо з'ясувати, яка літера відповідає коду "крапка-тире-крапка". Починаємо зліва, беремо крапку; далі йдемо за стрілками, вибираємо тире, а потім ще одну крапку. Отримуємо букву *R,* розташовану біля останньої крапки.

Така схема необхідна насамперед для того, щоб визначити код Морзе. По-перше, вона страхує від тупої помилки: не дає присвоїти двом різним буквам той самий код. По-друге, ви гарантовано використовуєте всі можливі коди, не вибудовуючи надмірно довгих послідовностей з крапок і тире.



Ризикуючи отримати схему, яка не поміститься на друкованій сторінці, ми могли б розширити її та додати туди п'ятизначні коди з крапок та тире. Послідовність п'яти крапок і тире дасть нам 32 (2 × 2 × 2 × 2 × 2, або 2^5) додаткового коду. Як правило, цього достатньо не тільки для букв, але і для 10 цифр і 18 розділових знаків, що включаються в абетку Морзе: цифри дійсно кодуються п'ятизначними послідовностями крапок і тире. Щоправда, багато інших п'ятизначних кодів зарезервовані не за розділовими знаками, а за літерами з діакритичними знаками.

Щоб система враховувала всі розділові знаки, до неї потрібно включити послідовності з шести крапок і тире. Таким чином отримаємо 64 (2 × 2 × 2 × 2 × 2 × 2, або 2^6) додаткових кодів для сумарної множини з 2 + 4 + 8 + 16 + 32 + 64, або 126, символів. Для абетки Морзе цього дуже багато, тому більшість таких довгих кодів залишаються невизначеними. Слово "*невизначений*" у даному контексті вказує на код, який нічого не означає. Якби ви, приймаючи абетку Морзе, отримали невизначений код, то могли б майже не сумніватися, що хтось просто припустився помилки.

У нас вистачило кмітливості побудувати цю невелику формулу:

$$\text{Кількість кодів} = 2^{\text{кількість крапок і тире.}}$$

Так давайте продовжимо нашу таблицю і подивимося, скільки кодів вийде з довших послідовностей крапок і тире.

Number of Dots and Dashes	Number of Codes
1	$2^1 = 2$
2	$2^2 = 4$
3	$2^3 = 8$
4	$2^4 = 16$
5	$2^5 = 32$
6	$2^6 = 64$
7	$2^7 = 128$
8	$2^8 = 256$
9	$2^9 = 512$
10	$2^{10} = 1024$

На щастя, не потрібно виписувати всі можливі коди, щоб визначити, скільки їх буде. Достатньо множити двійку на себе потрібну кількість разів.

Код Морзе називається *двійковим* (що буквально означає «два на два»), оскільки будь-який його елемент включає лише два компоненти: крапку та тире. Такий код подібний до монети, яка може впасти лише гербом або копійкою. Двійкові об'єкти (наприклад, монети) та двійкові коди (наприклад, абетка Морзе) завжди можна описати у вигляді ступенів двійки.

Проведений нами аналіз двійкових кодів – це проста вправа в одній математичній дисципліні, яка називається *комбінаторикою,* або *комбінаторним аналізом.* Традиційно комбінаторний аналіз особливо активно використовується в теорії ймовірностей та статистиці, оскільки пов'язаний з виявленням кількості варіантів комбінацій різних об'єктів (наприклад, монет або гральних кубиків). Він також допомагає зрозуміти, як складаються та розбираються коди.

Тема 2. Брайль та двійкові коди

Семюел Морзе не був першим, кому успішно вдалося транслювати літери писемної мови в код, що інтерпретується. Він не був першим і серед тих, чиє прізвище запам'яталося як назва коду, а не власне ім'я. Така честь випала сліпому французькому підлітку, що народився приблизно через 18 років після Морзе, але залишив слід в історії набагато раніше. Про життя Луї Брайля відомо небагато, але це захоплююча історія.

Луї Брайль народився 1809 року у французькому містечку Куврі, за 40 кілометрів на схід від Парижа. Батько хлопчика був упряжником. Будучи трьох років від народження (а в такому віці діти не повинні грати в батьківській майстерні), Луї випадково тицьнув собі в око упряжним ножем. У рані почався процес зараження, інфекція поширилася і друге око, і хлопчик повністю осліп. Напевно на нього чекало життя в невігластві та бідності (як і більшість сліпих у ті часи), але Луї виявив неабиякий розум і потяг до знань. Завдяки участі сільського пастора та шкільного вчителя Луї ходив до сільської школи разом з іншими хлопцями, а у віці десяти років вирушив до Паризького державного інституту для сліпих дітей.

Очевидно, одна з головних складнощів при навчанні незрячих у тому, що вони не можуть читати друковані книги. Засновник цієї паризької школи Валентин Гаюї (1745–1822) винайшов систему тиснених опуклих літер для читання їх на дотик. Але користуватися системою було складно, і вийшло лише кілька книг, надрукованих таким методом. Гаюї не зміг подивитися глибше. Для нього буква *А* залишалася буквою *А.* Вона мала виглядати (відчуватися) як *А.* (Спілкуючись мовою світлових сигналів, ми намагалися малювати літери у повітрі й переконалися, що такий прийом непрацездатний.) Мабуть, Гаюї не здогадався, що код, який сильно відрізняється від друкованого алфавіту, виявився б зручнішим для незрячих.

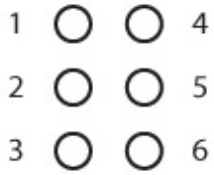
Прообраз такого альтернативного коду виник у досить незвичайному контексті. Шарль Барбіє, капітан французької армії, винайшов систему запису під назвою écriture nocturne, або «нічна абетка». У ній використовувалися візерунки опуклих крапок та тире на щільному папері. Передбачалося, що солдати могли б обмінюватися в темряві такими записками, коли потрібно дотримуватися тиші. Писати крапки та тире можна було спеціальним стилусом, на зразок шила. Потім опуклі крапки можна було читати навпомацки. Недолік системи Барб'є полягав у її надмірній складності. Комбінації крапок та тире відповідали звукам, а не буквам алфавіту, тому одне слово часто могло шифруватися різними кодами. Система добре працювала для обміну короткими повідомленнями в польових умовах, але зовсім не підходила для порівняно великих текстів, тим більше книг.

Луї Брайль познайомився із системою Барб'є у дванадцятирічному віці. Йому сподобалися опуклі крапки не тільки тому, що вони легко читалися навпомацки, а й тому, що їх було просто *писати.* Учень у класі, озброївшись папером та стилусом, справді міг записувати та читати такі повідомлення. Луї Брайль постарався вдосконалити цю систему, і через три роки (коли йому було п'ятнадцять) в загальних рисах склав власну, основи якої застосовуються і сьогодні. Багато років така система використовувалася лише в школах, але поступово увійшла у широке вживання. В 1835 Брайль підхопив туберкульоз, від якого і помер в 1852, у віці 43 років.

Сьогодні удосконалені варіанти системи Брайля змагаються з аудіокнигами, забезпечуючи незрячим доступ до письмової інформації. Проте шрифт Брайля, як і раніше, незамінний і є єдиною писемністю, доступною сліпоглухим. Шрифт Брайля застосовується навіть у громадських місцях, наприклад у ліфтах та банкоматах.

У цьому розділі ми розберемося як він працює. Ми не будемо *вивчати* код Брайля або щось запам'ятовувати. Ми лише спробуємо на цьому прикладі краще зрозуміти його природу.

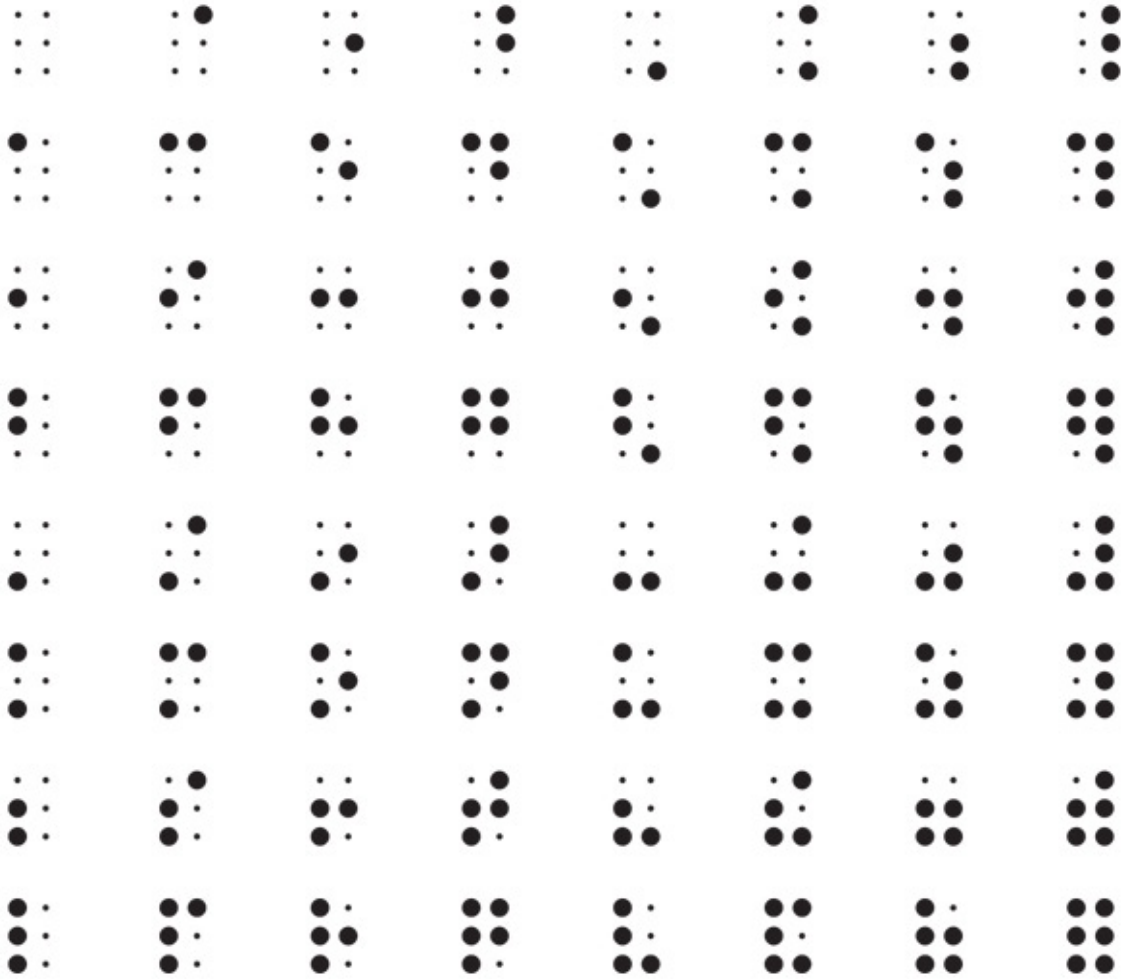
У шрифті Брайля кожен символ, присутній у звичайній письмовій мові, тобто літери, цифри та розділові знаки, кодується у вигляді однієї або декількох крапок у клітці розміром дві на три точки. Як правило, крапки у клітині нумеруються від 1 до 6.



В даний час існують спеціальні друкарські машинки – брайлівські принтери, що вибивають точки брайлівського шрифту на папері.

Оскільки книга вийшла б дуже дорогою, якби хоч пару сторінок набрали шрифтом Брайля, я користувався нотацією, що традиційно застосовується для передачі азбуки Брайля під час друку. У такій нотації відображаються всі шість крапок у клітці. Жирні точки – це опуклості на папері, дрібні – плоскі елементи клітини. Наприклад, у наступному брайлівському символі точки 1, 3 та 5 опуклі, а 2, 4 та 6 – ні.

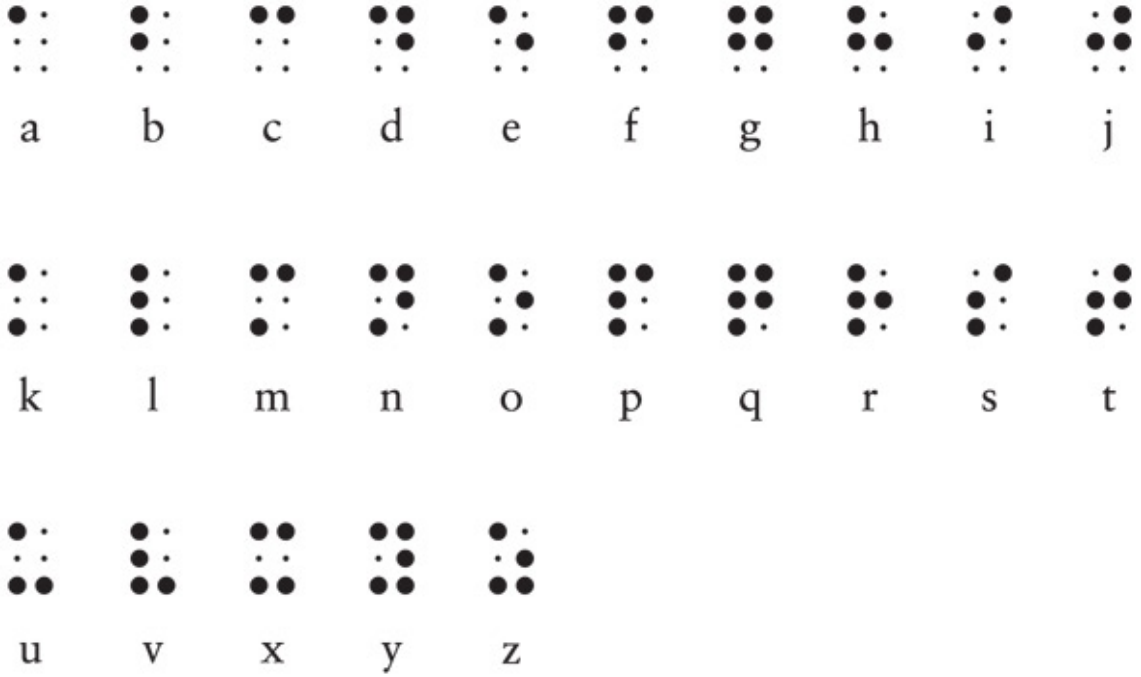
На даний момент нас має зацікавити, що ці точки *двійкові.* Будь-яка крапка може бути або опуклою, або плоскою. Таким чином, шрифт Брайля підпорядковується тим же принципам, які знайомі нам з абетки Морзе та комбінаторного аналізу. Відомо, що в клітині шість крапок, і кожна крапка може бути плоскою або опуклою, тому загальна кількість комбінацій, які складаються з шести плоских або опуклих крапок, дорівнює $2 \times 2 \times 2 \times 2 \times 2 \times 2$, або 2^6 або 64.



Як бачите, в системі Брайля можна представити 64 унікальні коди.

Якщо в шрифті Брайля використовується менше 64 кодів, то логічно запитати, чому не всі можливі варіанти в ході. Якщо в шрифті Брайля знайдеться понад 64 можливі коди, значить, збоїть або наш розум, або фундаментальні математичні істини з розряду «два плюс два і чотири».

Приступаючи до вивчення шрифту Брайля, розглянемо, як у ньому записуються малі літери латиниці.



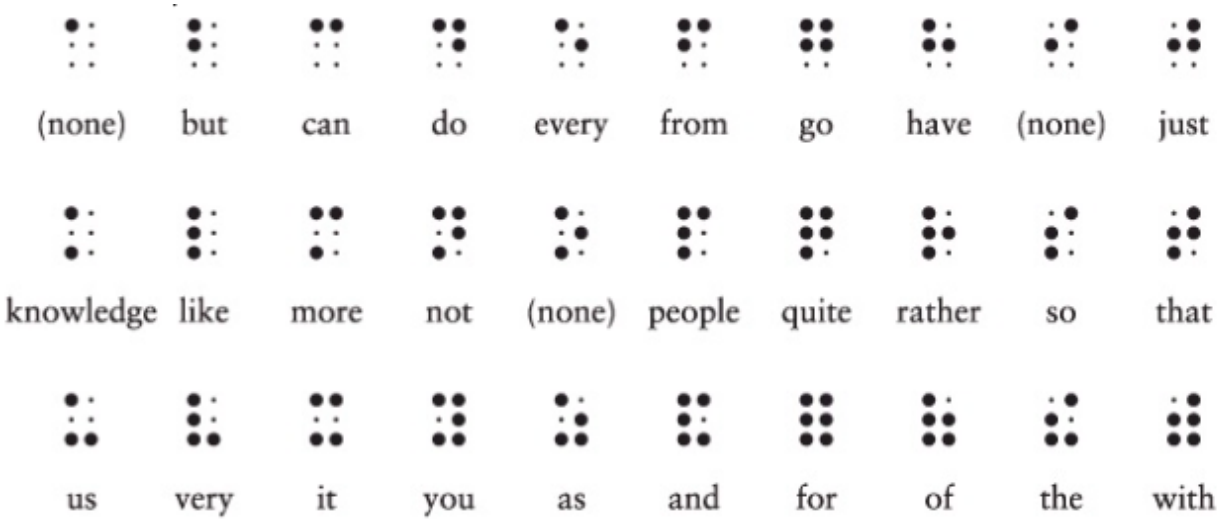
Наприклад, англійська фраза You and me записується в такий спосіб.

Важливо: між клітинами, що відповідають буквам у слові, ставляться невеликі прогалини; більш широка прогалина (по суті, ціла клітина, в якій немає опуклих крапок) відповідає пробілу між словами.

Саме такою є основа шрифту Брайля в редакції самого Брайля – як мінімум щодо латиниці. Луї Брайль також розробив коди для літер із діакритичними знаками (вони часто зустрічаються у французькій мові). Зверніть увагу: тут немає коду для літери *w,* яка в класичному французькому не використовується. (Не хвилюйтеся, і ця літера незабаром з'явиться.) Поки ми врахували лише 25 із 64 можливих кодів.

Уважно придивившись до наведених вище рядків, можна помітити, що в них простежується закономірність. У першому рядку (від *a* до *j*) у кожній клітині використовуються лише чотири верхні точки: 1, 2, 4 і 5. Другий ряд такий самий, як перший, але в ньому є і крапка 3. Третій ряд подібний до перших двох, але у ньому бачимо не тільки крапку 3, а й крапку 6.

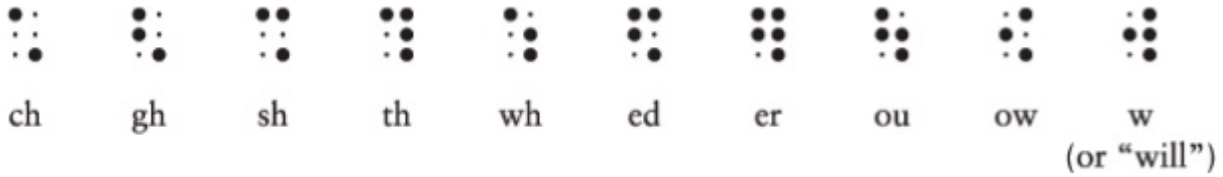
З часів Луї Брайля його шрифт доповнювали по-різному. Сучасна система, з допомогою якої зазвичай записуються подібні англійські тексти, називається «скорочений Брайль». У скороченому Брайлі багато спрощень, які допомагають берегти дерева та прискорювати читання. Наприклад, якщо код певної літери стоїть окремо, він означає поширене слово. У наступних трьох рядах наведено такі коди для слів.



Таким чином фразу You and me скороченим Брайлем можна записати так.

Ось ми й описали 31 код: пробіл без крапок, який ставиться між словами, і три рядки по десять кодів, що використовуються для позначення букв та слів. Ми досі і близько не витратили 64 теоретично доступні коди. Як ми переконаємося, у скороченому Брайлі жоден не залишився без діла.

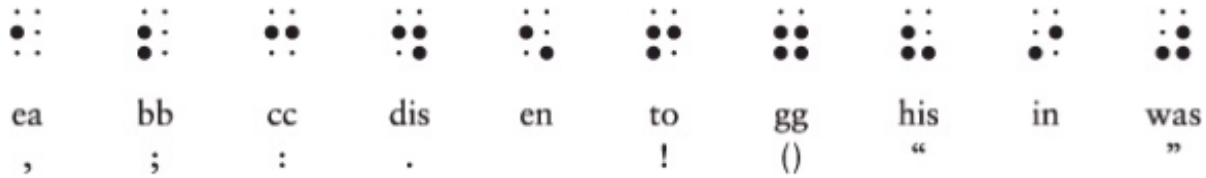
По-перше, можна використовувати коди букв *a - j,* додаючи до кожного з них опуклу крапку 6. Ці коди застосовуються в основному для скорочення в слові букв, для *w* і іншого скорочення слів.



** Will - допоміжне дієслово для утворення майбутнього часу.*

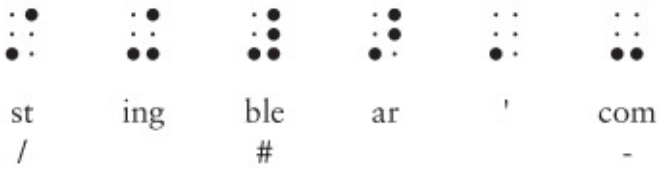
Наприклад, слово about можна записати скороченим Брайлем ось так.

По-друге, можна взяти коди букв *a – j* і зробити так, щоб використовувалися лише точки 2, 3, 5 і 6. Цими кодами позначаються деякі розділові знаки та скорочення, залежно від контексту.



Перші чотири наведені коди – це кома, крапка з комою, двокрапка і крапка. Зверніть увагу: як відкриваюча, так і закриваюча дужки позначаються одним і тим же кодом, а ось коди для лапки, що відкривається і закривається, відрізняються.

Поки що ми використовували 51 код. Далі наведено шість кодів, що представляють різні незадіяні комбінації крапок 3, 4, 5 і 6. З їх допомогою записують скорочення та деякі додаткові розділові знаки.



Код ble дуже важливий: якщо це не частина слова, то він означає, що наступні коди повинні інтерпретуватися як числа. Числові коди такі самі, як і літер *a – j.*

1 2 3 4 5 6 7 8 9 0

Отже, наведена нижче послідовність означає 256.

Якщо ви стежите за ниткою розповіді, пам'ятайте, що до максимуму (64) нам залишається ще

сім кодів. Ось вони.

Перший код (опукла крапка 4) – індикатор наголосу. Інші використовуються як префікси при деяких скороченнях, а також в інших цілях. Наприклад, при опуклих крапках 4 і 6 (п'ятий код у цьому ряду) код може означати або десяткову кому (для чисел), або логічний наголос – залежно від контексту.

Нарешті (якщо вам не терпиться дізнатися, як у шрифті Брайля записуються великі літери), у нас є опукла крапка 6. Це індикатор великої літери. Наступна після такого символу літера буде у

верхньому регістрі. Наприклад, ім'я автора цієї системи записується так.

Тут індикатор великої літери, буква *l,* буквосполучення *ou,* букви *i* і *s,* пробіл, ще один індикатор великої букви, а далі – букви *b, r, a, i, l, l* і *e* (на практиці цей запис може бути ще коротшим: відкидаються дві останні літери, тому що вони не вимовляються).

Отже, ми розглянули, як шість двійкових елементів (крапок) дають 64 можливі коди – і не більше. Виходить, що з цих кодів виконують подвійну роботу залежно від контексту. Особливо цікаві «числовий» та «літерний» індикатори (при цьому другий скасовує перший). Ці коди змінюють семантику інших кодів – тих, що йдуть за ними: з літер на цифри і назад із цифр на літери. Подібні коди часто називаються кодами *старшинства* або *перемикання.* Вони змінюють семантику всіх наступних кодів, доки перемикання не буде скасовано.

Індикатор великої літери означає, що наступна (і лише наступна) літера має бути у верхньому, а не в нижньому регістрі. Такий код прийнято називати *екрануючим* і він «захищає» послідовність інших кодів від банальної, рутинної семантики і забезпечує їм нову інтерпретацію. Читаючи наступні розділи, переконаємося, що коди перемикання та коди, що екранують, постійно використовуються в ситуаціях, коли письмову мову потрібно подати в двійковому вигляді.

Тема 3. Телеграфи та реле

Семюел Морзе народився 1791 року у місті Чарльстауні. Нині це північно-східна частина Бостона. На момент народження Морзе минуло вже два роки, як ратифікували Конституцію США. Йшов перший президентський термін Джорджа Вашингтона. Людовік XVI та Марія-Антуанетта через два роки будуть обезголовлені під час Французької революції. У 1791 році Моцарт завершив свою останню оперу «Чарівна флейта» і того ж року помер у віці 35 років.

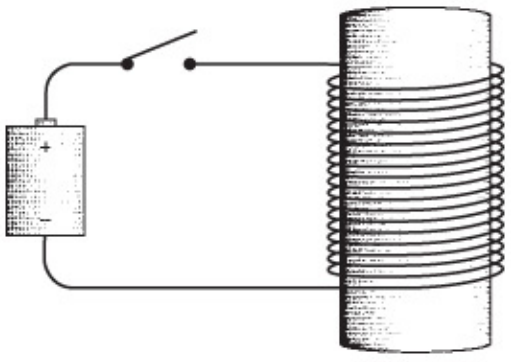
Морзе здобув освіту в Єлі та вивчав мистецтво в Лондоні. Він став успішним портретистом. Портрет генерала Лафайєта (1825) пензля Морзе досі експонується у Ратуші Нью-Йорка. В 1836 Морзе балотувався в мери Нью-Йорка як незалежний кандидат і отримав 5,7% голосів. Крім того, він був одним із перших, хто серйозно захоплювався фотографією. Морзе навчався у самого Луї Дагера і зробив одні з перших дагеротипів в Америці. У 1840 році він навчив цьому мистецтву 17-річного Метью Бреді, який разом з колегами згодом створив один із найбільш знімків Громадянської війни в США, портрети Авраама Лінкольна і Семюела Морзе. Все це лише ремарки для його різнобічної кар'єри. У наші дні Семюел Морзе найбільш відомий як винахідник телеграфу та абетки, названої на його честь.

Миттєвий зв'язок у глобальних масштабах, до якого ми так звикли, – нещодавній винахід. На початку XIX століття можна було спілкуватися або в реальному часі, або дистанційно, але те й інше одразу було неможливо. Дистанція реального спілкування була обмежена силою голосу (жодних звукопідсилювачів не існувало) і пильністю співрозмовника (щоправда, вас могли розглядати в підзорну трубу). Спілкуватися на великих відстанях можна було листуванням; для доставки листів був потрібний час і транспорт: коні, поїзди або кораблі.

За багато десятиліть до винаходу, зробленого Морзе, робилися численні спроби прискорити дистанційну комунікацію. Найпримітивніші варіанти були пов'язані з вибудовуванням ланцюжків людей-передавачів. Вони стояли на пагорбах і розмахували прапорцями, користуючись семафорною абеткою. Існували й складніші конструкції з руками-маніпуляторами, які, по суті, виконували самі функції, як і люди-семафори.

Ідея телеграфу (в буквальному перекладі з грецької «пишу далеко») на початку XIX століття виразно витала в повітрі, і крім Морзе за неї намагалися братися інші винахідники. Морзе приступив до експериментів у 1832 році. В принципі, ідея електричного телеграфу проста: на одному кінці дроту робимо якісь маніпуляції, ефект яких спостерігається на іншому кінці. Саме це й вийшло у нас, коли ми конструювали далекобійний ліхтарик. Однак Морзе не міг користуватися лампочкою як сигнальним пристроєм, оскільки саму лампочку винайшли лише в 1879 році. Натомість він задіяв явище *електромагнетизму.*

Якщо взяти залізний прут, обмотати його кількома сотнями петель тонкого дроту, а потім пропустити по цьому дроту струм, прут перетвориться на магніт. Тоді він буде притягувати інші залізні та залізні предмети. (В електромагніті вистачає тонкого дроту, щоб виникав досить великий опір, що не допускає короткого замикання). Якщо відрубати струм, то сталевий прут



втрачає магнітні властивості.

Електромагніт – основа телеграфу. Коли ми вмикаємо або вимикаємо важіль з одного боку ланцюга, ефект спостерігається з іншого боку.

Перші телеграфи Морзе були складнішими за пізніші моделі. Морзе вважав, що телеграф має виводити якусь інформацію на папері, як потім говорити комп'ютерники, створювати фізичну копію. Звичайно, це не обов'язково мають бути слова, оскільки це надто складно. Але *щось* на папері потрібно записувати, будь то каракулі або крапки та тире. Зверніть увагу: Морзе не міг вийти з площини і думав про папір і читання, як Валентин Гаюї вважав, що в книгах для сліпих мають бути опуклі літери алфавіту.

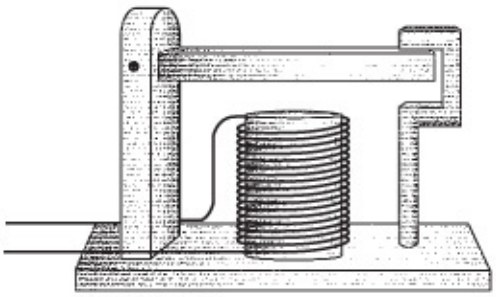
Хоча Семюел Морзе вже в 1836 році повідомив патентне бюро про те, що винайшов робочу модель телеграфу, лише в 1843 йому вдалося домогтися дозволу на публічну демонстрацію цього пристрою в Конгресі. Історичний день настав 24 травня 1844 року, коли телеграфна лінія зв'язала Вашингтон і Балтімор і по телеграфу вдалося успішно передати біблійну фразу: «Дивні справи Твої, Господи».

Звичайний телеграфний «ключ» для передачі повідомлень виглядав приблизно так:

Незважаючи на химерний вигляд, це був просто перемикач, оптимізований для максимально швидкісної роботи. Щоб довго працювати з таким ключем, його було зручніше утримувати між великим, вказівним та середнім пальцями і стукати ним вгору-вниз. Короткий удар ключем відповідав крапці з абетки Морзе, тривалий натиск – тире.

З іншого боку ланцюга розташовувався приймач, який був електромагнітом, що керував металевим важелем (спочатку електромагніт керував пером). Поки механізм, оснащений натягнутою пружиною, повільно простягав паперовий сувій через пристрій, перо скакало папером, виписуючи на ньому крапки і тире. Людина, яка вміє читати абетку Морзе, переводила ці точки і тире в букви і складала слова.

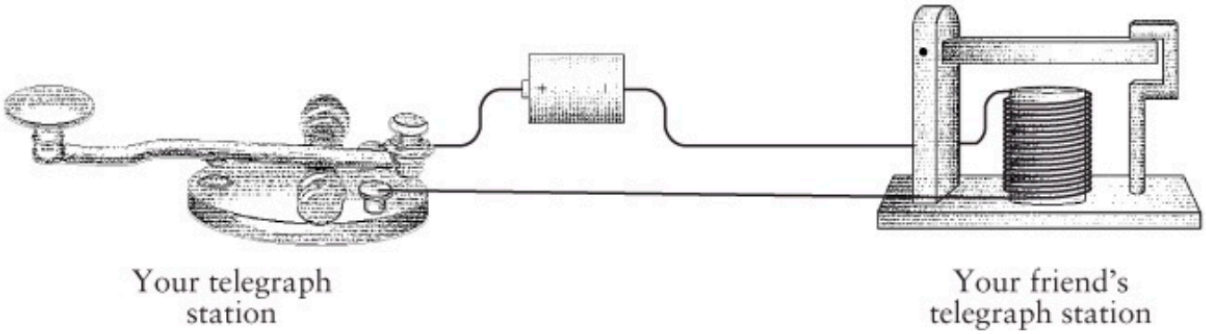
Так, люди ліниві, і телеграфісти незабаром виявили, що код можна перекладати, прислухаючись до тривалості ударів пера. У результаті від пера відмовилися, замінивши його на



більш традиційний телеграфний клопфер, який виглядав приблизно так.

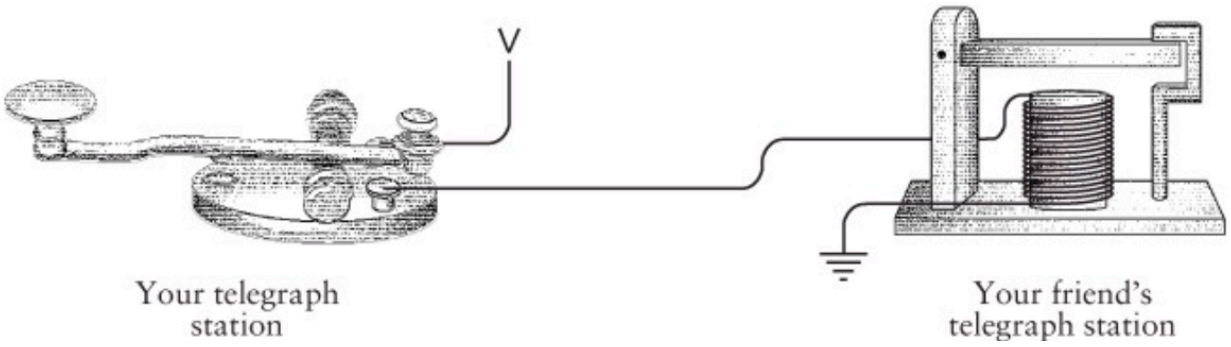
При натисканні телеграфного ключа електромагніт у клопфері опускав рухливу планку і робив характерний "клик". Коли ключ відпускали, планка відскакувала назад і видавала звук "клак". Швидке "клік-клак" відповідало точці, довше "клік-клак" – тире.

Ключ, клопфер, батарею та кілька дротів можна підключити один до одного, як у випадку зі світловим телеграфом, про який ми говорили у попередньому розділі.



Для з'єднання двох телеграфних станцій достатньо одного дроту, другу частину ланцюга замкнемо через землю.

Як і в попередньому розділі, замінимо підключену до землі батарею буквою *V.* Відповідно, повноцінний односпрямований пристрій виглядатиме так.



Для двонаправленого зв'язку нам просто знадобиться ще один ключ і передавач. Приблизно такий пристрій ми й збирали.

Саме з винаходом телеграфу починається епоха сучасних телекомунікацій. Вперше людям вдалося спілкуватися із співрозмовником за межами видимості та чутності, причому набагато оперативніше, ніж при відправці пошти голубами чи кіньми. Набагато цікавіше, що цей винахід застосовував двійковий код. У більш сучасних засобах кабельної та бездротової телекомунікації (телефон, радіо, телевізор) від двійкового коду відмовилися, і знову він увійшов у вжиток з появою комп'ютерів, компакт-дисків, цифрових відеодисків, цифрового супутникового телемовлення та телебачення високого розширення.

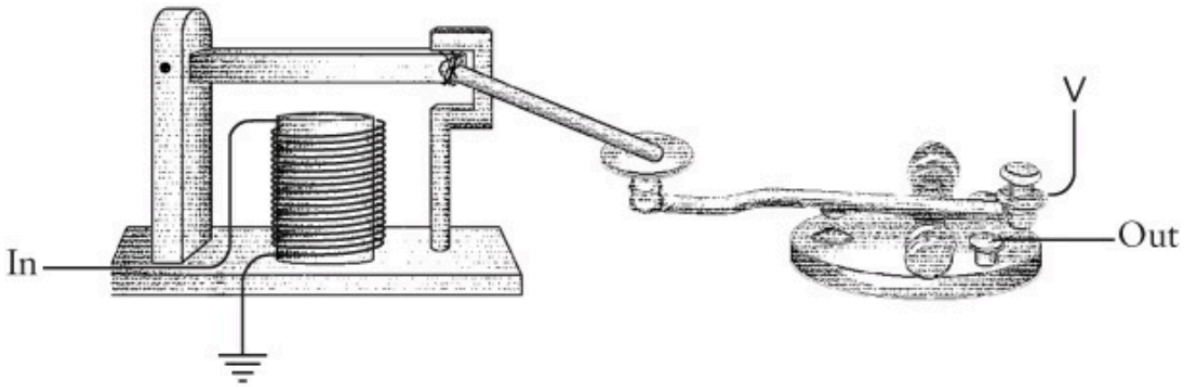
Телеграф Морзе перевершив інші моделі частково тому, що добре працював у перешкодах на лінії. Як правило, провід між ключем і клопфер залишався функціональний. Інші телеграфні системи були настільки невибагливі. Я вже згадував, що велика технічна проблема, пов'язана з телеграфом, полягає у опорі довгих дротів. Хоча на деяких телеграфних лініях використовувалася напруга до 300 вольт, і вони нормально працювали на відстані до 480 кілометрів, необмежено довгих дротів немає.

Рішення сконструювати систему ретрансляторів є очевидним. Через кожні 320 кілометрів можна посадити оператора, дати йому ключ і клопфер і доручити: «Отримав повідомлення передай його наступному».

Тепер уявіть, що телеграфна компанія запросила вас на роботу як такий оператор. Посадили вас десь у глушині між Нью-Йорком і Каліфорнією в хатині, де є тільки стіл та стілець. Через східне вікно в кімнату протягнутий провід, підключений до клопфера. Телеграфний ключ живиться від батареї, а з батареї в західне вікно протягнутий другий провід. Ваше завдання – приймати вхідні повідомлення з Нью-Йорка та пересилати їх до Каліфорнії.

Спочатку ви віддаєте перевагу дочекатися цілісного повідомлення, а потім переслати його. Записуєте літери, що відповідають клацанням клопфера, а коли повідомлення закінчиться – пересилаєте, відстукуючи ключем. Рано чи пізно ви здогадаєтеся, що зручніше транслювати повідомлення прямо в процесі отримання, не записуючи його повністю. Так заощаджується час.

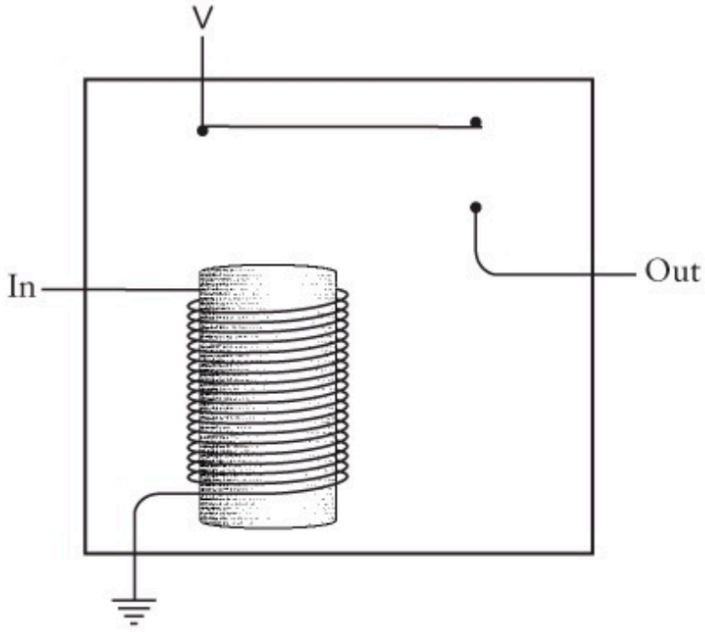
Одного разу ви пересилаєте повідомлення, дивитеся, як скаче вгору-вниз планка клопфера, дивитеся на власні пальці, як ви керуєтеся ключем. Знову дивитеся на клопфер, знову на ключ і усвідомлюєте, що ключ скаче в унісон з клопфером. Виходите на вулицю, берете дощечку, знаходите шнурок і за допомогою дощечки та шнурка зв'язуєте клопфер із ключем.



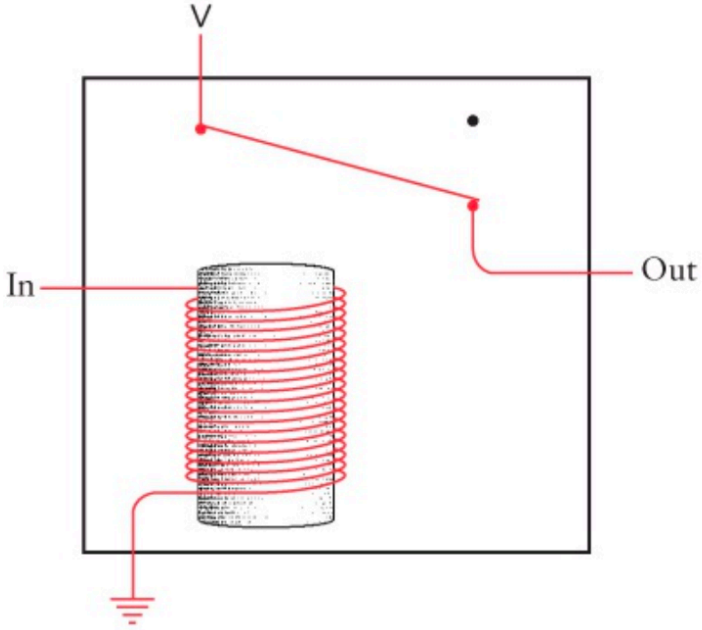
Тепер все працює саме, а ви можете влаштувати вільний вечір і піти порибалити.

Цікаво нафантазовано. Насправді Семюел Морзе ще на ранньому етапі концептуально уявляв собі такий пристрій. Ми винайшли пристрій під назвою *повторювач,* або *реле.* Реле нагадує клопфер, де струм, що входить, живить магніт, що тягне металевий важіль. Однак важіль – це елемент перемикача, що з'єднує батарею з вихідним дротом. У такому разі слабкий вхідний струм «підсилюється», і вихідний струм виходить набагато більшим.

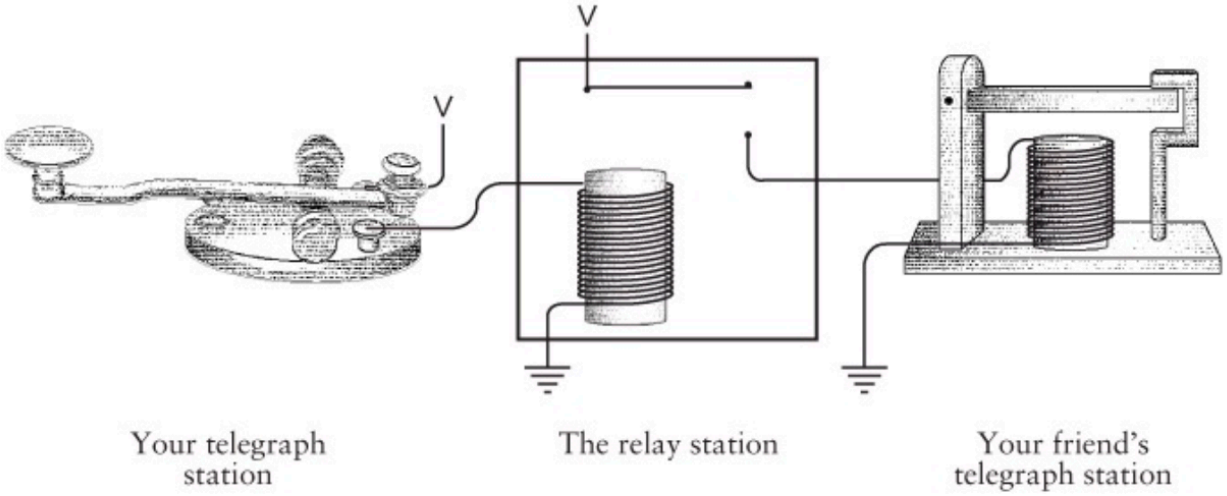
Схема реле має такий вигляд.



Коли вхідний струм активує електромагніт, останній підтягує гнучку металеву стрічку, що діє як перемикач, що пускає вихідний струм.



Отже, телеграфний ключ, клопфер і реле з'єднуються приблизно в такий спосіб.



Реле – чудовий пристрій. Безумовно, це перемикач, але такий, що переводиться зі стану «включений» у стан «вимкнений» і назад не людською рукою, а силою струму. За допомогою такого приладу можна робити дивовижні речі, а телеграф значною мірою дозволяє змоделювати комп'ютер.

Так, реле занадто апетитний винахід, щоб просто залишити його припадати пилом в музеї зв'язку. Заходимо до музею, хапаємо його, засовуємо у внутрішню кишеню піджака і швидко виходимо. Реле нам знадобиться. Однак перш ніж приступити до роботи з ним, треба навчитися рахувати.

Тема 4. Наші десять цифр

Ідея, що мова – просто код, цілком логічна. Багато хто як мінімум намагався вивчити іноземну мову у старших класах, тому важко посперечатися, що кішка в інших мовах може називатися cat, gato, chat, Katze, kot або καττα.

Здається, що числа менш пластичні у культурному контексті. Незалежно від того, якою мовою ми говоримо, практично будь-який співрозмовник на цій планеті, швидше за все, буде записувати числа точно так, як і ми.

1 2 3 4 5 6 7 8 9 10

Чи не тому математику називають універсальною мовою?

Безперечно, числа – найабстрактніший код, з яким доводиться мати справу у повсякденному житті. Вбачаючи число, ми не намагаємося його миттєво з чимось співвіднести.

3

Можна уявити три яблука або три інші предмети, але з тим же успіхом можна дізнатися з контексту, що йдеться про день народження дитини, телевізійний канал, хокейний рахунок, кількість чашок борошна, потрібних для приготування пирога. Вже тому, що цифри настільки абстрактні, нам складніше зрозуміти, що три яблука можна позначити не лише числом 3.

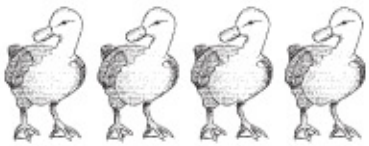
Більша *частина* цього розділу і вся наступна допоможуть переконатися, що таку сама кількість яблук можна позначити і числом 11.

Для початку давайте розлучимося з думкою, що серед 10 є щось особливе. Не дивно, що у більшості цивілізацій склалися системи числення з урахуванням числа 10 (чи 5). З давніх-давен люди рахували на пальцях. Якби людина мала вісім чи дванадцять пальців, то всі лічильні системи були б трохи іншими.

Саме тому система числення *з основою 10,* також іменована *десятковою,* обрана абсолютно довільно. Ми надаємо десятці чисел воістину магічне значення і вигадали для неї особливі назви. Десять днів утворюють декаду, десять десятиліть – століття, десять століть – тисячоліття. Тисяча тисяч – це мільйон, тисяча мільйонів – мільярд. Усі ці числа є ступенями числа 10.

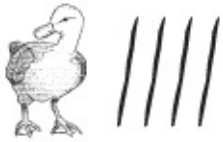
$10^1 = 10$
$10^2 = 100$
$10^3 = 1000$ (тисяча)
$10^4 = 10\ 000$
$10^5 = 100\ 000$
$10^6 = 1\ 000\ 000$ (мільйон)
$10^7 = 10\ 000\ 000$
$10^8 = 100\ 000\ 000$
$10^9 = 1\ 000\ 000\ 000$ (мільярд)

Більшість істориків вважають, що числа спочатку були вигадані для підрахунку предметів, наприклад людей, майна та торгових угод. Якщо в когось було чотири качки, їх можна було



позначити у вигляді чотирьох намальованих качок.

Нарешті людина, чия робота полягала у малюванні качок, подумала: «Навіщо малювати



чотирьох качок? Чи не зобразити одну і позначити, що насправді качок чотири, скажімо, рисками?

Коли знадобилося намалювати 27 качок, рисочки виглядають безглуздо.

Подумалося: «Має бути інший кращий спосіб», – так з'явилася система чисел.

З усіх найдавніших числових систем досі у ході римські цифри. Вони зустрічаються на циферблатах, ними вибивають дати на пам'ятниках, нумерують деякі сторінки в книгах, використовують при підрахунку деяких елементів і що найбільше дратує при вказівці інформації про авторські права в кінофільмах. (Іноді щоб відповісти на питання, в якому році було знято фільм, потрібно блискавично розшифрувати якісь MCMLIII у хвості титрів.)

Двадцять сім качок римськими цифрами буде так.

Принцип досить простий: *X* означає 10 рисок, *V* – п'ять.

Ось римські цифри, що збереглися до наших днів.

I V X L C D M

I – це одиниця; вона схожа на риску або один піднятий палець. *V* – це п'ятірка; можливо, цим символом позначалася долоня. З двох *V* складається *X,* тобто десятка.

L – це п'ятдесят, *C* – літера, з якої починається латинське centum, – сто, *D* – п'ять сотень, *M* – перша літера у слові mille – тисяча.

Хоча ми, можливо, з цим не погодимося, але протягом століть вважалося, що римські цифри зручні для складання та віднімання, саме тому вони так довго використовувалися в Європі під час бухгалтерії. Справді, при додаванні двох римських чисел просто виписуються поруч усі символи з цих двох чисел, та був застосовується лише кілька правил: п'ять *I* утворюють *V,* дві *V – X,* п'ять *X – L* тощо.

Складно множити та ділити числа, записані римськими цифрами. Багато інших ранніх числових систем (наприклад, грецька) аналогічно не підходять для складних математичних дій. Стародавні греки розробили чудову геометрію, яка досі практично без змін викладається в школах, але чи така відома давньогрецька алгебра?

Цифри, якими ми користуємось сьогодні, називаються індо-арабськими. Вони виникли в Індії, але були занесені до Європи арабськими математиками. Особливо прославився перський математик Мухаммад ібн Муса аль-Хорезмі (від імені якого походить слово «*алгоритм»)*, який написав близько 825 року книгу з алгебри, де користувався індійськими цифрами. Ця книга була переведена на латину близько 1120 року, дуже вплинула на Європу і стимулювала перехід з римських цифр на сучасні.

Індо-арабська система чисел відрізнялася від більш ранніх.

• Індо-арабська система називається *позиційною,* тобто будь-яка цифра може позначати в ній різну кількість залежно від того, в якій частині числа стоїть. Положення цифри у числі щонайменше (і навіть більше) важливо, ніж значення самої цифри. І в 100, і в 1 000 000 всього по одній одиниці, але всім відомо, що мільйон набагато більше сотні.

• Практично у всіх ранніх системах числення було щось, чого *немає* в індо-арабській системі, а саме: окремий символ для позначення десятки. У нашій системі числення такий символ *відсутній.*

• З іншого боку, практично у всіх ранніх числових системах не було дещо, що є в індо-арабській системі і, по суті, важливіше, ніж символ десятки, – символ нуля.

Так, нуль. Скромний нуль, безсумнівно, один із найважливіших винаходів в історії чисел та математики. Він забезпечує позиційний запис, оскільки дозволяє відрізнити 25 від 205 і від 250. Нуль спрощує багато математичних дій, незручні в непозиційних системах, особливо множення і поділ.

Уся структура індо-арабських чисел прояснюється, якщо звернути увагу, як ми їх вимовляємо. Наприклад, 4825: «Чотири тисячі вісімсот двадцять п'ять». Це означає:

чотири тисячі,

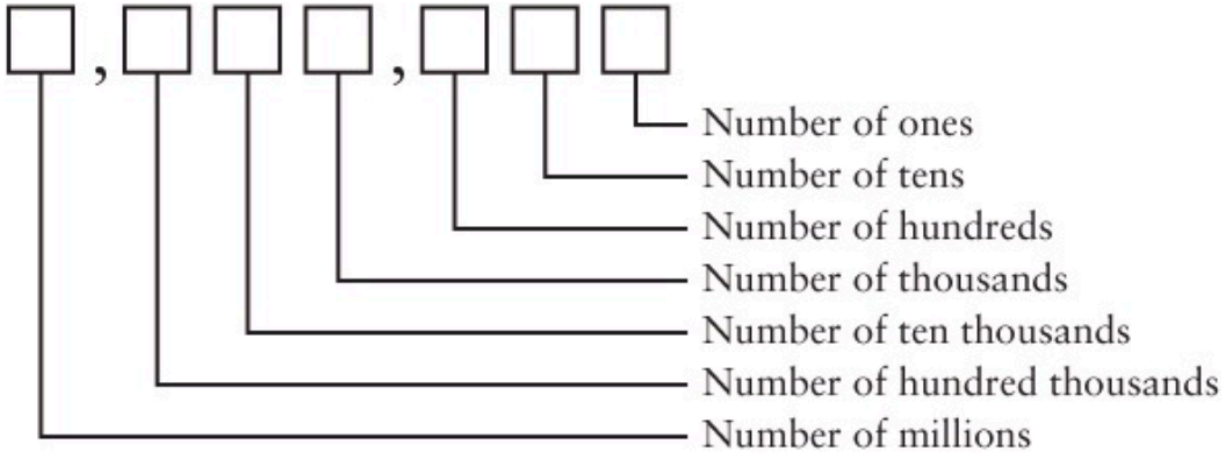
вісім сотень,
два десятки
та ще п'ять.

Або можна розкласти це число на компоненти, наприклад:
4825=4000+800+20+5.
Або ще дрібніше, ось так:
4825 = 4 × 1000+
8 × 100 +
2 × 10 +
5×1.

Або, скориставшись ступенями десятки, записати таке:
$4825 = 4 \times 10^3 +$
$8 \times 10^2 +$
$2 \times 10^1 +$
5×10^0.

Нагадую: будь-яке число в ступені 0 дорівнює одиниці.

Кожна позиція у багатозначному числі має певне значення, як показано на наступній схемі. У семи віконцях можна записати будь-яке число від 0 до 9999999.



Кожна позиція відповідає ступеню десятки. Спеціального символу для десятки не потрібно, оскільки 1 просто ставиться в потрібну позицію, а 0 використовується як символ заповнювача.

Найпрекрасніше в цьому випадку в тому, що дробові величини, що позначаються цифрами після десяткової коми, підпорядковуються тій же закономірності. Число 42 705,684 дорівнює:
4 × 10 000 +
2 × 1000+
7 × 100 +
0 × 10 +
5 × 1 +
6 ÷ 10 +
8 ÷ 100 +
4 ÷ 1000.

Це число можна записати і без поділу:
4 × 10 000 +
2 × 1000+
7 × 100 +
0 × 10 +
5 × 1 +
6 × 0,1 +
8 × 0,01 +
4×0,001.

Або за допомогою ступенів десятки:

$4 \times 10^4 +$
$2 \times 10^3 +$
$7 \times 10^2 +$
$0 \times 10^1 +$
$5 \times 10^0 +$
$6 \times 10^{-1} +$
$8 \times 10^{-2} +$
4×10^{-3}.

Зверніть увагу: спочатку ступінь сягає нуля, а потім отримує негативні значення.

Відомо, що 3 плюс 4 дорівнює 7. Аналогічно 30 плюс 40 дорівнює 70, 300 плюс 400 і 700 і 3000 плюс 4000 і 7000. У цьому і полягає краса індо-арабської системи. Складаючи скільки завгодно довгі десяткові числа, ми фактично вирішуємо це завдання поетапно. На кожному етапі ми лише складаємо однозначні числа. Саме тому хтось давно змушував вас запам'ятовувати таблицю

+	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	10
2	2	3	4	5	6	7	8	9	10	11
3	3	4	5	6	7	8	9	10	11	12
4	4	5	6	7	8	9	10	11	12	13
5	5	6	7	8	9	10	11	12	13	14
6	6	7	8	9	10	11	12	13	14	15
7	7	8	9	10	11	12	13	14	15	16
8	8	9	10	11	12	13	14	15	16	17
9	9	10	11	12	13	14	15	16	17	18

додавання.

Знайдіть у верхньому ряду та у лівому стовпці два числа, які хочете додати. Тримуйтесь від них прямо до центру, поки лінії не перетнуться, і отримайте суму. Наприклад, 4 плюс 6 дорівнює 10.

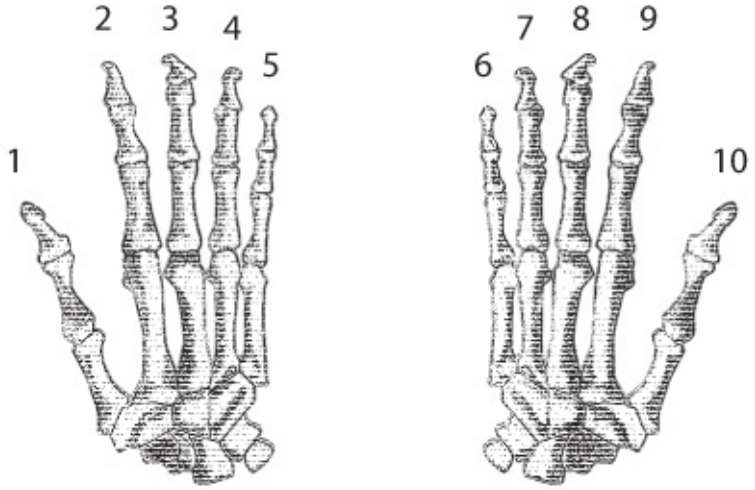
Аналогічно, якщо потрібно перемножити два десяткові числа, виконується складніша процедура, яка проте підрозділяється на дрібні етапи, пов'язані з перемноженням однозначних десяткових чисел. Пам'ятайте, у початковій школі ми мали вивчати і таблицю множення.

×	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9
2	0	2	4	6	8	10	12	14	16	18
3	0	3	6	9	12	15	18	21	24	27
4	0	4	8	12	16	20	24	28	32	36
5	0	5	10	15	20	25	30	35	40	45
6	0	6	12	18	24	30	36	42	48	54
7	0	7	14	21	28	35	42	49	56	63
8	0	8	16	24	32	40	48	56	64	72
9	0	9	18	27	36	45	54	63	72	81

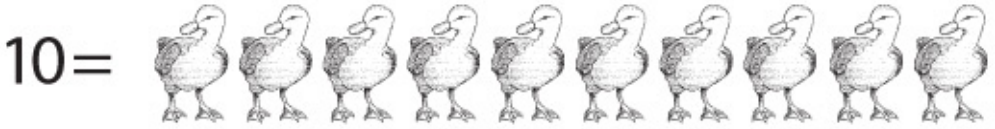
Головна краса позиційної нотації не в тому, як добре вона працює, а в тому, наскільки добре вона застосовна в системах числення, заснованих не на десятці. Наша система числення комусь може здатися незручною. Наприклад, у більшості героїв-мультяшок всього по чотири пальці на руці (або на лапі), тому їм було б зручніше користуватися системою з основою 8. Досить цікаво наступне: більша частина правил, відомих нам за десятковою системою, цілком застосовна *і* в вісімковій.

Тема 5. Альтернативи десятці

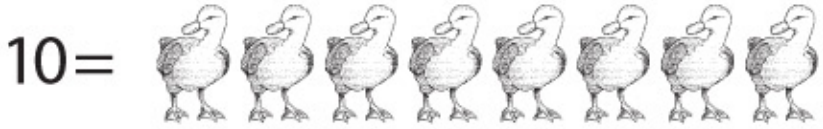
Число 10 – виключно важливе для людини. У більшості з нас по десять пальців на руках і на ногах, і ми, звичайно, вважаємо за краще, щоб і тих, і інших було по десять. Оскільки на пальцях зручно рахувати, людина побудувала всю систему числення на основі числа 10.

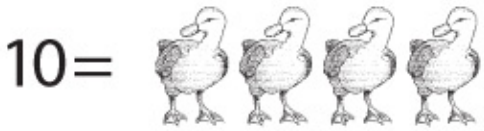


Як згадувалося в попередньому розділі, така система називається «*система з основою 10»,* або «*Десяткова».* Вона здається нам настільки природною, що спочатку важко навіть знайти альтернативу. Дійсно, коли бачимо число 10, нас тягне уявити, що воно означає, наприклад, десять качок.



Єдина причина, через яку виникає така асоціація, у тому, що качок стільки ж, скільки й пальців у нас на руках. Якби людина мала іншу кількість пальців, то й вважали б ми по-іншому, і число 10 означало б щось інше. Наприклад, число 10 може вказувати і таку кількість качок.





Або таку.

Або навіть таку.

Як тільки ми зрозуміємо, в якому випадку 10 означає двох качок, можна буде починати розмову про подання чисел під час роботи з перемикачами, проводами, лампочками та реле (далі – і з комп'ютерами).

Що, якби люди мали всього по чотири пальці на руці, як у героїв з мультиків? Ймовірно, нам би навіть не спало на думку розробляти десяткову систему числення. Навпаки, ми вважали б нормальним, природним, розумним, неминучим, незаперечним і безперечно вірним побудувати систему числення з основою 8. Вона називалася б не *десятковою,* а *вісімковою,* або *системою з основою 8.*

Якби наша система числення була побудована на основі 8, то цей символ нам би не був потрібний:

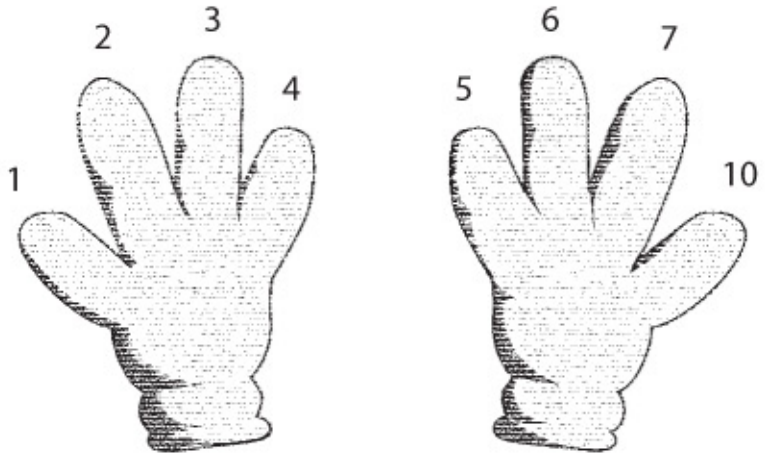
9.

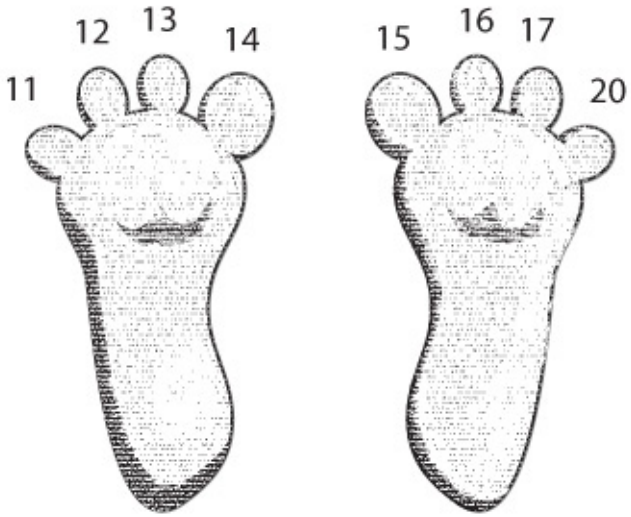
Покажіть цей символ герою з мультиків, і він запитає: «Що це? Навіщо це потрібно?» Якщо замислитись, то і без цього символу можна обійтися:

8.

У десятковій системі числення немає спеціального символу для десятки, відповідно у вісімковій системі числення його немає для вісімки.

У десятковій системі числення ми рахуємо: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, а потім 10. У вісімковій системі рахуємо: 0, 1, 2, 3, 4, 5, 6, 7, а потім що? Цифри скінчилися. Залишається лише 10, і це правильна відповідь. У вісімковій системі за 7 слідує 10. Але в такому випадку 10 відповідає не десяти пальцям, які є на двох руках у людини. У вісімковій системі 10 – це кількість пальців у героїв з мультиків.





Давайте рахувати далі на чотирипалих ступнях.

Маючи справу з іншими системами числення, крім десяткової, можна не плутатися, якщо називати число 10 *один-нуль.* Аналогічно 13 буде "*один-три",* а 20 – "*два-нуль".* Щоб *взагалі* обійтися без плутанини, можна говорити «*два-нуль із основою вісім»* або «*два-нуль вісімкових».*

Навіть коли у нас закінчаться пальці на руках і ногах, можна й надалі рахувати у вісімковій системі. В принципі, процес не відрізняється від рахунку в десятковій системі, просто ми пропускаємо всі числа, в яких є 8 або 9. Звичайно, конкретні числа позначають інші величини.

0, 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21, 22, 23, 24, 25, 26, 27, 30, 31, 32, 33, 34, 35, 36, 37, 40, 41, 42, 43, 44, 45, 46, 47, 50, 51, 52, 53, 54, 55, 56, 57, 60, 6 62, 63, 64, 65, 66, 67, 70, 71, 72, 73, 74, 75, 76, 77, 100…

Останнє число називається "*один-нуль-нуль".* Це загальна кількість пальців героя мультиків, помножена сама на себе.

При записі десяткових і вісімкових чисел можна уникнути плутанини, записуючи всі числа з нижніми індексами, що позначають належність до тієї чи іншої системи числення. Нижній індекс ДЕСЯТЬ означає «основа десять», тобто десяткову систему, а нижній індекс вісім – «основа вісім», або вісімкову систему.

Отже, Білосніжка зустріла 7 десять, або 7 вісім, гномів.
У мультяшок по 8 десять, або 10 вісім, пальців на руці.
Бетховен написав 9 десять, або 11 вісім, симфоній.
У людини 10 десять, або 12 вісім, пальців на руках.
У році 12 десять, або 14 вісім місяців.
У двох тижнях 14 десять, або 16 вісім днів.
Паспорт видають у 16 десять, або 20 вісім, років.
На добу 24 десять, або 30 восім, годин.
У латиниці 26 десять, або 32 вісім, букв.
В англійській кварті 907 десять, або 1134, грамів.
У покерній колоді 52 десять, або 64 восім, карт.
Найвідоміша адреса по Сансет-Стріп – 77 десять, або 115 вісім.
Довжина поля для американського футболу – 91 десять, або 131 вісім, метрів.
На старті жіночого одиночного заліку в турнірі Вімблдону – 128 десять, або 200 вісім, учасниць.
Площа Мемфісу дорівнює 640 десять, або 1000 вісім, квадратних кілометрів.

Зверніть увагу: у списку є кілька *круглих* вісімкових чисел. Круглим називається число, яке закінчується одним чи кількома нулями. Якщо десяткове число закінчується двома нулями, значить, воно кратне $100_{десять}$, а $100_{десять}$ це $10_{десять}$ помножене на $10_{дncять}$. У вісімковій системі два нулі в кінці числа означають, що число кратне $100_{вісім}$, тобто $10_{вісім}$ помножити на $10_{вісім}$ (або $8_{десять}$ помножити на $8_{десять}$, що дорівнює $64_{десять}$).

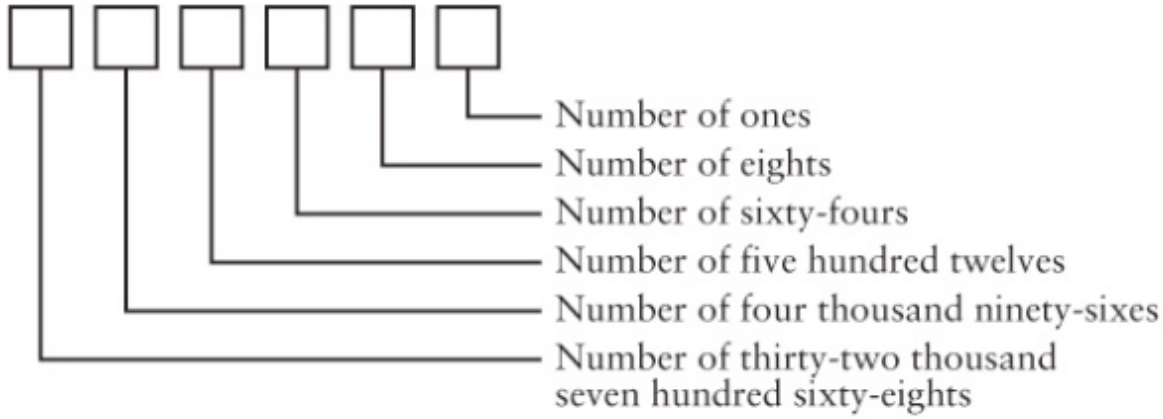
Можливо, ви також помітили, що такі круглі вісімкові числа, як $100_{вісім}$, $200_{вісім}$ і $400_{вісім}$, у десятковій системі відповідають $64_{десять}$, $128_{десять}$ і $256_{десять}$, і всі ці десяткові числа – степені двійки. Це логічно. Наприклад, число $400_{вісім}$, дорівнює $4_{вісім}$ помножити на $10_{вісім}$ і помножити на $10_{вісім}$, і все це – степені двійки. Щоразу при множенні степеня двійки на степінь двійки ми отримуємо ще один степінь двійки.

У наступній таблиці подано деякі степені двійки в десятковому та вісімковому поданні.

Power of	Two Decimal	Octal
2^0	1	1
2^1	2	2
2^2	4	4
2^3	8	10
2^4	16	20
2^5	32	40
2^6	64	100
2^7	128	200
2^8	256	400
2^9	512	1000
2^{10}	1024	2000
2^{11}	2048	4000
2^{12}	4096	10000

Круглі числа з правого стовпця нагадують, що системи числення, які відрізняються від десяткової, зручні для роботи з двійковими кодами.

Структурно вісімкова система аналогічна десятковій. Відмінності лише у деталях. Наприклад, кожна позиція у вісімковому числі – це цифра, помножена на степінь вісімки.



Отже, вісімкове число $3725_{\text{вісім}}$ можна розбити:
$3725_{\text{вісім}} = 3000_{\text{вісім}} + 700_{\text{вісім}} + 20_{\text{вісім}} + 5_{\text{вісім.}}$

Цю послідовність можна переписати трохи інакше. Наприклад, за допомогою степенів вісімки в їх десятковому поданні:

$3725_{\text{вісім}} = 3 \times 512_{\text{десять}} +$
$7 \times 64_{\text{десять}} +$
$2 \times 8_{\text{десять}} +$
$5 \times 1.$

Те ж саме, записане за допомогою степенів вісімки у вісімковому поданні:
$3725_{\text{вісім}} = 3 \times 1000_{\text{вісім}} +$

$7 \times 100_{\text{вісім}} +$
$2 \times 10_{\text{вісім}} +$
$5 \times 1.$

А можна зробити так:
$3725_{\text{вісім}} = 3 \times 8^3 +$
$7 \times 8^2 +$
$2 \times 8^1 +$
$5 \times 8^0.$

Якщо виконати ці розрахунки в десятковій системі, вийде $2005_{\text{десять.}}$ Таким чином вісімкові числа перетворюються на десяткові.

Вісімкові числа складаються і перемножуються точно як десяткові. Різниця в тому, що таблиці множення та додавання для вісімкових чисел будуються інакші. Ось таблиця додавання вісімкових чисел.

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	10
2	2	3	4	5	6	7	10	11
3	3	4	5	6	7	10	11	12
4	4	5	6	7	10	11	12	13
5	5	6	7	10	11	12	13	14
6	6	7	10	11	12	13	14	15
7	7	10	11	12	13	14	15	16

Наприклад, $5_{\text{вісім}} + 7_{\text{вісім}} = 14_{\text{вісім,}}$ тобто вісімкові числа можна додавати в стовпчик.

$$\begin{array}{r} 135 \\ + \quad 643 \\ \hline 1000 \end{array}$$

Починаємо праворуч: 5 плюс 3 дорівнює 10, 0 пишемо, 1 в пам’яті; 1 плюс 3 плюс 4 дорівнює 10, 0 пишемо, 1 в пам’яті; 1 плюс 1 плюс 6 дорівнює 10.

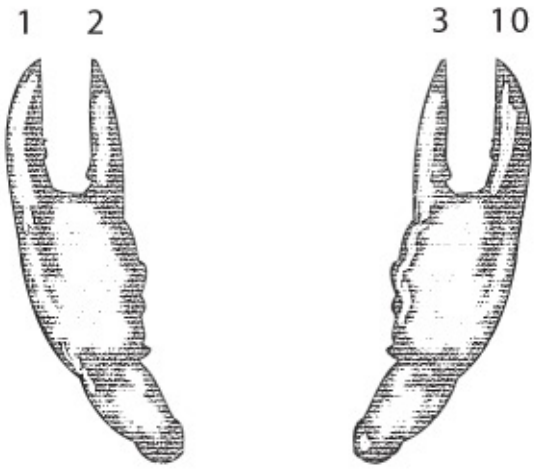
Аналогічно двічі два і у вісімковій системі дорівнює чотирьом. Але тричі три не дорівнює дев'яти. А як? Тричі три дорівнює $11_{\text{вісім,}}$ це стільки ж, скільки і $9_{\text{десять.}}$ Далі повністю наведено вісімкову таблицю множення.

×	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	10	12	14	16
3	0	3	6	11	14	17	22	25
4	0	4	10	14	20	24	30	34
5	0	5	12	17	24	31	36	43
6	0	6	14	22	30	36	44	52
7	0	7	16	25	34	43	52	61

Тут у нас 4 × 6 дорівнює $30_{\text{вісім,}}$ але $30_{\text{вісім}}$ дорівнює $24_{\text{десять,}}$ тобто 4 × 6 в десятковій системі.

Вісімкова система числення така ж повноцінна, як і десяткова.

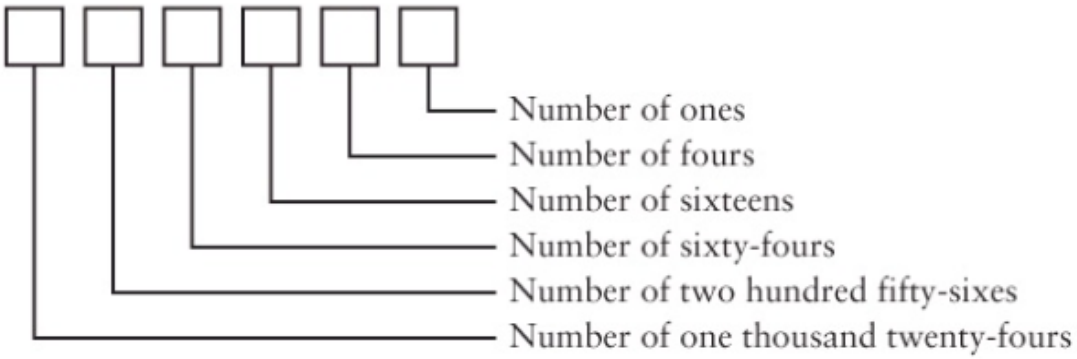
Ми розробили систему числення для героїв з мультиків. Тепер давайте створимо таку саму систему для омарів. Омари не мають пальців, але на кінчиках передніх лап у них клешні. Омарам



підійде четвіркова система числення з основою чотири.

Ось як рахують у четвірковій системі: *0, 1, 2, 3, 10, 11, 12, 13, 20, 21, 22, 23, 30, 31, 32, 33, 100, 101, 102, 103, 110 і т д.*

Не будемо докладно зупинятися на четвірковій системі, оскільки ми наближаємося до більш важливого питання. Як бачите, тут кожна позиція в числі відповідає ступеню четвірки.



У четвірковій системі числення число 31 232 можна записати так:
31 232 чотири *= 3 × 256* десять *+*
1 × 64 десять *+*
2 × 16 десять *+*
3 × 4 десять *+*
2 × 1 десять.

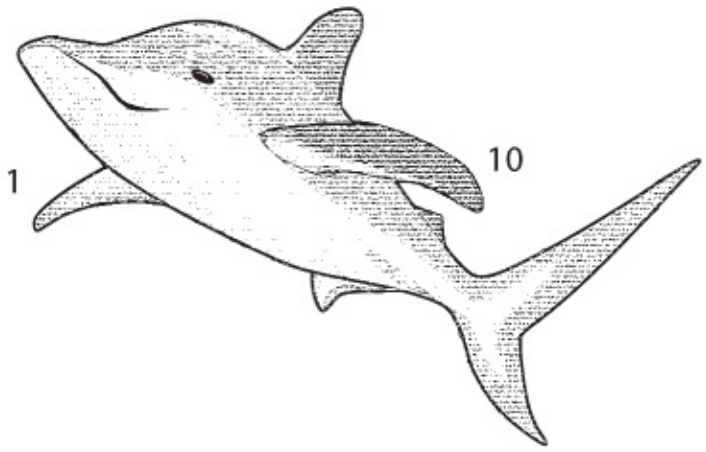
Що рівнозначно запису:
31 232 чотири *= 3 × 10 000* чотири *+*
1 × 1000 чотири *+*
2 × 100 чотири *+*
3 × 10 чотири *+*
2 × 1 чотири.

А це те саме, що й:
31 232 чотири *= 3 × 4* ^{4} *+*
1 × 4 ^{3} *+*
2 × 4 ^{2} *+*
3 × 4 ^{1} *+*
2 × 4 ^{0}.

Якщо ми виконаємо обчислення в десятковій системі числення, то виявимо, що 31 232 чотири – це 878 десять.

Тепер ми зробимо ще один стрибок, цього разу остаточний. Уявіть, що ми дельфіни і можемо використовувати для підрахунку два плавці. В даному випадку ми маємо справу з системою числення з основою 2, або *двійковою,* або, інакше, *бінарною* (від лат. binary – "подвійний", "що складається з двох частин"). Зрозуміло, що у нас буде лише дві цифри: 0 та 1.

З нулем і одиницею мало що можна зробити, і щоб звикнути до двійкових чисел, потрібна практика. Проблема в тому, що одразу закінчуються цифри. Наприклад, на наступному малюнку



показано, як дельфін рахує на плавцях.

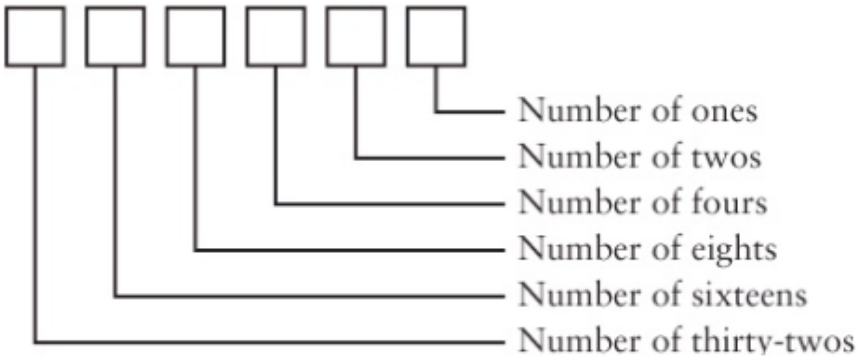
Так, у двійковій системі числення за 1 слідує 10. Це дивно, проте це не повинно дивувати. Незалежно від того, яку систему числення ми використовуємо, щоразу, коли у нас закінчуються окремі цифри, перше двозначне число завжди 10. У двійковій системі числення ми рахуємо:

0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111, 10000, 10001…

Ці числа можуть здатися більшими, але насправді це не так. Швидше, двійкові числа дуже швидко стають *довгими,* а не більшими.

Кількість голів у людей – 1 десять, або 1 два.
Кількість плавців у дельфінів – 2 десять, або 10 два.
Кількість чайних ложок у столовій ложці – 3 десять, або 11 два.
Кількість сторін у квадрата – 4 десять, або 100 два.
Кількість пальців на одній людській руці – 5 десять, або 101 два.
Кількість кінцівок у комах – 6 десять, або 110 два.
Кількість днів у тижні – 7 десять, або 111 два.
Кількість музикантів в октеті – 8 десять, або 1000 два.
Кількість планет у Сонячній системі, включаючи Плутон, – 9 десять, або 1001 два.
Кількість центнерів у тоні – 10 десять, або 1010 два.

У двійковому числі, що складається з великої кількості цифр, позиції знаків відповідають степеню двійки.



Таким чином, щоразу, коли зустрічаємо двійкове число, що складається з одиниці та наступних за нею нулів, ми розуміємо, що це число відповідає будь-якому з степенів двійки. Цей степінь дорівнює кількості нулів у цьому двійковому числі. Ось наша розширена таблиця степенів двійки, яка демонструє таке правило.

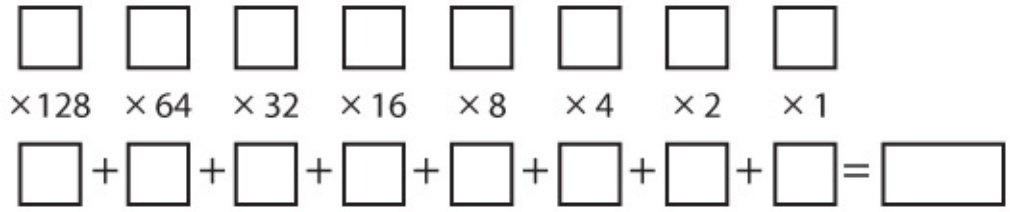
Power of Two	Decimal	Octal	Quaternary	Binary
2^0	1	1	1	1
2^1	2	2	2	10
2^2	4	4	10	100
2^3	8	10	20	1000
2^4	16	20	100	10000
2^5	32	40	200	100000
2^6	64	100	1000	1000000
2^7	128	200	2000	10000000
2^8	256	400	10000	100000000
2^9	512	1000	20000	1000000000
2^{10}	1024	2000	100000	10000000000
2^{11}	2048	4000	200000	100000000000
2^{12}	4096	10000	1000000	1000000000000

Припустимо, у нас є двійкове число 101101011010. Його можна записати так:
101101011010 два = 1 × 2048 десять +
0 × 1024 десять +
1 × 512 десять +
1 × 256 десять +
0 × 128 десять +
1 × 64 десять +
0 × 32 десять +
1 × 16 десять +
1 × 8 десять +
0 × 4 десять +
1 × 2 десять +
0 × 1 десять.

Або:
101101011010 два = 1 × 2 11 +
0 × 2 10 +
1 × 2 9 +
1 × 2 8 +
0 × 2 7 +
1 × 2 6 +
0 × 2 5 +

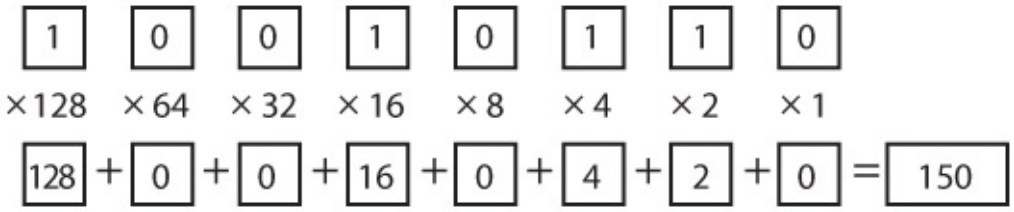
$1 \times 2^4 +$
$1 \times 2^3 +$
$0 \times 2^2 +$
$1 \times 2^1 +$
$0 \times 2^0.$

Якщо просто скласти всі складові в десятковій системі, отримаємо 2048 + 512 + 256 + 64 + 16 + 8 + 2, що становить 2906 десять.

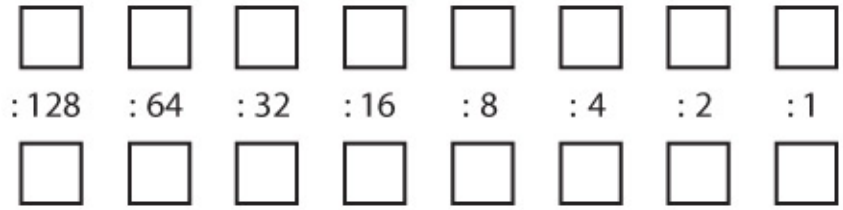


Для легшого перетворення двійкових чисел на десяткові можна використовувати таку схему.

Ця схема дозволяє конвертувати числа до восьми двійкових розрядів; її можна легко розширити. Введіть до восьми цифр у вісім верхніх полів, по одній цифрі на кожен квадрат. Виконайте вісім операцій множення та введіть їх результати у вісім нижніх полів. Додайте числа у цих восьми полях для отримання остаточного результату. Цей приклад демонструє процес знаходження десяткового еквівалента двійкового числа 10010110.



Перетворити десяткові числа від 0 до 255 на двійкові не так просто, проте ви можете



використовувати наступну схему.

Процес перетворення складніший, ніж здається, тому уважно дотримуйтесь вказівок. Помістіть десяткове число (менше або 255) у верхній лівий квадрат. Розділіть це число (ділене) на перший дільник (128), як показано на схемі. Помістіть цілу частину в нижнє поле (лівий нижній квадрат), а залишок від поділу – у правому полі (другий квадрат у верхньому ряду). Цей перший залишок є ділимим, яке братиме участь у наступній операції поділу, де як дільник використовується число 64.

Пам'ятайте, що кожна ціла частина дорівнюватиме або 0, або 1. Якщо ділене менше дільника, то ціла частина від поділу дорівнюватиме 0, а залишок – самому діленому. Якщо ділене більше або дорівнює дільнику, то ціла частина від розподілу дорівнюватиме 1, а залишок – різниці між ділимим і дільником. Ось як конвертується число 150.

150	22	22	22	6	6	2	0
: 128	: 64	: 32	: 16	: 8	: 4	: 2	: 1
1	0	0	1	0	1	1	0

Якщо вам потрібно скласти або перемножити два двійкові числа, ймовірно, буде легше виконати обчислення в двійковій системі, не перетворюючи числа на десяткові. Це має сподобатися. Уявіть, як швидко ви могли б освоїти додавання, якби потрібно було запам'ятати тільки це.

+	**0**	**1**
0	0	1
1	1	10

Давайте за допомогою цієї таблиці додамо два двійкові числа.

$$\begin{array}{r} 1100101 \\ +\ 0110110 \\ \hline 10011011 \end{array}$$

Починаючи з правого стовпця: 1 плюс 0 дорівнює 1. Другий стовпець праворуч: 0 плюс 1 дорівнює 1. Третій стовпець: 1 плюс 1 дорівнює 0, 1 в пам’яті. Четвертий стовпець: 1 (перенесене значення) плюс 0 плюс 0 дорівнює 1. П'ятий стовпець: 0 плюс 1 дорівнює 1. Шостий стовпець: 1 плюс 1 дорівнює 0, 1 в пам’яті. Сьомий стовпець: 1 (перенесене значення) плюс 1 плюс 0 = 10.

Таблиця множення навіть простіша, ніж таблиця додавання, оскільки її можна скласти, використовуючи два базові правила множення: множачи на 0, отримуємо 0, множення на 1 не впливає на вихідне число.

×	**0**	**1**
0	0	0
1	0	1

Ось процес множення числа 13 $_{\text{десять}}$ на число 11 $_{\text{десять}}$ у двійковій системі числення.

$$\begin{array}{r} 1101 \\ \times\ 1011 \\ \hline 1101 \\ 1101 \\ 0000 \\ 1101 \\ \hline 10001111 \end{array}$$

Результат – 143 десять.

Люди, які працюють із двійковими числами, часто передують їх нулями, тобто пишуть нулі зліва від першої 1, наприклад 0011 замість 11. Це зовсім не впливає на значення, а служить виключно для краси. У наступній таблиці наведено перші шістнадцять двійкових чисел та їх десяткові еквіваленти.

Binary	Decimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	10
1011	11

Розгляньмо список двійкових чисел. Зверніть увагу на кожен із чотирьох вертикальних стовпців, що складаються з нулів та одиниць, і зауважте, як ці цифри чергуються в стовпцях зверху вниз:

- у крайньому правому стовпці – 0 та 1;
- у другому стовпці праворуч – два 0 та дві 1;
- у наступному стовпці – чотири 0 та чотири 1;
- у крайньому лівому стовпці – вісім 0 та вісім 1.

У цьому є порядок, чи не так? Дійсно, ви можете легко написати наступні шістнадцять двійкових чисел, просто повторивши перші шістнадцять і додавши 1 на початку.

Binary	Decimal
10000	16
10001	17
10010	18
10011	19
10100	20
10101	21
10110	22
10111	23
11000	24
11001	25
11010	26
11011	27
11100	28
11101	29
11110	30
11111	31

Ось ще один спосіб дивитися на це: при виконанні підрахунку в двійковому форматі крайня цифра справа (також звана молодшим розрядом) почергово набуває значень 0 і 1. Щоразу, коли вона змінюється з 1 на 0, друга цифра справа, наступна за молодшим розрядом, також змінюється або з 0 на 1, або з 1 на 0. Тому щоразу, коли двійкова цифра змінюється з 1 на 0, наступна за нею цифра також змінюється або з 0 на 1, або з 1 на 0.

При записі великих десяткових чисел ми використовуємо коми через кожні три знаки для полегшення сприйняття*. Наприклад, якщо ви побачите число 12000000, ймовірно, доведеться підрахувати кількість цифр, однак, побачивши число 12,000,000, ви відразу зрозумієте, що воно означає 12 мільйонів.

Двійкові числа дуже швидко можуть стати дуже довгими. Наприклад, 12 мільйонів у двійковій системі числення записується так: 101101110001101100000000. Щоб таке число було легше сприймати, кожні чотири двійкові розряди зазвичай поділяються пробілами (1011 0111 0001 1010 00). Далі ми розглянемо стисліший спосіб запису двійкових чисел.

Звівши систему числення до двійкових цифр 0 та 1, ми досягли межі. Далі спрощувати нікуди. Більше того, двійкова система поєднує арифметику з електрикою. У попередніх розділах ми розглядали перемикачі, дроти, лампочки та реле, і будь-який з цих об'єктів може відображати двійкові цифри 0 та 1.

Провід може являти собою двійкову цифру. Якщо ним йде струм, то двійкова цифра дорівнює 1, якщо ні – 0.

Перемикач може являти собою двійкову цифру. Якщо перемикач увімкнено, або замкнуто, то двійкова цифра дорівнює 1, якщо перемикач вимкнений, або розімкнуто, то двійкова цифра – 0.

Лампочка може являти собою двійкову цифру. Якщо лампочка горить, то двійкова цифра дорівнює 1, якщо ні – 0.

Телеграфне реле може являти собою двійкову цифру. Якщо реле замкнуте, двійкова цифра дорівнює 1, якщо розімкнуто – 0.

Двійкові цифри безпосередньо стосуються комп'ютерів.

Приблизно 1948 року американський математик Джон Тьюкі (1915-2000) усвідомив, що у майбутньому словосполучення «двійкова цифра» (binary digit), мабуть, набуде значно більшого значення – з поширенням комп'ютерів. Він вирішив створити нове, коротше слово, щоб замінити ці громіздкі п'ять складів, і розглядав такі варіанти, як bigit і binit, але зупинився на короткому, простому, елегантному і чудовому слові bit («біт»).

* Кома для поділу розрядів числа використовується переважно в англомовній нотації; дробова частина числа у такому разі відокремлюється крапкою. В Україні прийнято розділяти розряди пробілами, а дробову частину відокремлювати комою.

Тема 6. Логіка та перемикачі

Що таке істина? Аристотель вважав, що вона пов'язана з логікою. Збірка його творів під назвою «Органон» (датована IV століттям до н. е.) – найраніший твір, де докладно висвітлюється ця тема. Для давніх греків логіка – засіб аналізу мови з метою знаходження істини, тому вона вважалася формою філософії. Основа логіки Аристотеля – *силогізм.* Найвідоміший силогізм (який фактично відсутній у роботах Аристотеля) формулюється так:

Усі люди смертні;
Сократ – людина;
отже, Сократ смертний.

У силогізмі з двох передумов, що вважаються істинними, виводиться висновок.

Смертність Сократа може бути досить очевидною, проте існує безліч різноманітних силогізмів. Розглянемо наступні дві передумови, які запропонував математик XIX століття Чарльз Доджсон, відомий як Льюїс Керрол:

Усі філософи логічні;
нелогічна людина завжди вперта.

У даному випадку висновок не очевидний. Він формулюється так: «Деякі вперті люди не є філософами». Зверніть увагу на несподіване і невизначеність, що привносить невизначеність слово «деякі».

Протягом двох тисяч років математики боролися з логікою Аристотеля, намагаючись приборкати її за допомогою математичних символів і операторів. До XIX століття найближче до вирішення цього завдання вдалося підійти лише Готфріду Вільгельму фон Лейбніцу (1646-1716), який займався логікою в молодості, а потім зацікавився іншими речами, наприклад одночасно з Ісааком Ньютоном розробив диференціальне числення (незалежно від нього)*. Потім на сцену вийшов Джордж Буль.

Джордж Буль народився в 1815 році в Англії в соціумі, де його шанси на успіх були дуже малі. Оскільки він був сином чоботаря та колишньої покоївки, його перспективи не сильно відрізнялися від перспектив його предків через жорстку класову ієрархію британського суспільства. Однак завдяки своєму допитливому розуму та батькові, який цікавився наукою, математикою та літературою, молодий Джордж здобув освіту, яка, як правило, є привілеєм хлопчиків з вищих класів суспільства. Він вивчав латину, грецьку мову та математику. Ранні роботи Буля з математики дозволили йому в 1849 стати першим професором математики в Королівському коледжі Корка.

Декілька математиків у середині 1800-х років працювали над формальним визначенням логіки (серед них особливо виділявся Огастес де Морган). Однак саме Буль здійснив справжній концептуальний прорив: спочатку в короткій книзі «Математичний аналіз логіки, або Нарис обчислення дедуктивних висновків» (1847), потім у більш об'ємному та амбітному творі «Дослідження законів мислення», на яких засновані математичні теорії логіки та ймовірностей (1854), яке коротко також називається «Дослідження законів мислення». Буль помер у 1864 році у віці 49 років від пневмонії, яку він підхопив, потрапивши під дощ дорогою на лекцію.

Назва книги Буля 1854 говорить про постановку амбітного завдання: оскільки мозок розумної людини мислить, використовуючи логіку, то, знайшовши спосіб математичного уявлення логіки, ми отримаємо математичний опис того, як працює мозок. Зрозуміло, в наш час така *думка* здається досить наївною (або просто вона значно випереджає свій час).

Винайдена Булем алгебра дуже схожа на звичайну. У звичайній алгебрі *операнди* (зазвичай літери) позначають цифри, а *оператори* (наприклад, «+» і «×») вказують, як ці числа мають об'єднуватися. Як правило, ми використовуємо звичайну алгебру для вирішення таких завдань: Аня має три яблука. У Бетті вдвічі більше яблук, ніж у Ані. У Кармен на п'ять яблук більше, ніж у Бетті. У Дейрдри втричі більше яблук, ніж у Кармен. Скільки яблук у Дейрдри?

Щоб вирішити це завдання, спочатку перетворимо її на арифметичні вирази, використовуючи чотири літери, що відповідають кількості яблук, що є у кожної з чотирьох жінок:

A = 3;

$$Б = 2 \times А;$$
$$К = Б + 5;$$
$$Д = 3 \times К.$$

Ми можемо об'єднати ці чотири вирази в одне шляхом підстановки, а потім уже виконати операції додавання та множення:

$$Д = 3 \times К;$$
$$Д = 3 \times (Б + 5);$$
$$Д = 3 \times ((2 \times А) + 5);$$
$$Д = 3 \times ((2 \times 3) + 5);$$
$$Д = 33.$$

Маючи справу зі звичайною алгеброю, ми дотримуємося певних правил. Ці правила настільки вкоренилися в практиці, що ми більше не думаємо про них як про правила і навіть іноді забуваємо їх назви. Проте будь-яка форма математики підкоряється їм.

Перше правило у тому, що додавання і множення є *комутативними* операціями. Це означає, що можна міняти операнди місцями в обох частинах виразу, не впливаючи на результат:

$$А + В = В + А;$$
$$А \times В = В \times А.$$

Навпаки, операції віднімання та поділу *не є* комутативними.

Додавання та множення – *асоціативні* операції, тобто:

$$А + (В + С) = (А + В) + С;$$
$$А \times (В \times С) = (А \times В) \times С.$$

Нарешті, множення *дистрибутивне* по відношенню до додавання:

$$А \times (В + С) = (А \times В) + (А \times С).$$

Іншою характеристикою звичайної алгебри є те, що вона завжди оперує числами, наприклад, кілограмами сиру, кількістю качок, відстанню, яку проїхав поїзд, або віком членів сім'ї. Геній Буля зробив алгебру абстрактнішою, відокремивши її від концепції числа. У булевій алгебрі (саме таку назву отримала алгебра Буля) операнди відносяться не до числа, а до *класів.* Клас – це просто набір предметів, який надалі став *безліччю.*

Поговоримо про кішок. Кішки можуть бути чоловічої та жіночої статі. Для зручності безліч котів позначатимемо літерою *М,* а безліч кішок – *Ж.* Майте на увазі, що ці два символи не відповідають кількості кішок. Кількість котів і кішок може змінюватися з часом у міру того, як нові особини народжуються, а старі, на жаль, йдуть у інший світ. Ці букви позначають класи кішок із специфічними характеристиками. Говорячи про котів, ми можемо просто сказати "М".

Ми також можемо використовувати інші літери для позначення забарвлення кішок: літерою *Р* описати безліч рудих, літерою *Ч* – безліч чорних, літерою *Б* – безліч білих, а буквою *Д* – безліч кішок усіх «інших» кольорів, тобто кішок, які не входять до класів Р, Ч чи Б.

Нарешті (принаймні у нашому прикладі) кішки можуть бути або стерилізованими, або ні. Давайте позначимо буквою *С* безліч стерилізованих кішок, а буквою *Н* – безліч нестерилізованих.

У звичайній (числовій) алгебрі оператори «+» і «×» використовуються для позначення операцій додавання та множення. У булевій алгебрі застосовуються такі самі символи «+» і «×», що може викликати плутанину. Всім відомо, як додавати і множити числа у звичайній алгебрі, але як можна додавати та множити *класи*?

Справа в тому, що в булевій алгебрі ми фактично нічого не додаємо і не множимо. Натомість символи «+» і «×» означають щось зовсім інше.

У булевій алгебрі символ «+» – це *об'єднання* двох класів, яке передбачає об'єднання всього, що відноситься до першого класу, з усім, що належить до другого. Наприклад, вираз Ч + Б означає безліч всіх кішок чорного та білого забарвлення.

Символ «×» – це *перетин* двох класів, тобто перетин безлічі елементів, що належать як першому, так і другому класу. Наприклад, Ж × Р – клас усіх кішок жіночої статі та рудого забарвлення. Як і в звичайній алгебрі, ми можемо написати Ж × Р у вигляді Ж і Р або просто ЖР

(саме так вважав за краще писати сам Буль). Ви можете розглядати ці дві літери як два прикметники, що описують безліч «рудих кішок жіночої статі».

Щоб не сплутати звичайну алгебру з булевою, замість символів «+» і «×» для позначення об'єднання та перетину класів іноді використовуються символи U та ∩.

Однак звільняючий вплив Буля на математику частково полягав у тому, щоб зробити використання знайомих операторів абстрактнішим, тому, наслідуючи його приклад, я вирішив не вводити нові символи.

Комутативні, асоціативні та дистрибутивні правила залишаються справедливими у булевій алгебрі. Більше того, тут оператор «+» є дистрибутивним по відношенню до оператора «×», чого не можна сказати про звичайну алгебру:

$$Б + (Ч \times Ж) = (Б + Ч) \times (Б + Ж).$$

Об'єднання білих та чорних кішок-самок рівнозначне перетину двох об'єднань: білих та чорних кішок, а також білих кішок та кішок-самок. Це складно зрозуміти, але саме так і влаштовано.

Булевій алгебрі необхідні ще два символи. Вони схожі на числа, але ними не є, оскільки іноді з ними поводяться не так, як з числами. Символ 1 означає багато всіх речей, про які ми говоримо. У цьому прикладі 1 – це безліч всіх кішок:

$$М+Ж=1.$$

Значить, багато кішок містить самців і самок. Так само воно включає всіх кішок рудого, чорного, білого та інших забарвлень:

$$Р+Ч+Б+Д=1.$$

Крім того, безліч всіх кішок можна отримати і так:

$$З + Н = 1.$$

Символ 1 може використовуватися зі знаком мінус, щоб вказати на безліч усіх речей, *що виключає* певну підмножину, наприклад:

$$1 - М.$$

Як бачите, це безліч усіх кішок, окрім самців. Безліч всіх кішок, що виключає всіх самців, відповідає безлічі кішок жіночої статі:

$$1 - М = Ж.$$

Інший необхідний символ – 0, а в булевій алгебрі 0 означає порожню множину, яка нічого не містить. Порожня множина – результат перетину двох взаємовиключних множин, наприклад, безліч кішок-гермафродитів:

$$Ж \times М = 0.$$

Зверніть увагу: символи 1 і 0 іноді працюють однаково в булевій та у звичайній алгебрі. Наприклад, перетин безлічі всіх кішок і кішок жіночої статі відповідає безлічі кішок-самок:

$$1 \times Ж = Ж.$$

Перетин порожньої множини і безлічі кішок-самок представляє порожню множину:

$$0 \times Ж = 0.$$

Об'єднання порожньої множини і множини всіх кішок-самок – це безліч кішок-самок:

$$0 + Ж = Ж.$$

Однак іноді результати в булевій та у звичайній алгебрі відрізняються. Наприклад, об'єднання всіх кішок і кішок-самок відповідає безлічі всіх кішок:

$$1 + Ж = 1.$$

Це не має сенсу у звичайній алгебрі.

Оскільки Ж – множина всіх кішок-самок, а 1-Ж – множина всіх кішок, які не є самками, об'єднання цих двох множин відповідає 1:

Ж + (1 - Ж) = Ж + М = 1.

Перетин двох множин відповідає 0:

Ж × (1 − Ж) = 0.

З історичної точки зору це формулювання – важлива віха в логіці, звана *законом протиріччя,* який говорить, що щось не може одночасно бути собою і своєю протилежністю.

Де булева алгебра дійсно відрізняється від звичайної, то це в наступному виразі:

Ж × Ж = Ж.

Перетин безлічі кішок-самок і безлічі кішок-самок, як і раніше, безліч кішок-самок. Цей вислів має сенс у булевій алгебрі. Однак він неправильний, якби літера *Ж* означала число. Буль вважав, що вираз $Х^2$ = Х є єдиним виразом, що відрізняє його алгебру від звичайної. Ось ще один булев вираз, який виглядає дивно з точки зору звичайної алгебри:

Ж + Ж = Ж.

Об'єднання безлічі кішок-самок і безлічі кішок-самок, як і раніше, є безліччю кішок-самок.

Булева алгебра надає математичний метод на вирішення силогізму Аристотеля. Давайте розглянемо перші дві його частини:

Усі люди смертні;

Сократ – людина.

Буквою *Л* ми позначимо безліч всіх людей, літерою *Х* – безліч смертних істот, а буквою *С* – безліч Сократів. Що означає вираз «всі люди смертні»? Перетин безлічі всіх людей і безлічі всіх смертних істот – це безліч усіх людей:

Л × Х = Л.

Вираз Л × Х = Х був би неправильним, оскільки багато всіх смертних істот включає кішок, собак та дерева.

Вислів «Сократ – людина» означає, що перетин безлічі Сократів (дуже невеликої множини) і множини всіх людей (набагато більшої множини) представляє безліч Сократів:

С × Л = С.

Оскільки з першого рівняння відомо, що Л дорівнює Л × Х, можемо підставити цей вираз до другого:

С × (Л × Х) = С.

Відповідно до асоціативного закону це рівнозначно виразу:

(С × Л) × Х = С.

Однак ми вже знаємо, що С × Л дорівнює С, тому можемо спростити вираз, використовуючи цю підстановку:

С × Х = С.

Тепер ми закінчили. Ця формула показує, що перетин безлічі Сократів і безлічі всіх смертних істот є С, а це означає, що Сократ смертний. Якби натомість виявилося, що С × Х дорівнює 0, ми дійшли б висновку, що Сократ не був смертним. Якби ми виявили, що С × Х дорівнює Х, то висновок полягав у тому, що Сократ є єдиною смертною істотою, а решта безсмертних.

Використання булевої алгебри може здатися зайвим для доказу очевидного факту (особливо з огляду на те, що Сократ довів власну смертність 2400 років тому), проте її можна використовувати для того, щоб визначити, чи задовольняє щось певний набір критеріїв. Можливо, одного разу ви зайдете в зоомагазин і скажете продавцеві: «Мені потрібен стерилізований кіт білого або рудого забарвлення, або стерилізована кішка будь-якого

забарвлення, крім білого, або я візьму будь-яку з чорних кішок, що є у вас». І продавець скаже, що вам потрібна кішка з множини, що описується наступним виразом:

(М × С × (Б + Р)) + (Ж × С × (1 - Б)) + Ч.

Правильно? І ви відповісте: «Так! Точно!»

Перевіряючи, чи правильно продавець вас зрозумів, можна відмовитися від понять об'єднання та перетину, замість них використовувати слова АБО та І. Я пишу ці слова великими літерами, тому що вони не тільки відповідають поняттям у звичайній мові, але й можуть являти собою операції у булевій алгебри. Коли ви формуєте об'єднання двох множин, ви фактично берете елементи з першої або другої множини. А коли ви формуєте перетин, то берете тільки ті елементи, які одночасно належать першим І другим множинам. Крім того, ви можете використовувати слово НЕ скрізь, де є символ 1, за яким слідує знак «мінус». Таким чином:

• символ «+» (який раніше позначав об'єднання) тепер означає АБО;

• символ «×» (який раніше позначав перетин) тепер означає І;

• вираз «1-» (раніше позначає безліч всіх елементів, крім чогось) тепер означає НЕ.

Саме тому наведений вище вираз може бути записано:

(М І С І (Б АБО Р)) АБО (Ж І С І (НЕ Б)) АБО Ч.

Як бачите, це відповідає тому, що ви сказали. Зверніть увагу, як дужки уточнюють ваші побажання. Вам потрібна кішка, що належить одній з трьох множин.

(М І С І (Б АБО Р))
АБО
(Ж І С І (НЕ Б))
АБО
Ч

За допомогою цієї формули продавець може виконати те, що називається *перевіркою умови.* Непомітно ми перейшли до дещо іншої форми булевої алгебри, де літери не позначають безлічі. Натомість літери тепер можуть відповідати числам. Однак буквам може бути присвоєно тільки значення 0 або 1. Число 1 означає так, істина, дана конкретна кішка задовольняє цим критеріям, число 0 – ні, брехня, дана кішка не задовольняє цим критеріям.

Спершу продавець приносить нестерилізованого рудого кота. Ось вираз, що описує безліч прийнятних кішок:

(М × С × (Б + Р)) + (Ж × С × (1 – Б)) + Ч.

Ось як воно виглядає після підстановки значень 0 та 1:

(1 × 0 × (0 + 1)) + (0 × 0 × (1–0)) + 0.

Зверніть увагу: єдиними символами, яким було надано значення 1, є М і Р, оскільки йдеться про рудого кота.

Тепер потрібно спростити цей вираз. Якщо він спрощується до 1, то кішка задовольняє ваші критерії; якщо він спрощується до 0, то кішка критеріям не задовольняє. Майте на увазі, що в процесі спрощення висловлювання ми насправді нічого не додаємо і не множимо, хоча зазвичай можемо вдати, що виконуємо ці операції. Більшість тих же правил застосовуються тоді, коли символ «+» означає АБО, а символ «×» – І. Іноді в сучасних текстах для позначення І та АБО використовуються символи «Λ» та «V» замість «×» та «+». Однак саме тут символи «+» та «×», мабуть, мають найбільше значення.

Коли символ «×» означає І, можливі результати:

0 × 0 = 0;
0 × 1 = 0;
1 × 0 = 0;
1×1 = 1.

Іншими словами, результат дорівнює 1 тільки в тому випадку, якщо лівий І правий операнди дорівнюють 1. Ця операція відповідає звичайному множенню і називається *кон'юнкцією,* і її можна

описати за допомогою невеликої таблиці, аналогічної таблицям додавання та множення, наведеним в темі 8.

AND	**0**	**1**
0	0	0
1	0	1

Коли символ "+" означає АБО, можливі наступні результати.
0 + 0 = 0;
0 + 1 = 1;
1+0=1;
1+1=1.

Результат дорівнює 1, якщо лівий АБО правий операнд дорівнює 1. Результат цієї операції схожий на результати звичайного додавання, за винятком того, що в даному випадку 1 + 1 дорівнює 1. Результати операції АБО, яка називається *диз'юнкцією,* можна подати у вигляді іншої таблиці.

OR	**0**	**1**
0	0	1
1	1	1

Ми готові використовувати ці таблиці для обчислення:
(1×0×1) + (0×0×1) + 0 = 0 + 0 + 0 = 0.

Результат 0 – "ні", "брехня", це кошеня не підходить.
Потім продавець приносить стерилізовану білу кішку. Вихідний вираз виглядав так:
(М × С × (Б + Р)) + (Ж × С × (1 - Б)) + Ч.
Знову підставимо в нього значення 0 та 1:
(0 × 1 × (1 + 0)) + (1 × 1 × (1-1)) + 0.

І спростимо його:
(0 × 1 × 1) + (1 × 1 × 0) + 0 = 0 + 0 + 0 = 0.

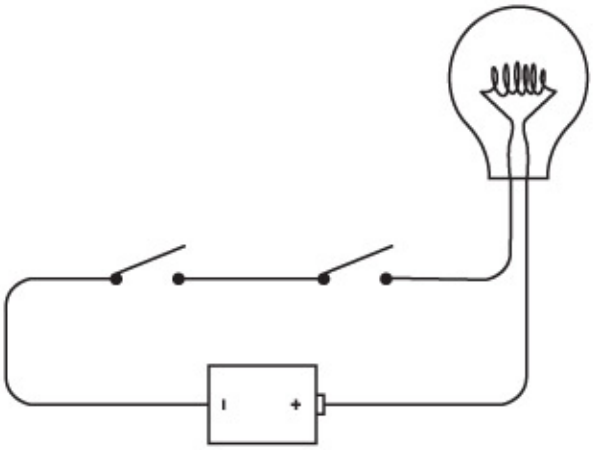
Ще одне нещасне кошеня відкинуто.
Потім продавець приносить стерилізовану сіру кішку. (Сірий відповідає критерію «інше забарвлення», тобто не біле, не чорне і не руде.) Ось відповідний вираз:
(0 × 1 × (0 + 0)) + (1 × 1 × (1-0)) + 0.

Тепер спростимо його:
(0 × 1 × 0) + (1 × 1 × 1) + 0 = 0 + 1 + 0 = 1.

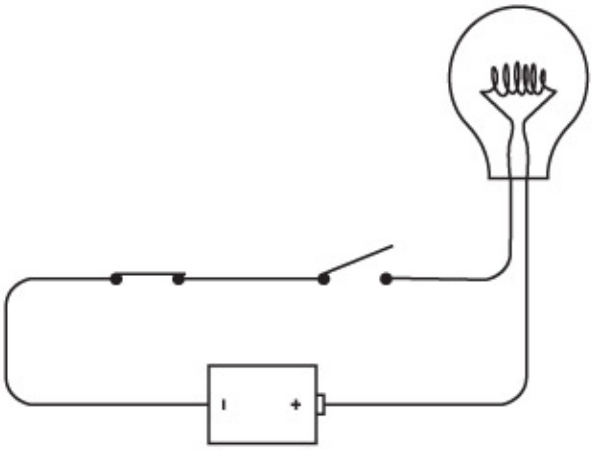
Результат обчислення, що дорівнює 1, означає «так», «істина», кошеня знайшло свій будинок. (Крім того, він виявився наймилішим!)

Пізніше того ж вечора, поки кошеня спить у вас на колінах, ви запитуєте себе, чи не можна підключити кілька перемикачів до лампочки для полегшення процесу перевірки кошенят на відповідність вашим критеріям. (Так, ви дуже дивна дитина.) Самі не знаючи того, ви впритул наблизилися до вирішального концептуального прориву. Ви ось-ось проведете деякі експерименти, які поєднають алгебру Джорджа Буля з електрикою і уможливлять проектування та збирання комп'ютерів, що працюють з двійковими числами. Однак нехай це вас не лякає.

Щоб провести такий експеримент, ви, як завжди, з'єднуєте лампочку та батарейку, але використовуєте два перемикачі замість одного.

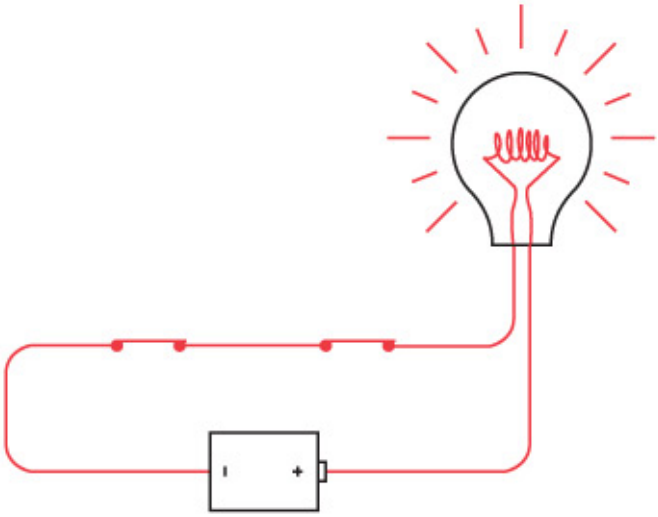


Вважається, що перемикачі, підключені один за одним, *послідовно з'єднані.* Якщо ви



замикаєте лівий перемикач, нічого не відбувається.

Якщо ви залишаєте лівий перемикач розімкненим, а замикаєте правий, також нічого не



станеться. Лампочка спалахує, коли і лівий, і правий перемикачі виявляються замкнутими.

Ключовим у цьому випадку є союз «і». Лівий *і* правий перемикачі повинні бути замкнуті, щоб струм йшов ланцюгом.

Ця схема вирішує невелике логічне завдання. Фактично лампочка відповідає на запитання: «Чи замкнуті обидва перемикачі?» Ми можемо підсумувати результати роботи цієї схеми у таблиці.

Open	Open	Not lit
Open	Closed	Not lit
Closed	Open	Not lit
Closed	Closed	Lit

У попередньому розділі ми говорили про те, як за допомогою двійкових цифр, або бітів, можна подати будь-яку інформацію, починаючи від чисел і закінчуючи напрямком великого пальця Роджера Еберта. Ми могли сказати, що нуль біт означає, що палець спрямований вниз, а один біт – палець спрямований вгору. Перемикач може перебувати у двох положеннях, тому для його опису достатньо одного біта. Можна сказати, що 0 – це "перемикач розімкнуто", а 1 – "перемикач замкнутий". Лампочка також має два стани, отже, для їх опису достатньо одного біта. Можна сказати, що 0 – "лампочка не горить", а 1 – "лампочка горить". Тепер ми просто переписуємо наведену вище таблицю.

Left Switch	Right Switch	Lightbulb
0	0	0
0	1	0
1	0	0
1	1	1

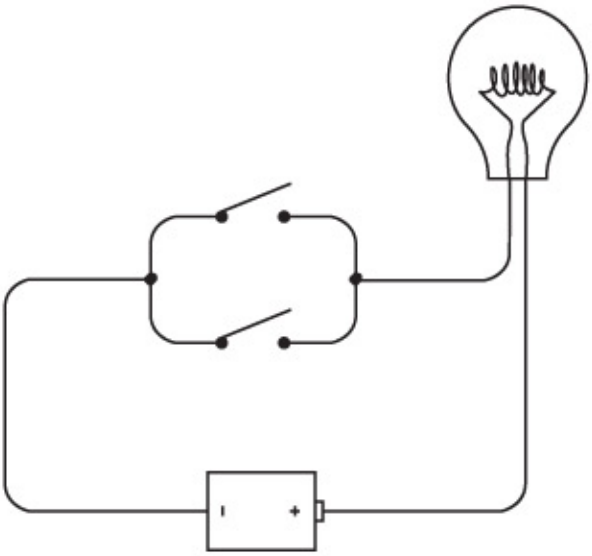
Зверніть увагу: якщо ми поміняємо місцями лівий та правий перемикачі, результати залишаться незмінними. Нам не обов'язково розрізняти перемикачі. Саме тому таблицю можна переписати так, щоб вона нагадувала наведені І/АБО.

Switches in Series	0	1
0	0	0
1	0	1

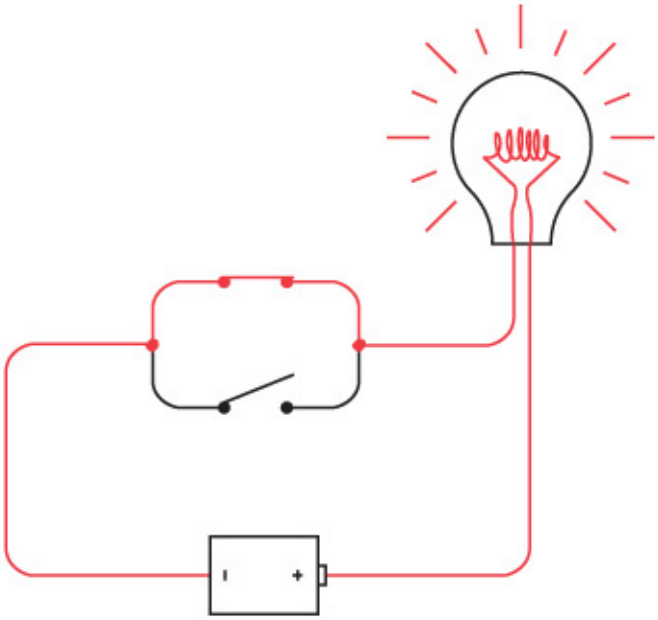
Справді, це відповідає таблиці з наслідками виконання булевої операції І.

AND	0	1
0	0	0
1	0	1

Ця проста схема фактично виконує операцію І в булевій алгебрі.
Тепер спробуйте з'єднати два перемикачі інакше.

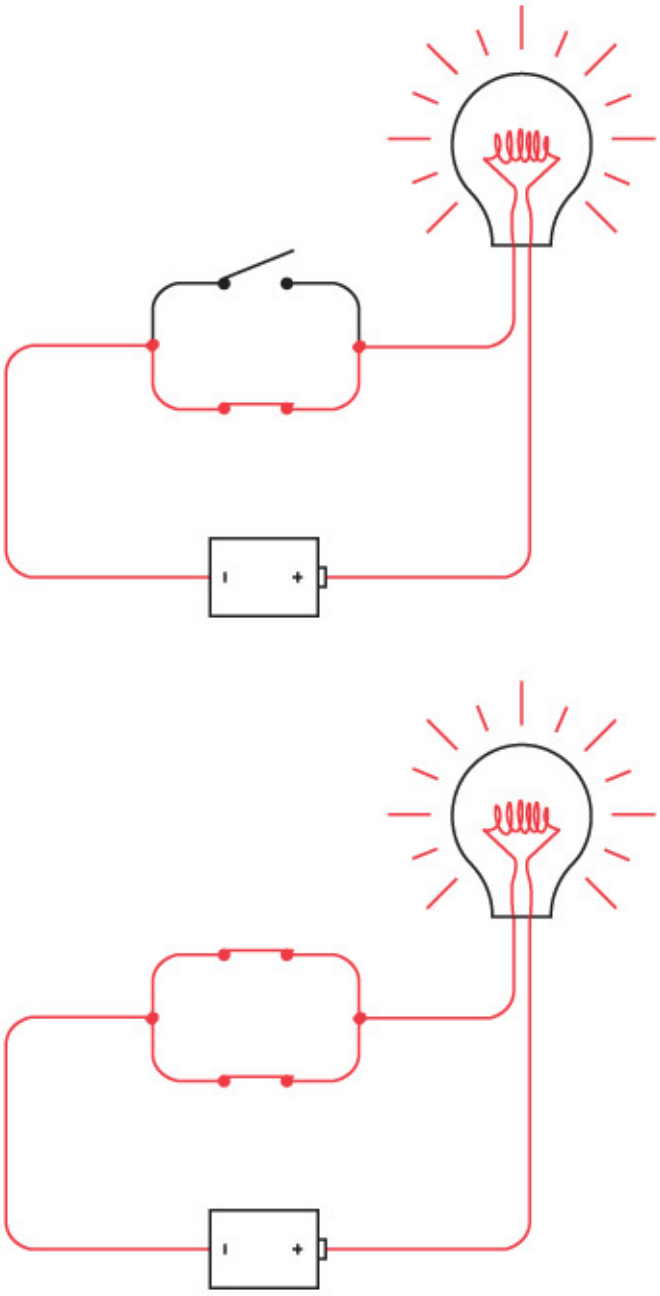


Перемикачі з'єднані *паралельно.* Різниця між цим та попереднім способом з'єднання полягає



в тому, що ця лампочка загориться, якщо ви замкнете верхній перемикач.

Або нижній перемикач.



Можна також замкнути обидва перемикачі.

Лампочка спалахує, якщо замкнути верхній *або* нижній перемикач. Ключовим словом у разі є союз «або».

Знову ж таки, дана схема вирішує логічне завдання. Лампочка відповідає на запитання: «Чи замкнутий хоча б один перемикач?» У таблиці показані результати роботи цієї схеми.

Left Switch	Right Switch	Lightbulb
Open	Open	Not lit
Open	Closed	Lit
Closed	Open	Lit
Closed	Closed	Lit

Тепер знову використовуємо 0 для позначення розімкнутого перемикача або лампочки, що не горить, і 1 – для позначення замкнутого перемикача або лампочки, що горить, в результаті чого отримаємо наступну таблицю.

Left Switch	Right Switch	Lightbulb
0	0	0
0	1	1
1	0	1
1	1	1

Знову ж таки нічого не зміниться, якщо перемикачі поміняти місцями, тому таблицю можна заповнити наступним чином.

Switches in Parallel	0	1
0	0	1
1	1	1

Ймовірно, ви вже здогадалися, що ця таблиця відповідає результатам булевої операції АБО.

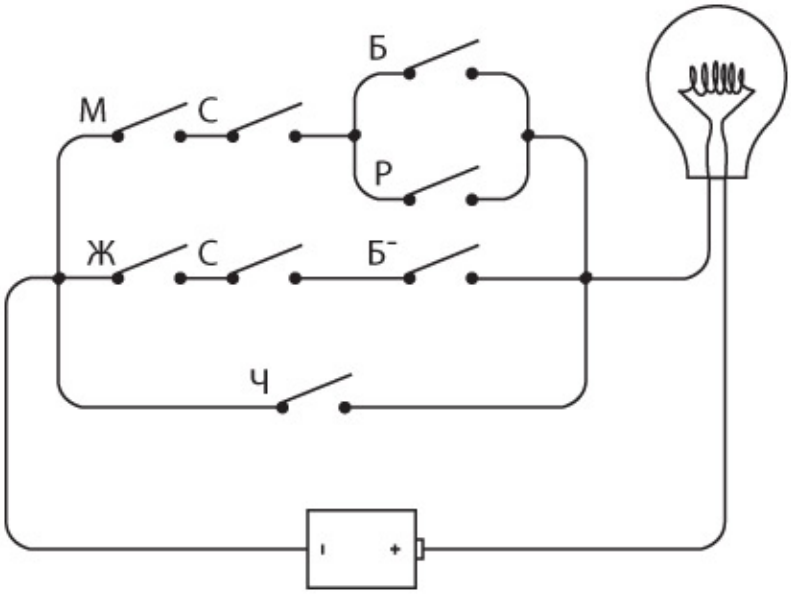
OR	0	1
0	0	1
1	1	1

Отже, два з'єднаних паралельно перемикачі виконують операцію, еквівалентну булевій операції АБО.

З'явившись у зоомагазині, ви сказали продавцеві: «Мені потрібен стерилізований кіт білого або рудого забарвлення; або стерилізована кішка будь-якого забарвлення, крім білого; або я візьму будь-яку з наявних у вас чорних кішок», – і продавець склав такий вираз:

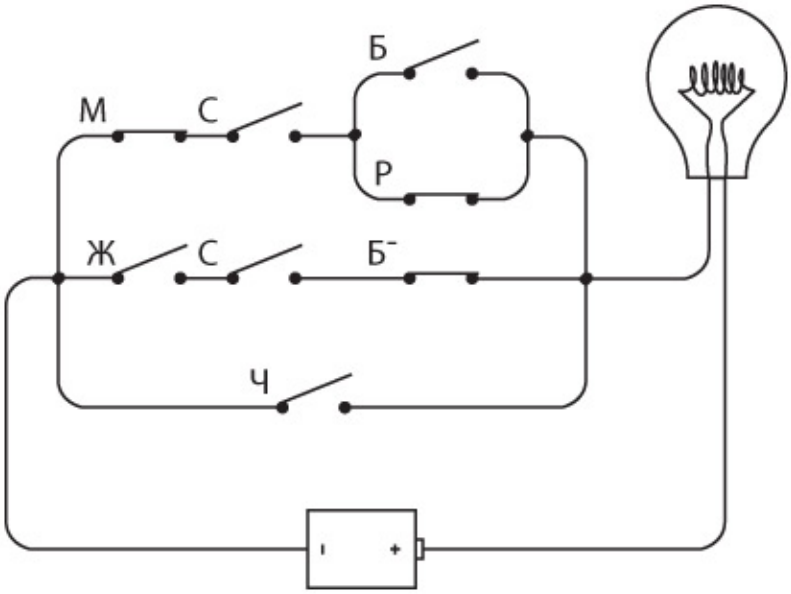
(М × С × (Б + Р)) + (Ж × С × (1 − Б)) + Ч.

Тепер, коли ви знаєте, що два з'єднані послідовно перемикачі виконують логічну операцію І (позначається символом «×»), а два перемикачі, з'єднані паралельно, – логічну операцію АБО (позначається символом «+»), ви можете з'єднати вісім перемикачів.



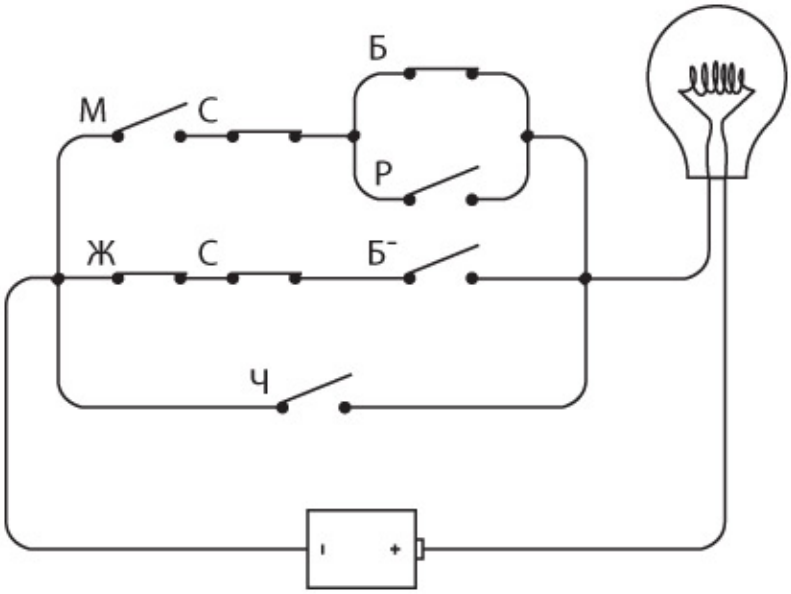
Усі перемикачі у цій схемі позначені літерами, відповідними літерам у булевому виразі. (Б ¯ означає НЕ Б і є альтернативним способом запису виразу 1 - Б). Дійсно, якщо ви переглянете електричну схему зліва направо і зверху вниз, то зіткнетеся з літерами в тому самому порядку, в якому вони представлені у виразі. Кожен символ «×» відповідає місцю схеми, де два перемикачі (або дві групи перемикачів) з'єднані послідовно, кожен символ «+» – місцю схеми, в якому два перемикачі (або дві групи перемикачів) з'єднані паралельно.

Як ви пам'ятаєте, продавець спершу приніс нестерилізованого рудого кота. Замкніть

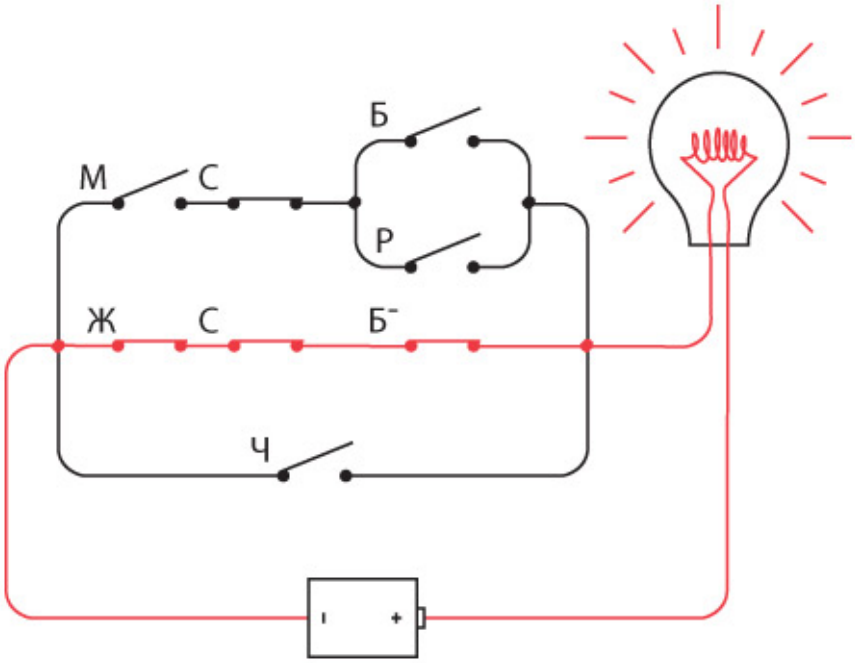


відповідні перемикачі.

Незважаючи на те, що перемикачі М, Р і НЕ Б замкнуті, лампочка не спалахує. Потім продавець приніс стерилізовану білу кішку.



Знову ж таки, замкнуті не всі потрібні перемикачі для того, щоб спалахнула лампочка.



Нарешті продавець приносить стерилізовану сіру кішку.

Так можна замкнути всі потрібні перемикачі, запалити лампочку та показати, що кошеня задовольняє всім вашим критеріям.

Джордж Буль ніколи не збирав таку схему. Йому ніколи не доводилося бачити логічний вираз, реалізований за допомогою перемикачів, дротів та лампочок. Зрозуміло, однією з перешкод було те, що лампа розжарювання була винайдена лише через 15 років після смерті Буля. Проте Семюел Морзе продемонстрував свій телеграф у 1844 році – за десять років до публікації книги Буля «Дослідження законів мислення», і йому нічого не вартувало замінити лампочки в наведеній вище схемі клопфером.

Нікому в XIX столітті не вдалося вловити зв'язок між булевими операціями І та АБО і послідовним і паралельним з'єднанням простих перемикачів – ні математику, ні електрику, ні оператору телеграфу. Це не спало на думку навіть батькові-засновнику комп'ютерної революції – Чарльзу Бебіджу (1792–1871), який переписувався з Булем і був знайомий з його роботою, а більшу частину життя витратив на розробку різницевої, а потім аналітичної машини, яка через століття буде вважатися попередником сучасних комп'ютерів. Зараз ми знаємо, що Беббіджу допомогло усвідомлення того, що замість шестерень та важелів для виконання обчислень краще використовувати телеграфні реле.

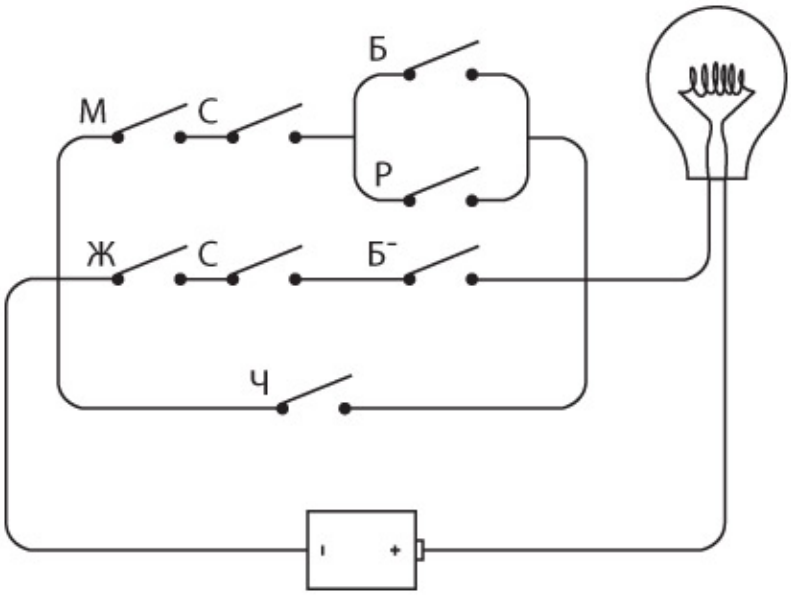
Так, телеграфне реле.

Тема 7. Логічні вентилі

У далекому майбутньому, коли історія примітивних обчислень XX століття перетвориться на перекази, хтось, мабуть, припустить, що *логічні вентилі* були названі на честь однойменного сантехнічного пристрою. Це не зовсім так. Ми невдовзі побачимо, що логічні вентилі справді нагадують звичайні вентилі, якими проходить вода, і виконують елементарні логічні завдання, блокуючи чи пропускаючи електричний струм.

У попередній темі ми розглядали сценарій, коли ви увійшли до зоомагазину і сказали продавцеві: «Мені потрібний білий або рудий стерилізований кіт або стерилізована кішка будь-якого кольору, крім білого, або будь-яка кішка або кіт чорного кольору». Ці критерії можна поєднати в наступний логічний вираз, а також виразити за допомогою схеми з перемикачів та лампочки:

(М × С × (Б + Р)) + (Ж × С × (1 − Б)) + Ч.



Така схема іноді називається *мережею,* хоча в даний час це слово набагато частіше використовується для позначення з'єднаних між собою комп'ютерів, а не набору перемикачів.

Незважаючи на те, що всі позначені на схемі елементи були винайдені в XIX столітті, тоді ніхто не уявляв, що логічні вирази можна реалізувати безпосередньо у вигляді електричних кіл. Ця можливість була усвідомлена лише у 1930-х роках Клодом Шенноном (1916–2001), який у 1938 році захистив знамениту магістерську дисертацію під назвою «Символьний аналіз реле та комутаторів». Через десять років вперше було опубліковано його статтю «Математична теорія зв'язку», у якій слово «біт» (bit) використовувалося для позначення *двійкової цифри.*

Зрозуміло, задовго до 1938 було відомо, що для протікання струму при послідовному з'єднанні двох перемикачів обидва повинні бути замкнуті, а при паралельному з'єднанні – лише один з них. Однак ніхто так ясно і переконливо, як Шеннон, не показав, що для проектування схем з перемикачами інженери-електрики можуть використовувати всі інструменти булевої алгебри. Зокрема, якщо ви можете спростити логічний вираз, що описує схему, можете спростити і саму схему.

Наприклад, вираз, що містить ваші критерії вибору кішки, виглядає так:

(М × С × (Б + Р)) + (Ж × С × (1 − Б)) + Ч.

Використовуючи сполучний закон, ми можемо змінити порядок змінних, об'єднаних знаком І («×»), і переписати вираз:

(С × М × (Б + Р)) + (С × Ж × (1 − Б)) + Ч.

Для ясності введу два додаткові символи X та Y:

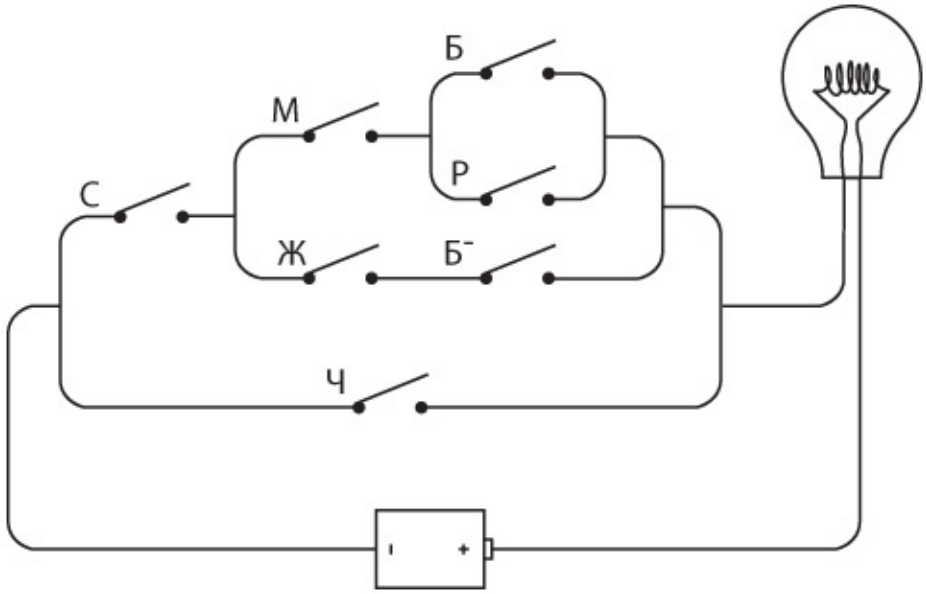
X = М × (Б + Р);

Y = Ж × (1 - Б).

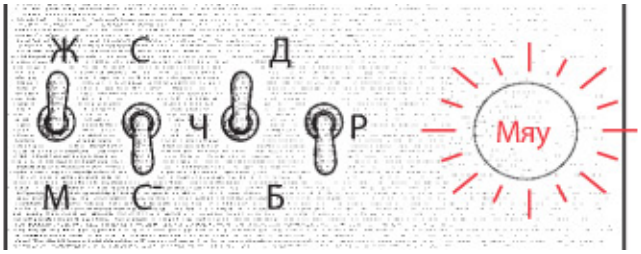
Тепер вираз із критеріями вибору кішки можна записати так:
(С × Х) + (С × Y) + Ч.

Нарешті ми можемо повернути значення виразів, що відповідають символам Х та Y.
Зверніть увагу: змінна С зустрічається двічі. Використовуючи розподільчий закон, цей вислів можна переписати тільки з однією змінною С:
(С × (Х + Y)) + Ч.

Тепер підставимо у вираз значення Х і Y:
(С × ((М × (Б + Р)) + (Ж × (1 − Б)))) + Ч.
Через безліч дужок цей вираз не виглядає спрощеним. Однак він містить на одну змінну менше, а значить, у схемі менше перемикачів. Ось її переглянута версія.



Справді, побачити, що ця схема еквівалентна попередній, легше, ніж помітити тотожність виразів.
Насправді в цьому ланцюгу, як і раніше, на три перемикачі більше, ніж потрібно. Теоретично для вибору ідеальної кішки має бути достатньо чотирьох перемикачів. Чому чотири? Кожен перемикач – це біт. Одного перемикача вистачить для вказівки статі (розімкнутий – відповідає коту, замкнутий – кішці), ще один вказуватиме на стерилізовану кішку в замкнутому стані та нестерилізовану – у розімкнутому, ще два дозволять розпізнати колір. Існують чотири можливі кольори: білий, чорний, рудий та «інший». І ми знаємо, що чотири варіанти можна визначити за допомогою двох бітів, тому для задання кольору потрібно лише два перемикачі. Наприклад, білому кольору можуть відповідати два розімкнуті перемикачі, чорному – один замкнутий, рудому – другий замкнутий, а "іншим" – два замкнутих.
Тепер давайте побудуємо пульт керування для вибору кішки, який складатиметься з лампочки та чотирьох перемикачів (схожих на ті, за допомогою яких ви вмикаєте та вимикаєте світло).



Перемикач замкнено, коли знаходиться в положенні вгору, розімкнуто – коли знаходиться в положенні вниз. Боюся, що позначення двох перемикачів для вибору кольору кішки можуть здатися трохи незрозумілими, проте це наслідок спроби обійтись при створенні пульта керування мінімумом коштів. Лівий перемикач у цій парі позначений буквою *Ч;* замикання лише лівого перемикача (як показано на малюнку) відповідає чорному кольору. Правий перемикач у цій парі позначений буквою *Р;* замикання тільки правого перемикача відповідає рудому кольору, замикання обох – "іншому" кольору (цей варіант позначений буквою *Д)*. Розмикання обох перемикачів відповідає білому кольору та позначається буквою *Б* унизу.

Якщо користуватися комп'ютерною термінологією, набір перемикачів – це *пристрій введення* інформації, що управляє поведінкою ланцюга. У цьому випадку перемикачі відповідають чотирьом бітам, що дозволяють описати кішку. *Пристроєм виведення* є лампочка, яка загоряється, якщо положення перемикачів узгоджується з описом кішки. Перемикачі, зображені на попередньому малюнку, описують нестерилізовану чорну кішку. Її характеристики задовольняють вашим критеріям, тому лампочка спалахує.

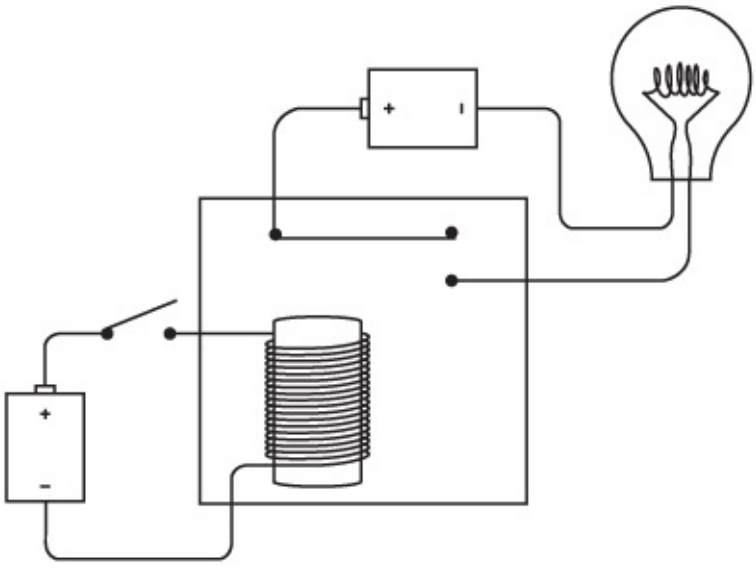
Тепер нам потрібно лише сконструювати схему, яка пожвавить цей пульт керування.

Як ви пам'ятаєте, дисертація Клода Шеннона називалася «Символьний аналіз реле та комутаторів». Описані ним реле були дуже схожі на телеграфні, про які ми говорили в темі 6. Однак на момент публікації роботи Шеннона реле використовувалися для інших цілей, зокрема телефонної мережі.

Подібно до перемикачів, реле можна з'єднувати послідовно і паралельно для вирішення простих логічних завдань. Ці комбінації називаються *логічними вентилями.* Коли я говорю, що ці логічні вентилі вирішують *прості* логічні завдання, я маю на увазі максимально прості завдання. Перевага реле в порівнянні з перемикачами полягає в тому, що їх можна вмикати та вимикати автоматично (за допомогою інших реле), а не вручну. Таким чином, логічні вентилі можна комбінувати для вирішення складніших завдань, наприклад для виконання простих арифметичних операцій. У наступній темі буде показано, як із перемикачів, лампочок, джерела живлення та телеграфних реле можна зібрати лічильну машину (нехай і працюючу виключно з двійковими числами).

Як відомо, реле відігравали ключову роль роботі телеграфної системи. Через великі відстані проводи, що з'єднують телеграфні станції, мали дуже високий опір. Потрібно було пристрій, здатний приймати слабкий сигнал і передавати ідентичний, але потужніший. Реле вирішувало це завдання, використовуючи електромагніт для керування перемикачем. По суті, реле *посилювало* слабкий сигнал для отримання більш потужного.

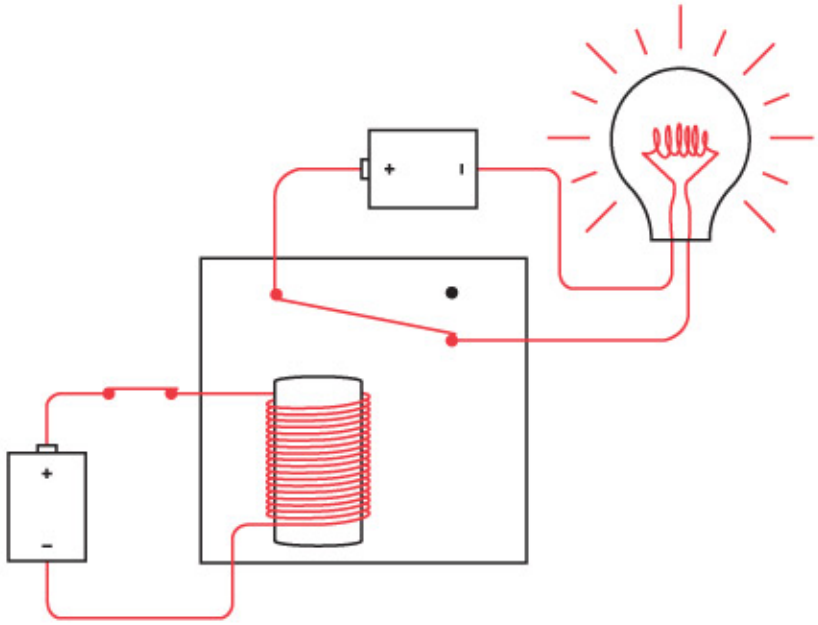
У наші плани не входить використання реле для посилення слабкого сигналу. Нас цікавить лише те, що реле є перемикачем, яким можна керувати не вручну, а електрикою. Ми можемо



з'єднати реле з перемикачем, лампочкою та парою батарейок.

Зверніть увагу: перемикач зліва розімкнуто, а лампочка не горить. Коли ви замкнете перемикач, струм з батареї ліворуч від нього потече по витках котушки, намотаної на залізний

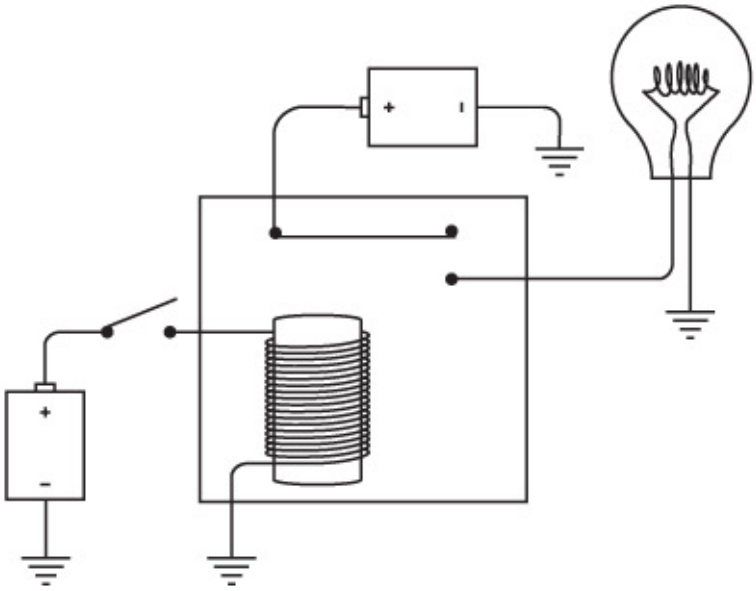
сердечник, який отримає магнітні властивості і притягне гнучку металеву смужку, що, у свою чергу, призведе до замикання ланцюга та включення лампочки.



Коли електромагніт притягує металеву смужку, реле вважається *активованим.* Після розмикання вимикача залізний сердечник втрачає магнітні властивості, а металева смужка повертається у вихідне положення.

Такий спосіб запалити лампочку здається досить складним, і це справді так. Якби ми хотіли обмежитись лише включенням лампочки, ми могли б обійтися і без реле. Однак перед нами стоїть складніше завдання.

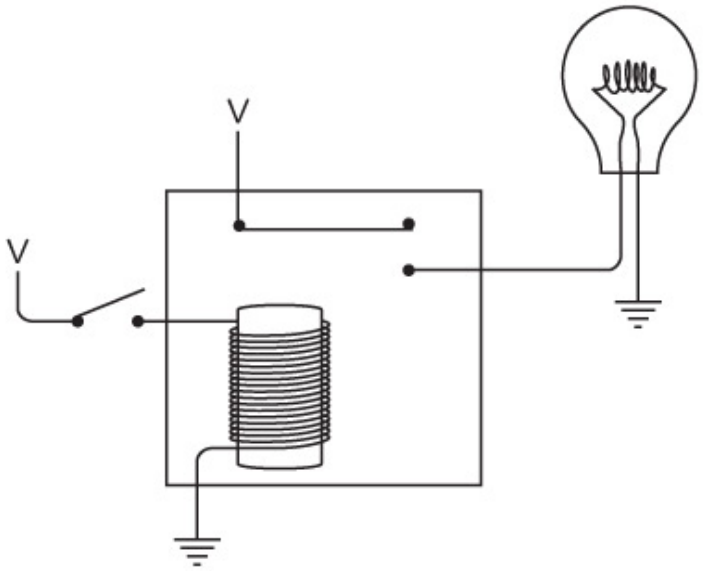
У цій темі ми часто використовуватимемо реле (а після складання логічних вентилів повністю від них відмовимося), тому хочу спростити схему. Ми можемо позбавитися деяких проводів за допомогою землі. У цьому прикладі «земля» – просто загальний провід; до реальної



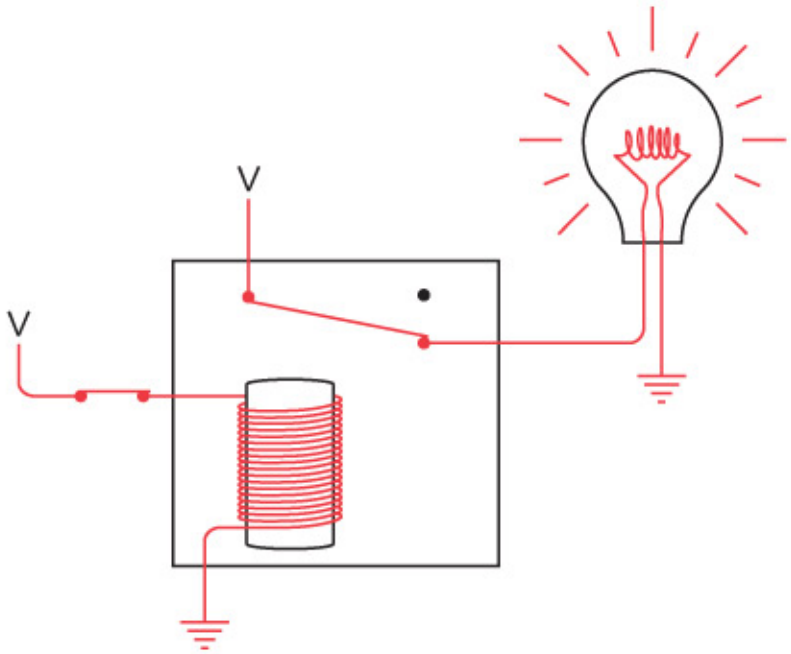
землі нічого не потрібно підключати.

Розумію, це не схоже на спрощення, але ми ще не закінчили. Важливо: негативні контакти обох акумуляторів підключені до землі.

Так що скрізь, де нам зустрінеться подібне зображення, замінимо його великою літерою *V* (яка означає voltage – "напруга"), як зробили це в темах 5 і 6. Тепер реле виглядає так.

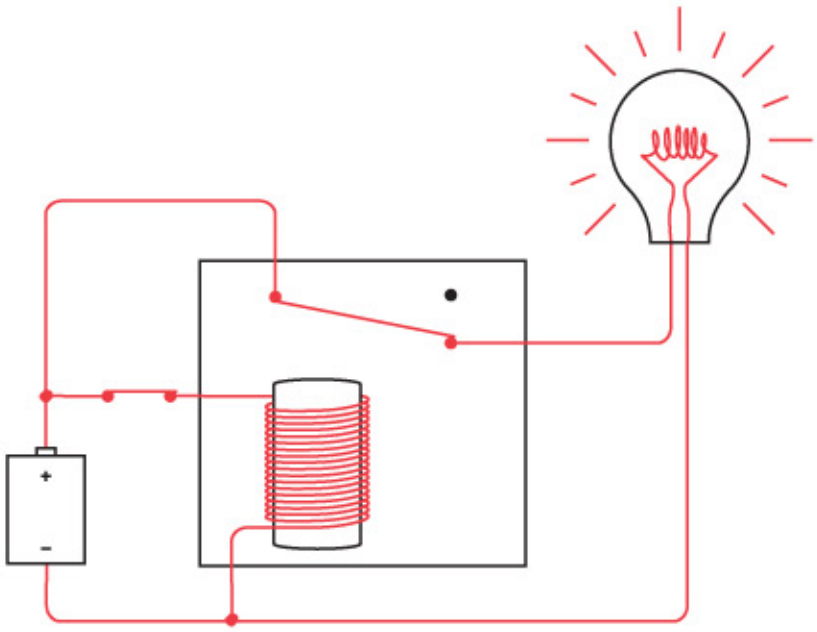


Коли перемикач замкнутий, струм між джерелом живлення (V) та землею тече через котушку електромагніта. Це змушує електромагніт притягнути гнучку металеву смужку, яка замикає

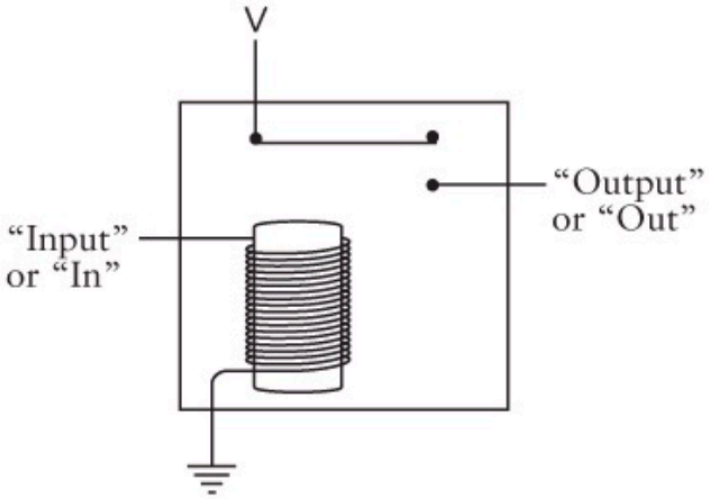


ланцюг між джерелом живлення, лампочкою та землею, і лампочка загоряється.

На цих схемах присутні два джерела живлення та дві землі, проте всі джерела живлення, як і всі землі, на наведених у цій темі схемах можуть бути з'єднані один з одним. Всі схеми, що складаються з реле та логічних вентилів, зображені в цій та наступній темах, допускають використання тільки однієї (хоча і потужної) батарейки. Наприклад, попередню схему можна перемалювати лише з одним джерелом живлення.

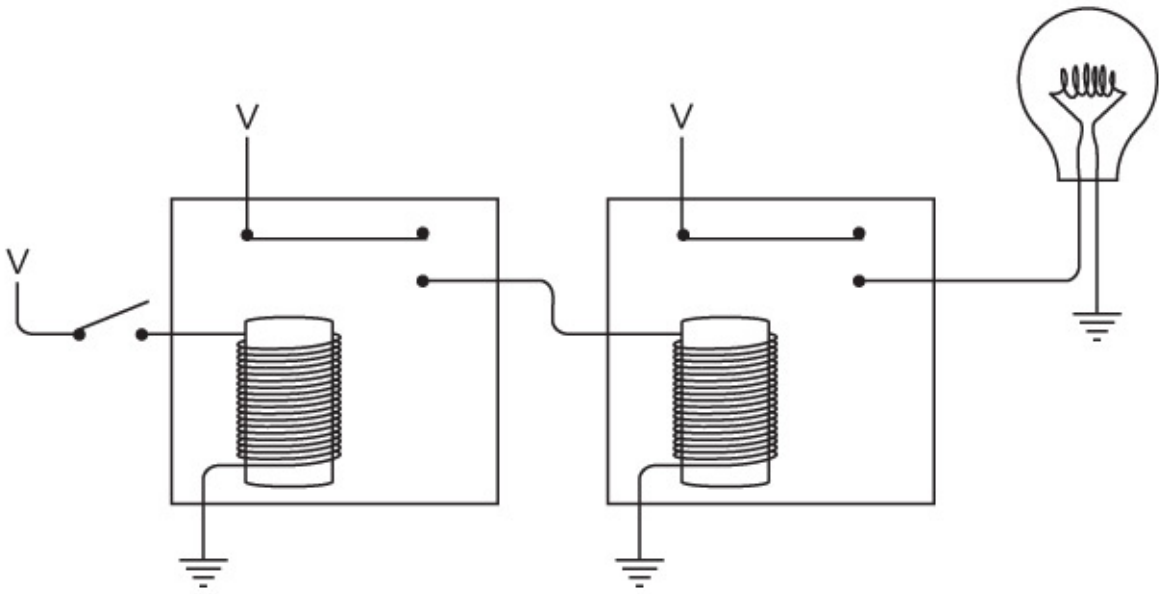


Враховуючи те, що ми збираємося робити з реле, ця схема не досить зрозуміла. Краще уникати замкнутих ланцюгів і розглядати роботу реле, як і у випадку з описаним раніше пультом керування, з точки зору *вхідного* та *вихідного* сигналів.

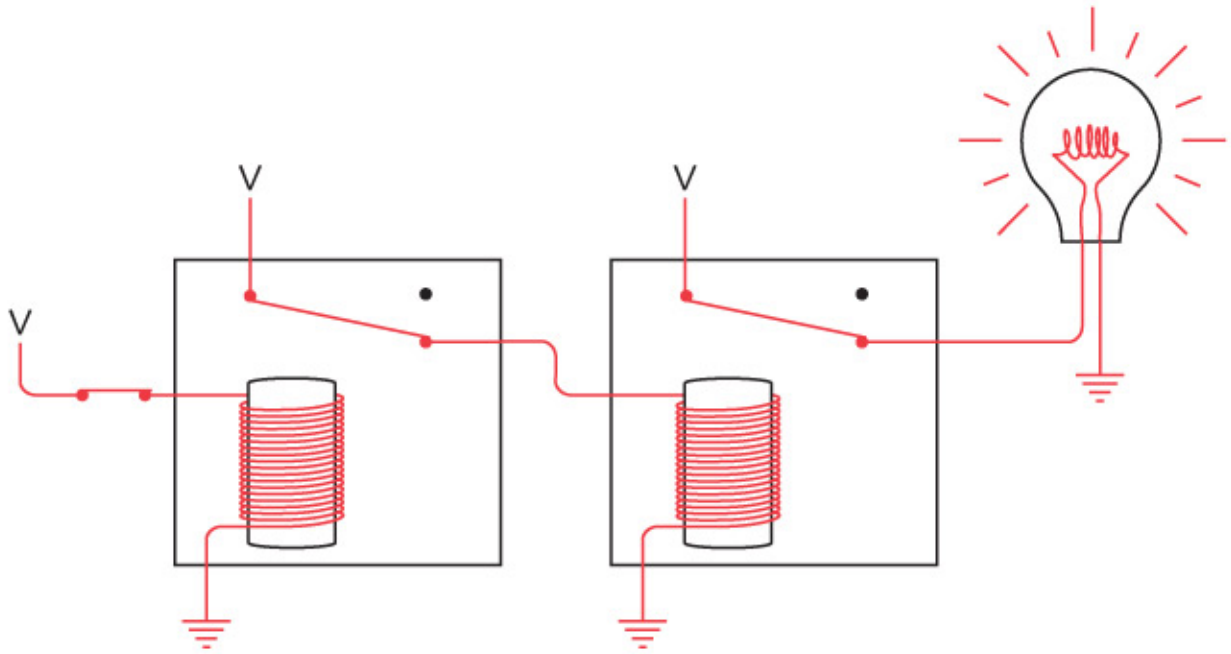


Якщо напруга надходить на вхід (наприклад, якщо він з'єднаний з джерелом живлення за допомогою перемикача), активується електромагніт, і на виході з'являється напруга.

До входу реле не обов'язково підключати перемикач, а до виходу лампочку. Вихід одного реле може бути підключений до іншого входу.



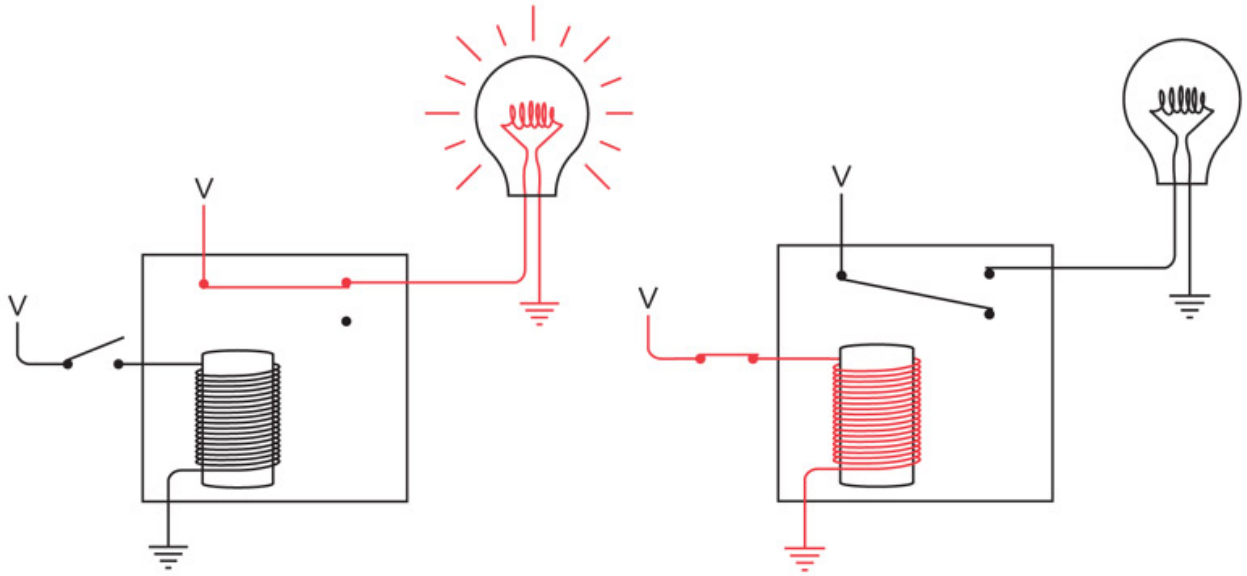
Замикання перемикача активує перше реле, яке потім подає напругу другому. Спрацювання



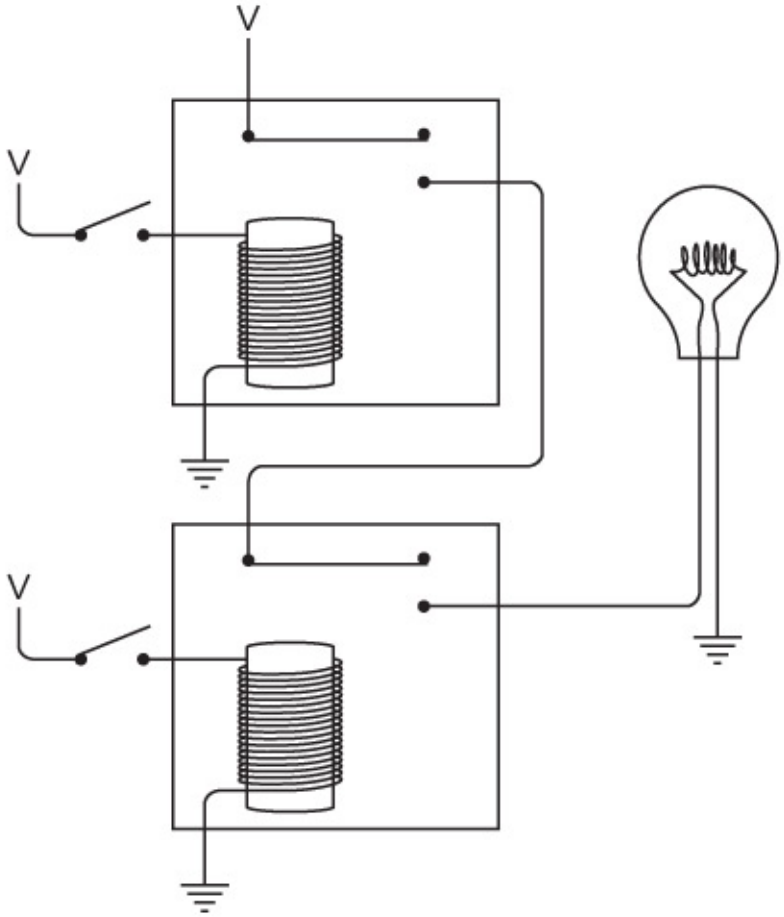
другого реле призведе до включення лампочки.

Поєднуючи кілька реле, можна конструювати логічні вентилі.

Насправді лампочку можна підключити до реле двома способами. Зверніть увагу на гнучку металеву деталь, яку притягує електромагніт. У стані спокою вона торкається одного контакту. Коли електромагніт притягує її, вона торкається іншого контакту. Ми використовували нижній контакт як вихід реле, проте могли б застосувати і верхній. У цьому випадку вихід реле змінюється на протилежний, і лампочка спалахує при розмиканні вхідного перемикача. При замиканні вхідного перемикача лампочка гасне.

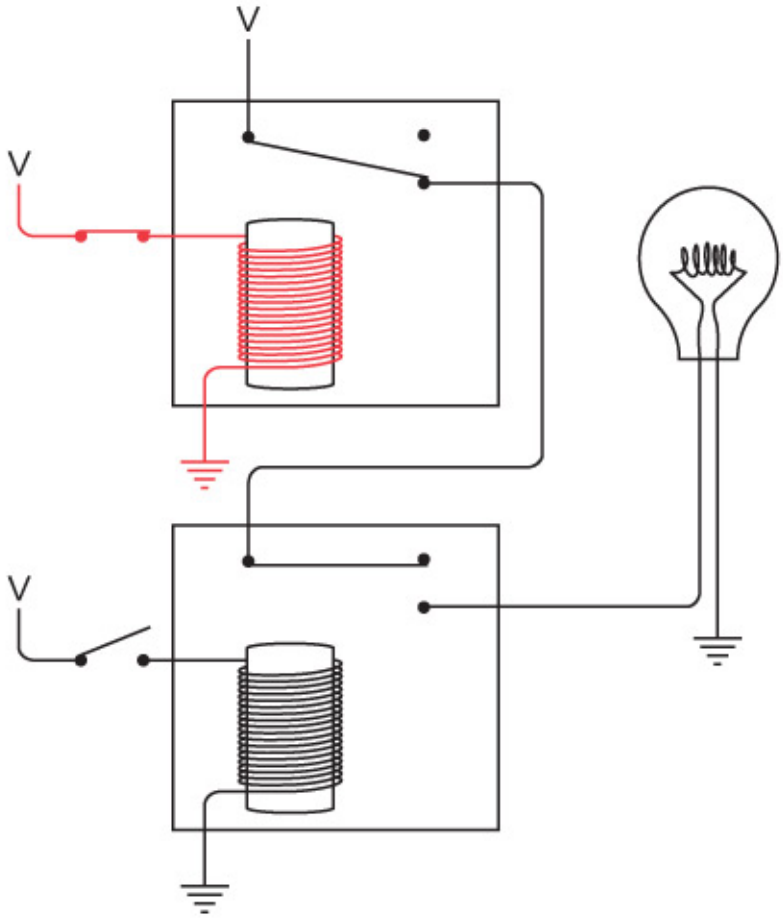


Реле такого типу називається *двопозиційним.* Воно має два електрично протилежні виходи. Коли на одному з них є напруга, на іншому її немає.

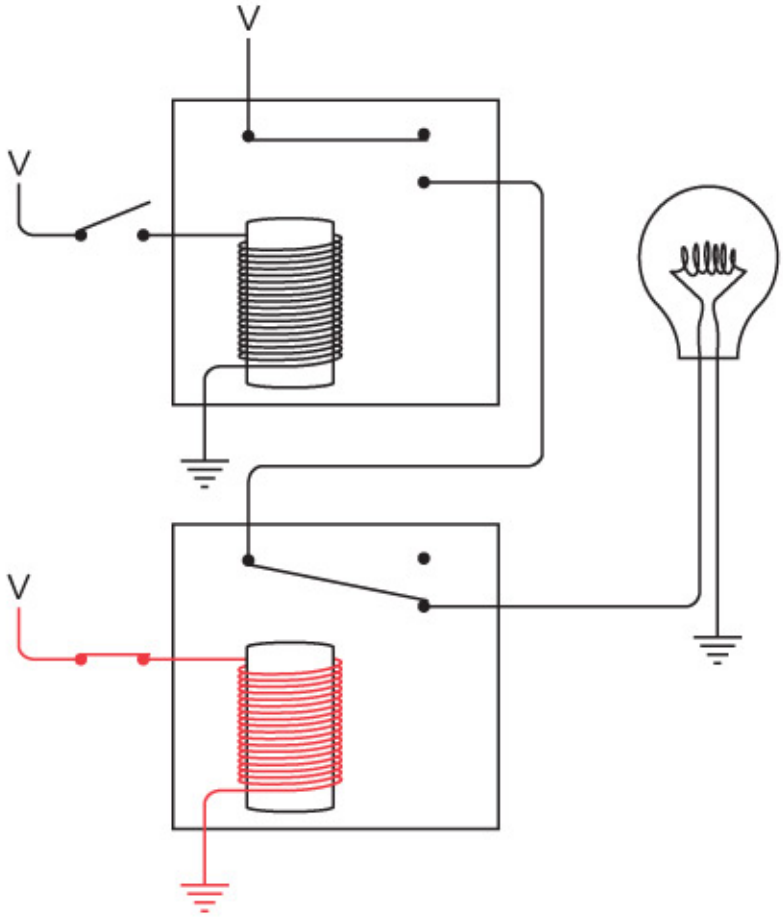


Як і у випадку з перемикачами, два реле можуть бути послідовно з'єднані.

Вихід верхнього реле подає напругу на нижнє. Як бачите, коли обидва перемикачі розімкнені, лампочка не горить. Спробуємо замкнути верхній перемикач.

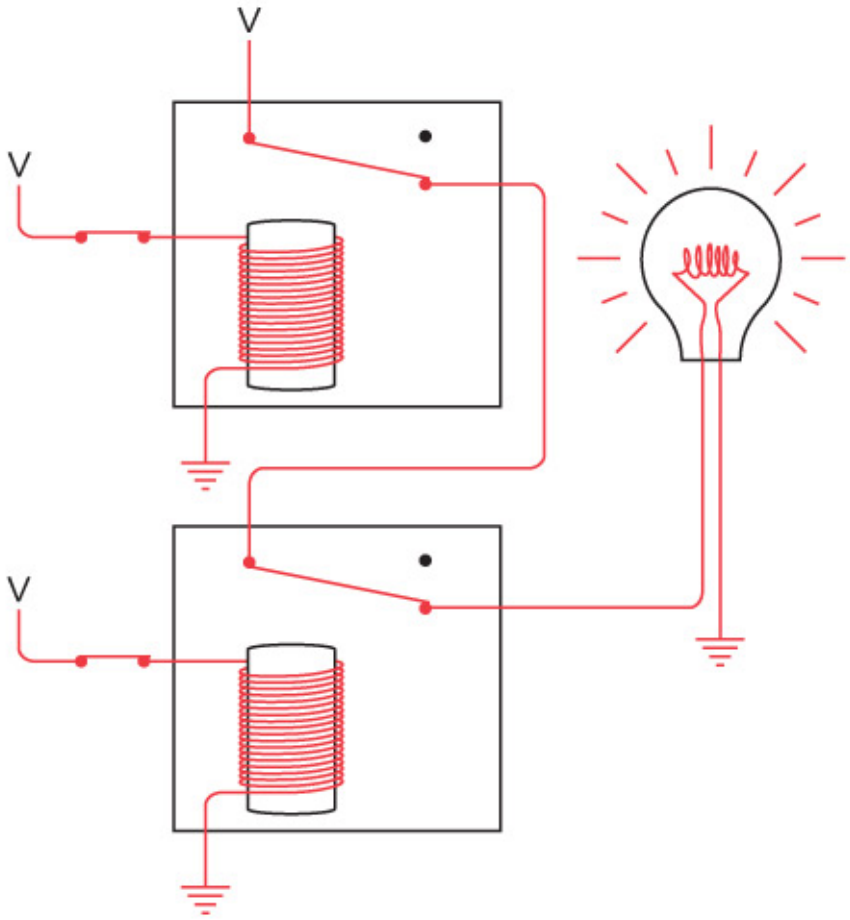


Лампочка не спалахує, оскільки нижній перемикач все ще розімкнуто, і друге реле не



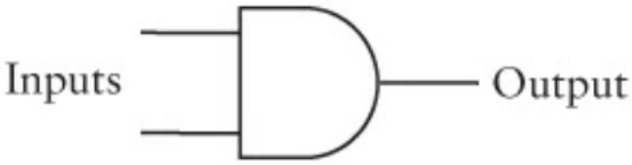
спрацьовує. Спробуємо розімкнути верхній перемикач та замкнути нижній.

Лампочка, як і раніше, не горить. Струм до неї не доходить, тому що не спрацювало перше реле. Єдиним способом запалити лампочку є замикання обох перемикачів.



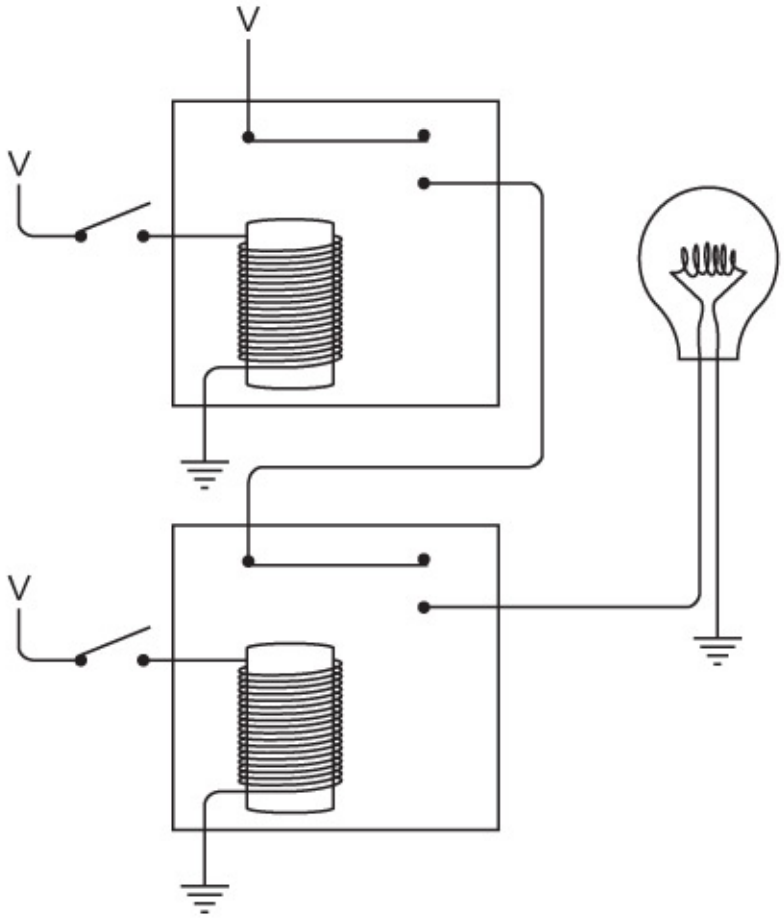
Тепер активовані обидва реле, і між джерелом живлення, лампочкою та землею тече струм.

Подібно до двох з'єднаних послідовно перемикачів, ці два реле вирішують невелике логічне завдання. Лампа спалахує лише у випадку спрацьовування обох реле. Така схема послідовного з'єднання двох реле називається вентилем *I.* Для його позначення на схемах інженери-електрики використовують спеціальний символ, який має такий вигляд.

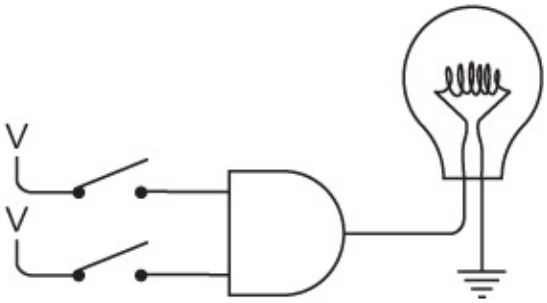


Це перший із чотирьох основних логічних вентилів. Вентиль I має два входи (ліворуч на наведеній вище схемі) та один вихід (праворуч). Вам часто зустрічатиметься саме таке зображення вентиля I – з входами зліва та виходом праворуч. Справа в тому, що людям, які звикли читати зліва направо, також зручніше вивчати електричні схеми зліва направо. Однак входи логічного вентиля можна було б зобразити вгорі, праворуч або внизу.

Вихідна схема з двома послідовно з'єднаними реле, двома перемикачами та лампочкою виглядала так.

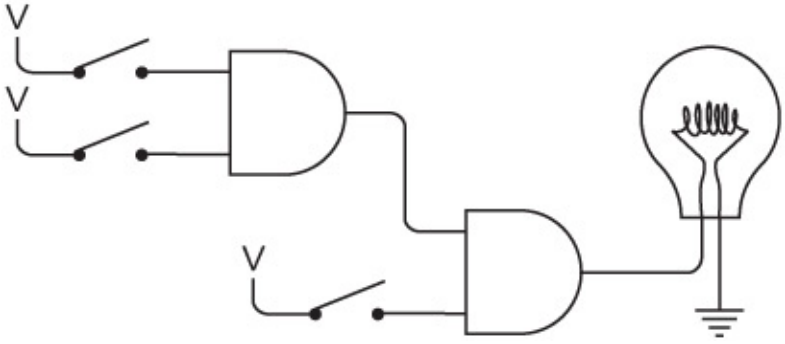


З використанням символу вентиля І ця ж схема набуває наступного вигляду.



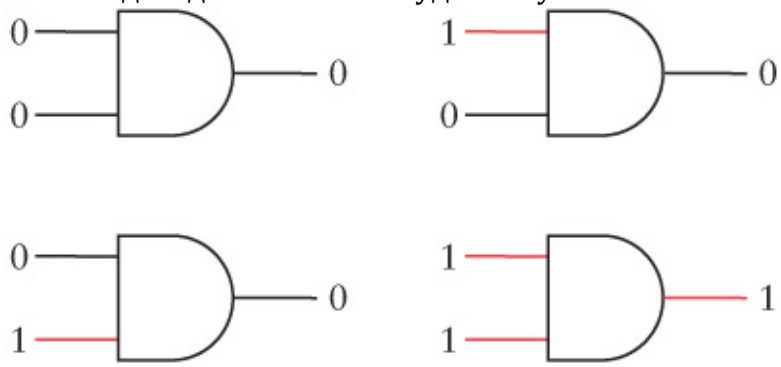
Зверніть увагу: символ вентиля І не тільки використовується замість двох з'єднаних послідовно реле, але також має на увазі, що верхнє реле підключено до джерела живлення та обидва реле з'єднані із землею. Знову ж таки, лампочка загоряється тільки у разі замикання верхнього *і* нижнього перемикачів. Саме тому ця схема називається *вентилем І.*

Входи вентиля І не обов'язково повинні бути з'єднані з перемикачами, а вихід – підключений до лампочки. В даному випадку ми маємо справу просто з напругою на входах та виході. Наприклад, вихід одного вентиля може бути входом другого вентиля І.



Ця лампочка загориться лише у разі замикання всіх трьох перемикачів. Тільки якщо будуть замкнуті верхні два перемикачі, вихід першого вентиля І активує перше реле у другому вентилі І. Нижній перемикач активує друге реле у другому вентилі.

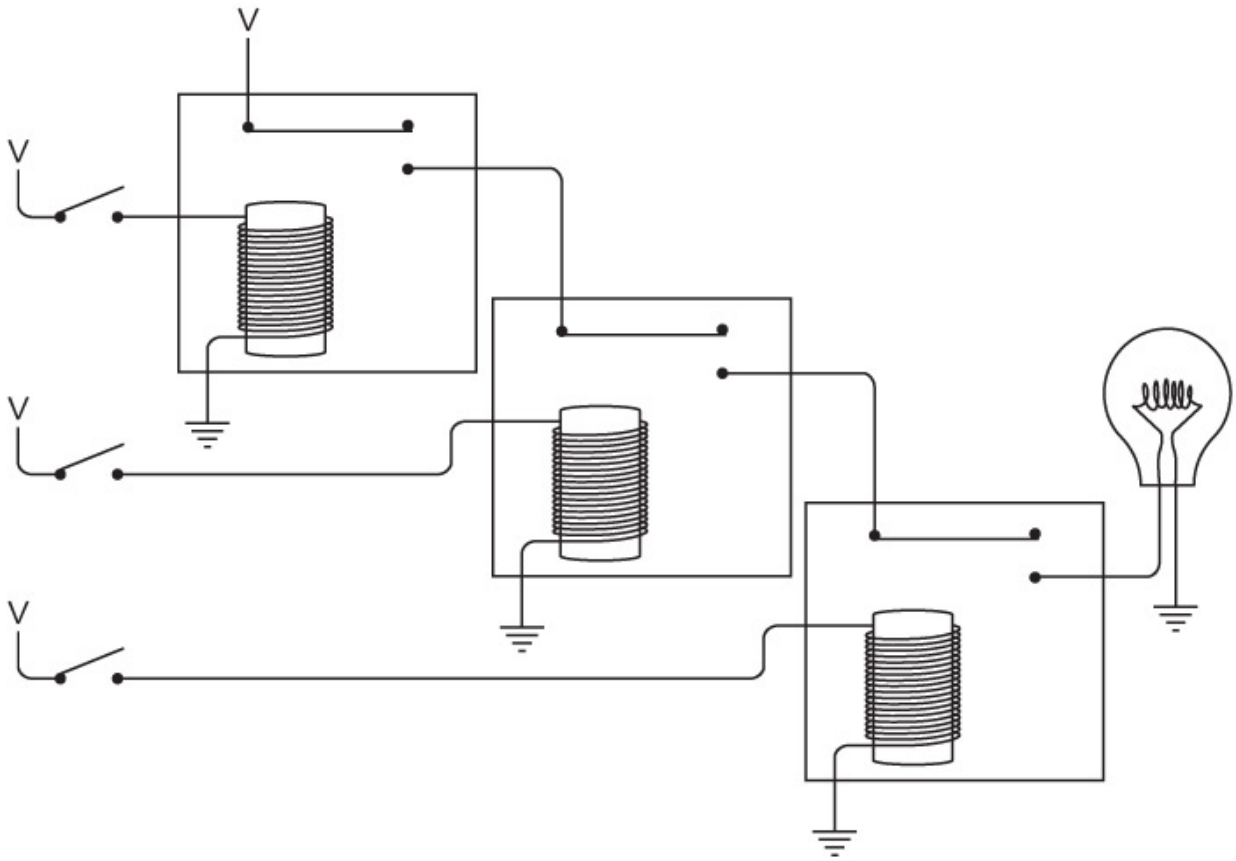
Якщо ми висловимо відсутність напруги у вигляді 0, а її наявність у вигляді 1, то залежність вихідного сигналу вентиля І від вхідних сигналів буде наступною.



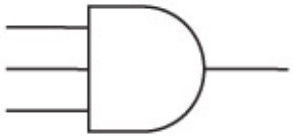
Як і у випадку з двома послідовно з'єднаними перемикачами, роботу вентиля можна описати за допомогою невеликої таблиці.

AND	**0**	**1**
0	0	0
1	0	1

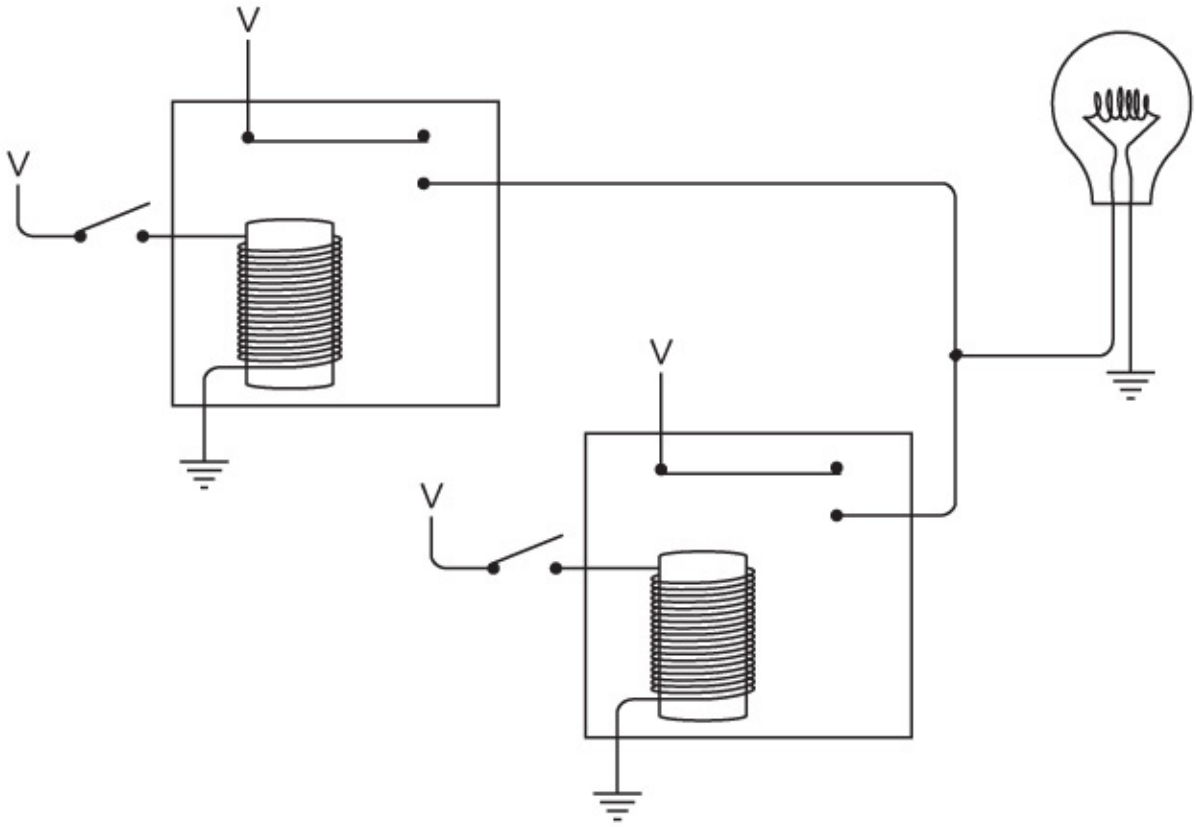
Можна зробити вентиль І з більш ніж двома входами. Наприклад, ви послідовно з'єднали три реле.



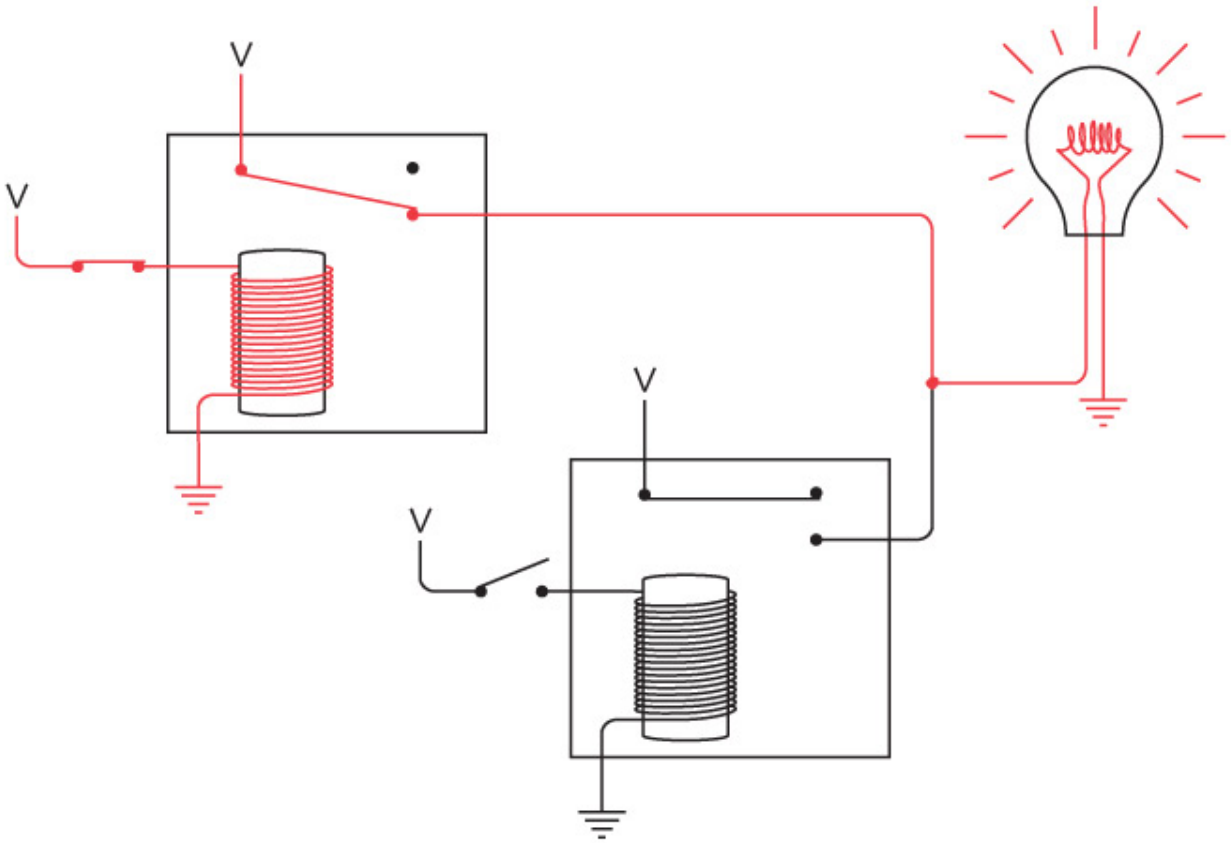
Лампочка спалахує при замиканні всіх трьох перемикачів. Така конфігурація позначається таким символом.



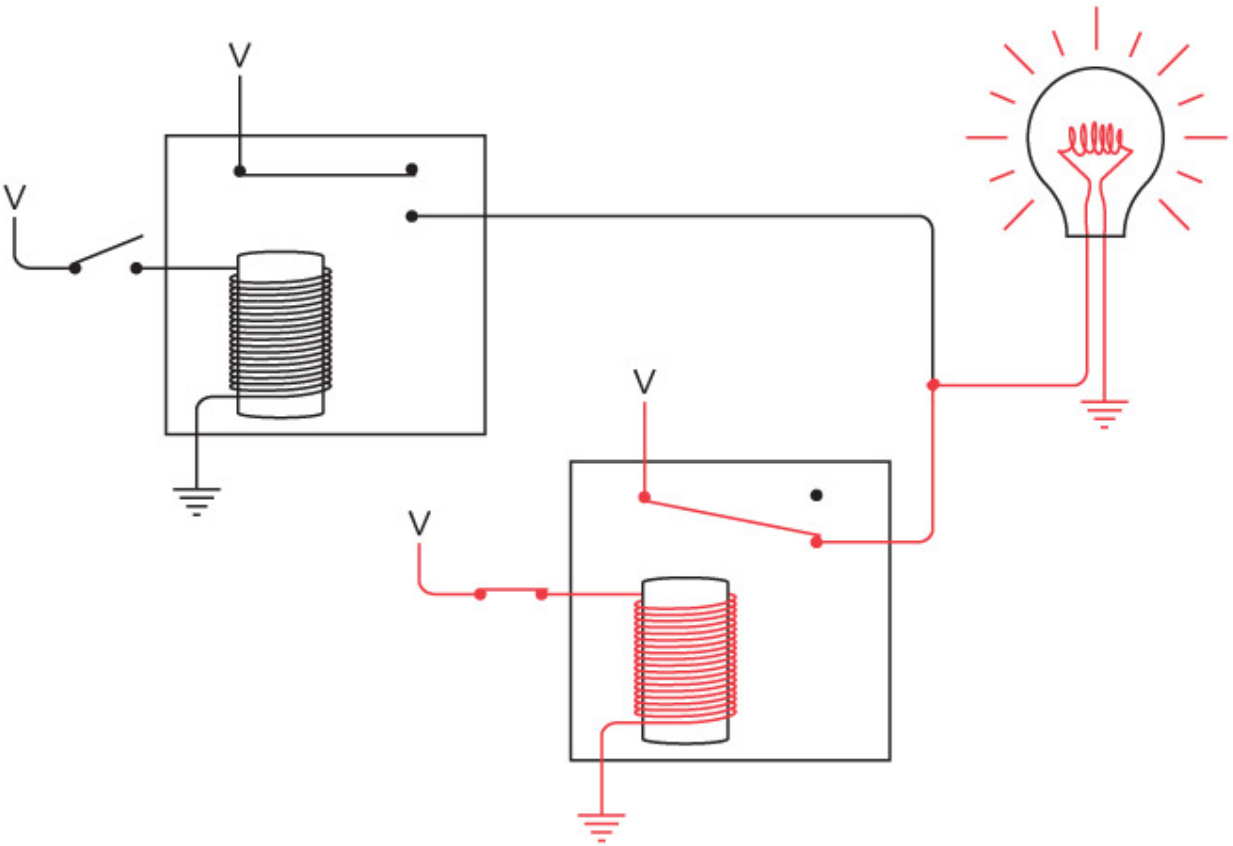
Така схема називається *трьохвходовим вентилем* І.
Наступний логічний вентиль складається з двох реле, з'єднаних паралельно.



Важливо: виходи двох реле з'єднані один з одним. Цей поєднаний вихід подає живлення на лампочку. Для того, щоб лампочка загорілася, достатньо активувати одне з двох реле. Наприклад, якщо ми замкнемо верхній перемикач, лампочка загориться, оскільки отримує живлення від лівого реле.

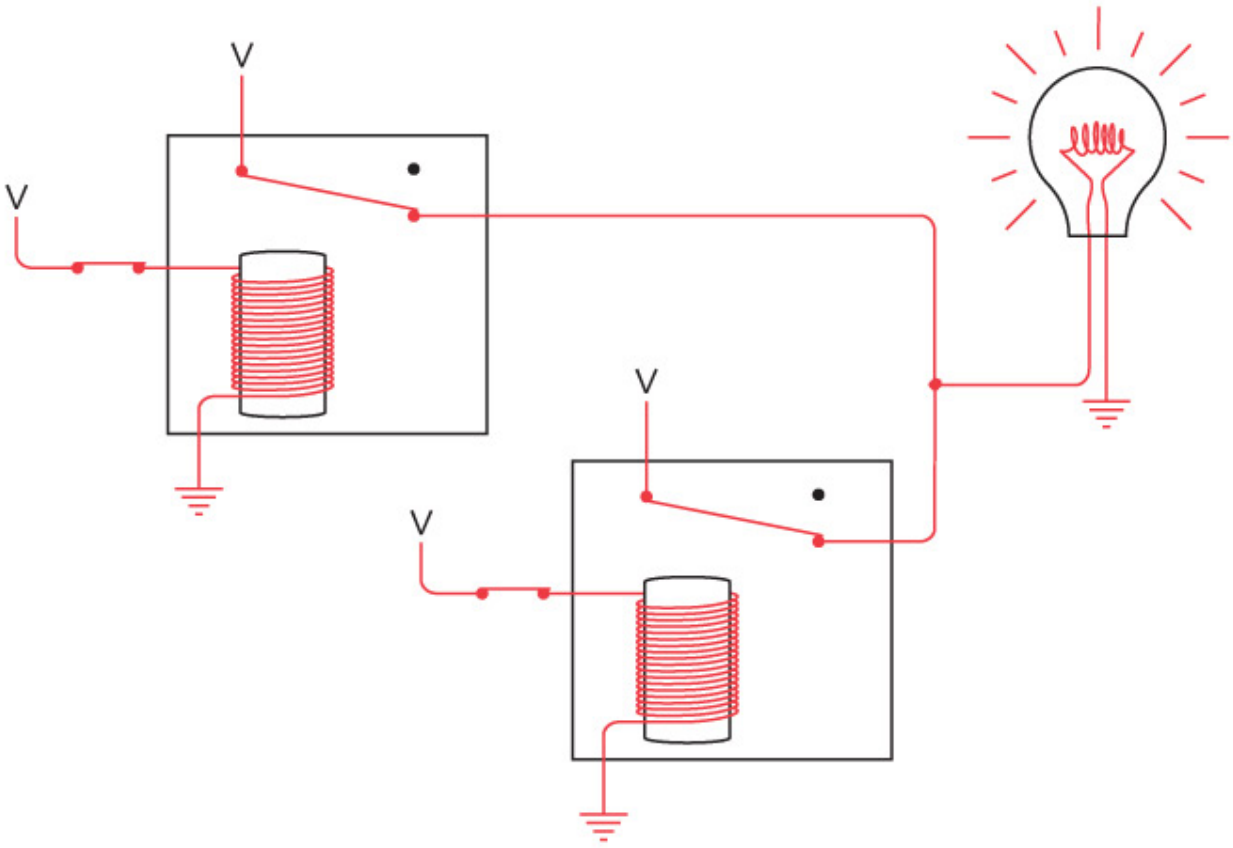


Аналогічно лампочка загориться, якщо ми залишимо верхній вимикач розімкненим, але

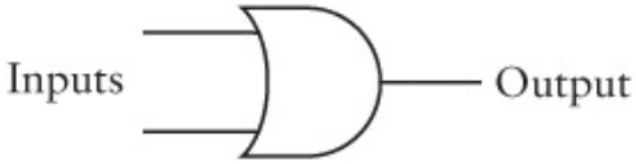


замкнемо нижній.

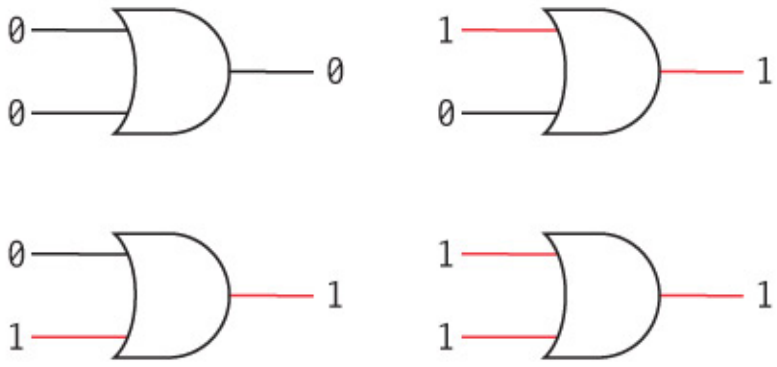
Лампочка також загориться під час замикання обох перемикачів.



В даному випадку ми перебуваємо в ситуації, коли лампочка загоряється при замиканні верхнього *або* нижнього перемикача. Ключовим тут є слово *або,* тому дана схема називається *вентилем АБО.* Для позначення інженери-електрики використовують такий символ.



Він дещо схожий на символ вентиля І, за винятком закруглення боку входів. На виході вентиля АБО є напруга, якщо вона подається на один із двох його входів. Якщо ми позначимо відсутність напруги 0, а наявність – 1, то вентиль АБО зможе перебувати у чотирьох можливих станах.

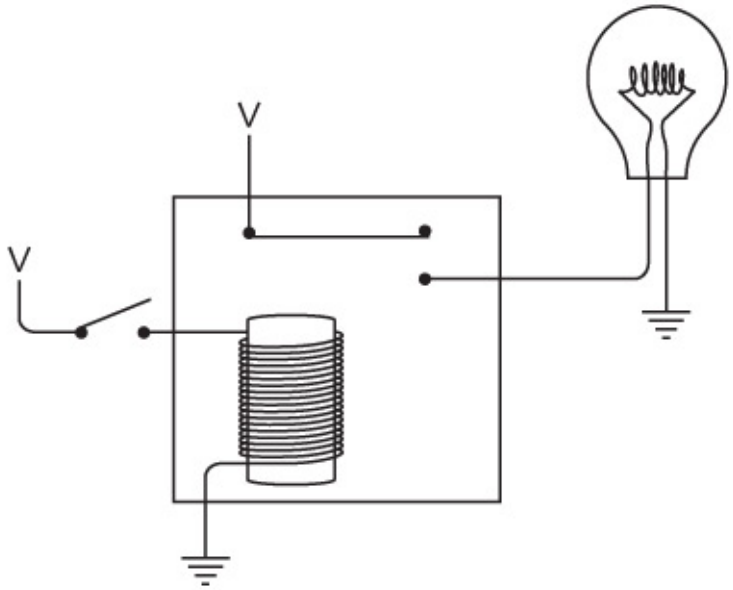


Результати роботи вентиля АБО можна подати у вигляді таблиці.

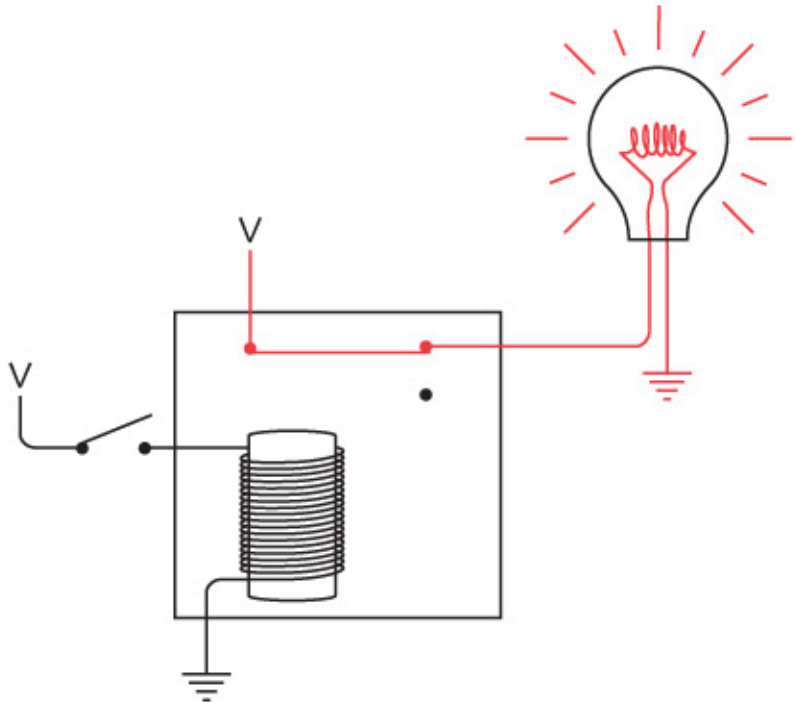
OR	0	1
0	0	1
1	1	1

Вентиль АБО також може мати більше двох входів. Вихід такого вентиля дорівнює 1, якщо будь-який з його входів дорівнює 1; вихід вентиля дорівнює 0, якщо всі його входи дорівнюють 0.

Раніше я пояснив, що реле, які використовуються нами, називаються двопозиційними, тому що їх виходи можуть бути підключені двома різними способами. Як правило, при розімкнутому перемикачі лампочка не горить.

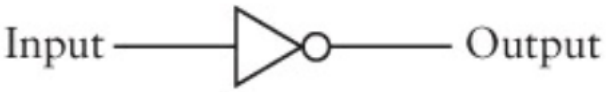


При замиканні перемикача лампочка загоряється.

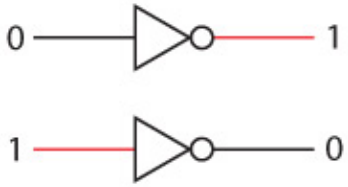


Крім того, можна використовувати інший контакт, щоб лампочка загорялася при *розмиканні* перемикача.

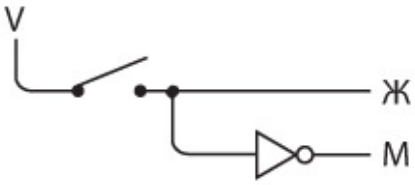
У цьому випадку лампочка гаснутиме при замиканні перемикача. Підключене в такий спосіб одиночне реле називається *інвертором.* Інвертор не є логічним вентилем (логічні вентилі завжди мають два або більше входів), однак він часто виявляється дуже корисним та зображується за допомогою спеціального символу.



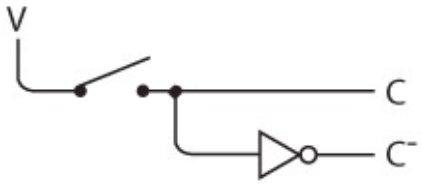
Ця схема називається інвертором, тому що вона інвертує 0 (відсутність напруги) в 1 (наявність напруги) і навпаки.



Тепер, коли у нас є інвертор, вентиль *І* та вентиль АБО, ми можемо приступити до створення пульта керування, який дозволить автоматизувати вибір ідеальної кішки. Почнемо з перемикачів. Перший перемикач у замкнутому стані відповідає кішці, у розімкнутому – коту. Так ми зможемо генерувати два сигнали, які позначимо Ж та М.

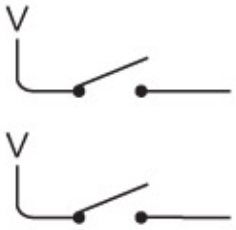


Ж дорівнює 1, М дорівнює 0, і навпаки. Аналогічно другий перемикач відповідає



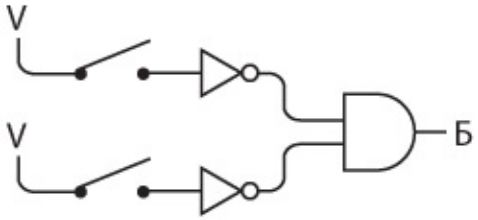
стерилізованій кішці у замкнутому стані, нестерилізованій – у розімкнутому.

З наступними двома перемикачами справа трохи складніша. Різні комбінації повинні



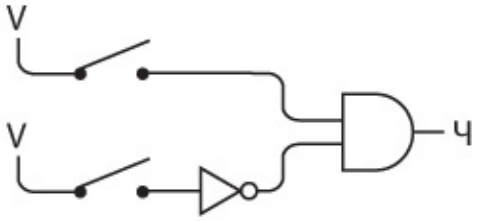
відповідати чотирьом різним кольорам. Ось два перемикачі, підключені до джерела живлення.

Коли обидва перемикачі розімкнені (як показано на малюнку), вони відповідають білому кольору. Ось як можна використовувати два інвертори і один вентиль І для того, щоб згенерувати сигнал, позначений буквою *Б,* який дорівнюватиме 1 (наявність напруги), якщо ви вибрали білу кішку, і 0 (відсутність напруги), якщо не вибрали.



Коли перемикачі розімкнені, входи обох інверторів дорівнюють 0. Таким чином, обидва вихідні сигнали інверторів (які подаються на входи вентиля І) дорівнюють 1. Це означає, що вихід вентиля І дорівнює 1. При замиканні будь-якого з перемикачів вихід вентиля І дорівнюватиме 0.

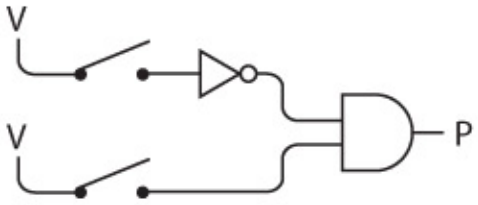
Щоб вибрати чорну кішку, ми замикаємо перший перемикач. Це можна зробити,



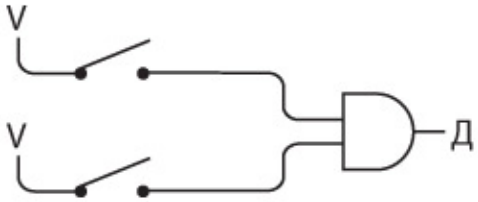
використовуючи один інвертор та вентиль І.

Вихід вентиля І дорівнюватиме 1 тільки в тому випадку, якщо перший перемикач замкнутий, а другий – розімкнутий.

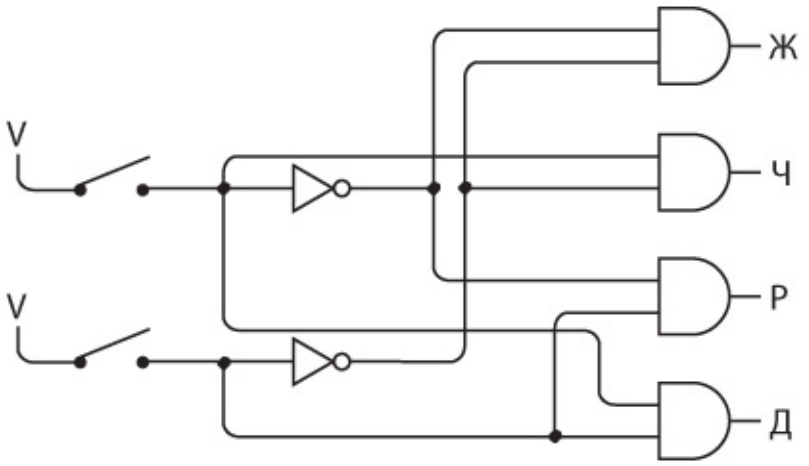
Аналогічно замикання другого перемикача означатиме вибір рудої кішки.



Замикання обох перемикачів означає, що нам потрібна кішка іншого кольору.



Тепер об'єднаємо описані вище чотири невеликі схеми в одну. (Як зазвичай, чорними крапками позначаються з'єднання проводів; проводи, на перетинах яких чорних крапок немає, *не з'єднані.)*



Розумію, у цих хитросплетіннях дротів складно розібратися. Однак якщо ви уважно простежите, звідки подаються сигнали на входи кожного з вентилів І, і проігноруєте те, куди вони йдуть, то побачите, що схема працює. Якщо обидва перемикачі розімкнуті, вихід Б дорівнюватиме 1, а решта – 0. Якщо перший перемикач замкнутий, вихід Ч дорівнюватиме 1, а решта – 0 і т.д.

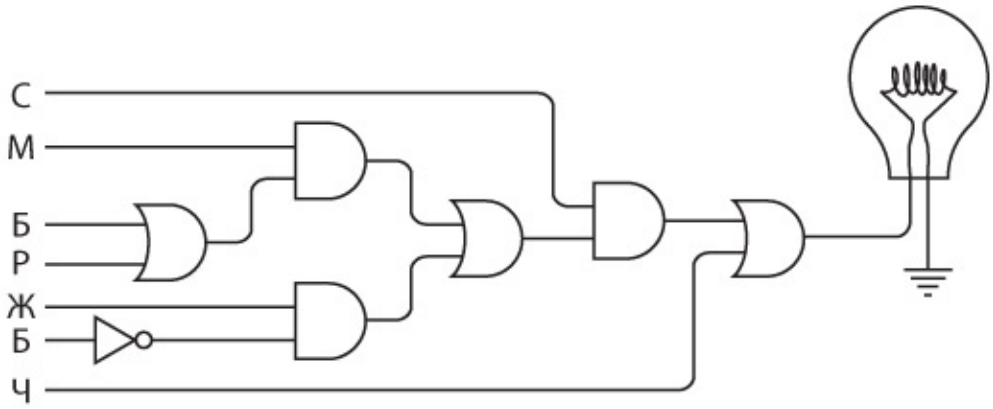
Для з'єднання вентилів та інверторів існує кілька простих правил: вихід одного вентиля (або інвертора) може бути входом одного або кількох інших вентилів (інверторів). Однак виходи двох або більше вентилів (інверторів) ніколи не з'єднуються один з одним.

Ця схема, що складається з чотирьох вентилів І та двох інверторів, називається *дешифратором двох ліній на чотири.* На його вхід подається два біти, які в різних комбінаціях можуть представляти чотири різні значення. На його виході утворюються чотири сигнали, лише один з яких дорівнює 1 у будь-який момент часу (який конкретно залежить від вхідних значень). За аналогічним принципом ви можете створити дешифратор трьох ліній на вісім чи дешифратор чотирьох ліній на шістнадцять тощо.

Ще раз наведу спрощений логічний вираз для вибору кішки:

(С × ((М × (Б + Р)) + (Ж × (1 − Б)))) + Ч.

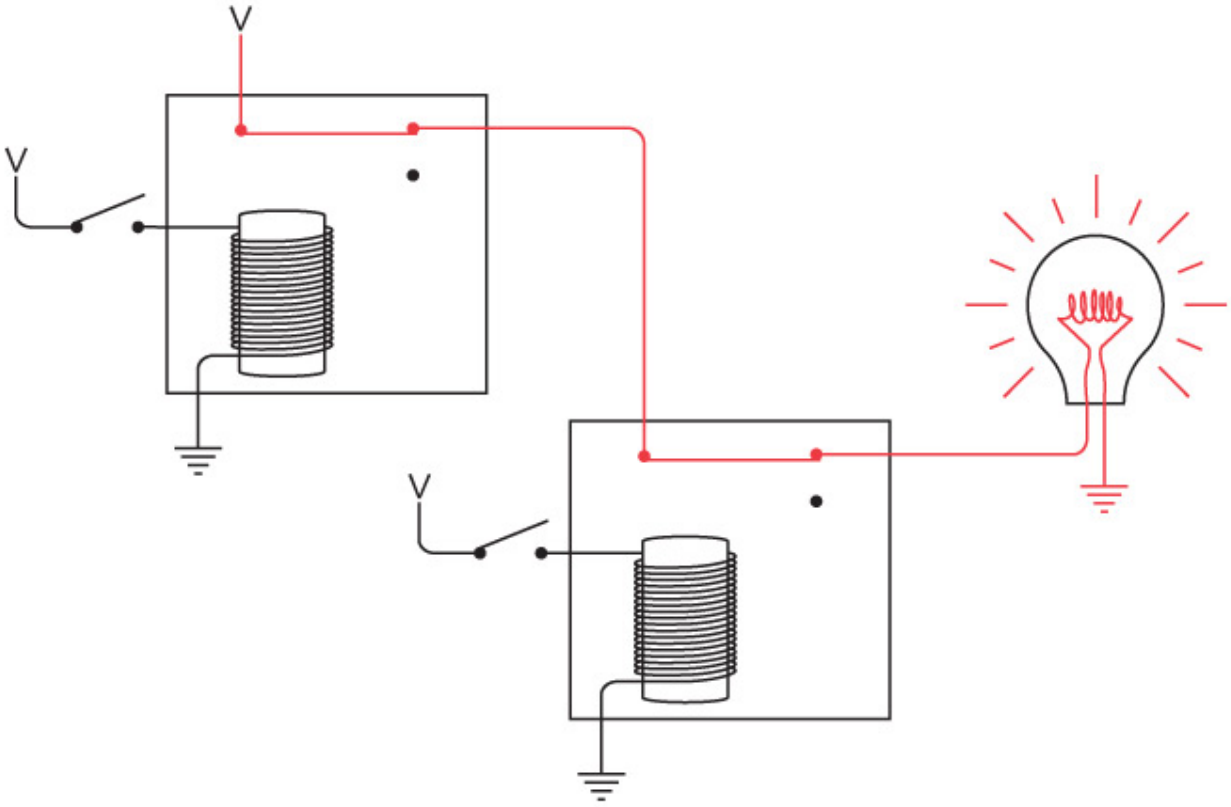
Кожному знаку "+" у цьому виразі повинен відповідати вентиль АБО, а кожному знаку "×" – вентиль І.



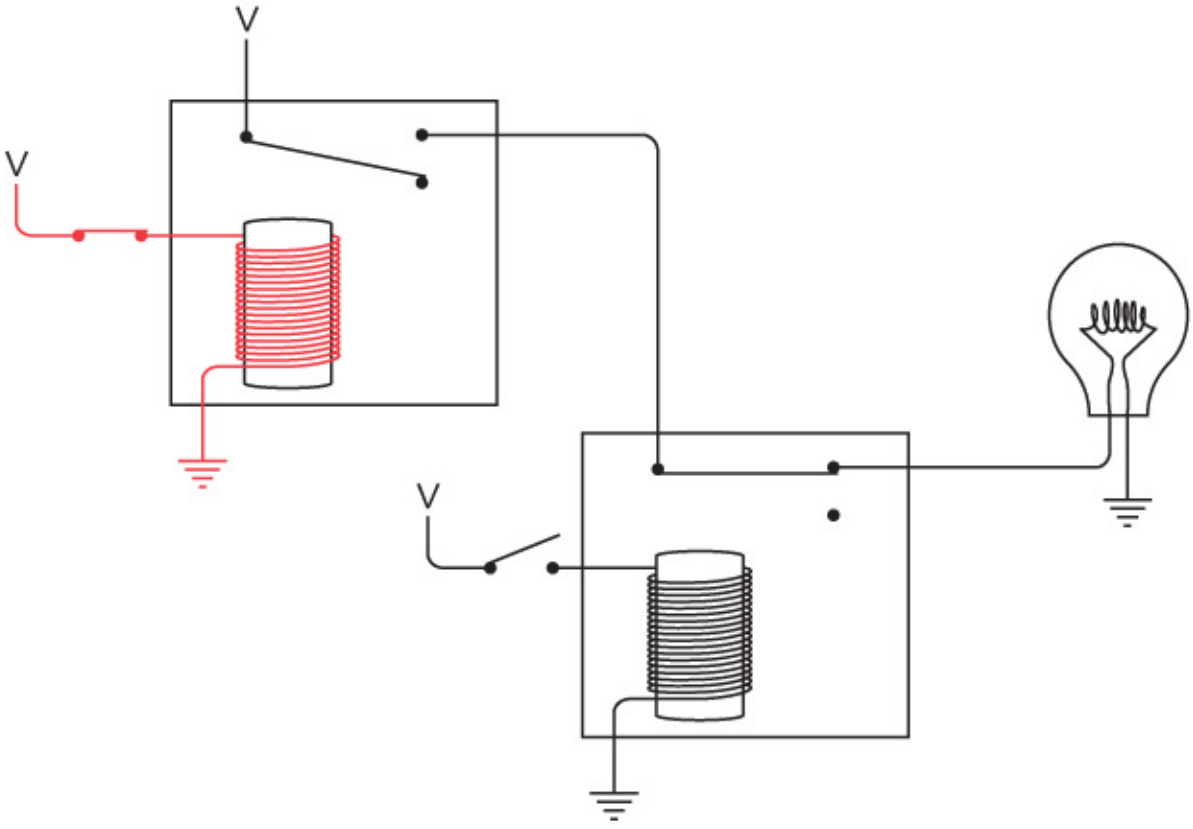
Порядок символів зліва від схеми відповідає їх порядку у виразі. Ці сигнали надходять від перемикачів, з'єднаних з інверторами та дешифратором «2 на 4». Зверніть увагу на використання інвертора для реалізації частини виразу 1 - Б.

Ви можете подумати, що у схемі використовується занадто багато реле, і це справді так. У ній є два реле для кожного вентиля І і АБО і одне реле для кожного інвертора. Можу лише порадити звикнути до такого стану справ. У наступних темах ми будемо використовувати набагато більше реле. Просто радійте, що вам не доведеться купувати реле та збирати ці схеми вдома.

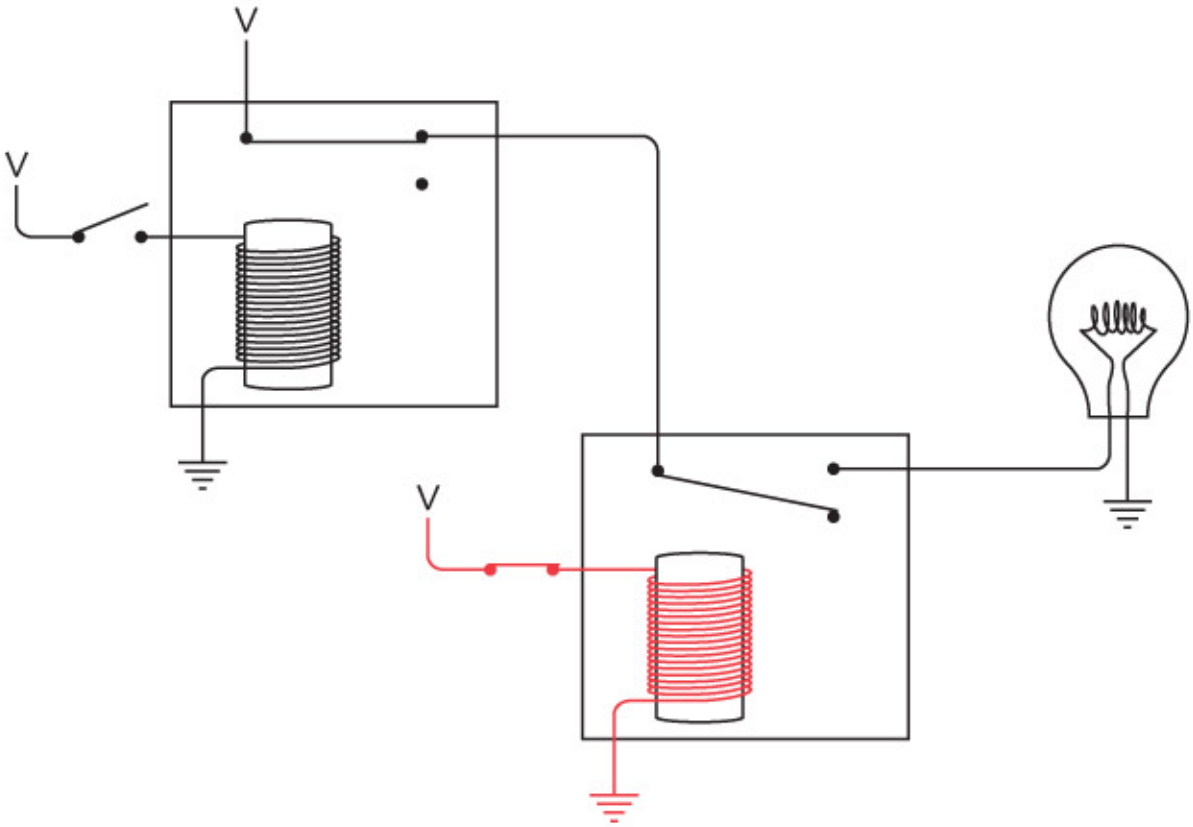
Зараз ми розглянемо ще два логічні вентилі. Обидва використовують вихід реле, на якому є напруга, коли реле не активовано (вихід, що використовується в інверторі). Наприклад, у наступній конфігурації вихід одного реле подає живлення на вхід другого. Коли обидва входи відключені від джерела живлення, лампочка світиться.



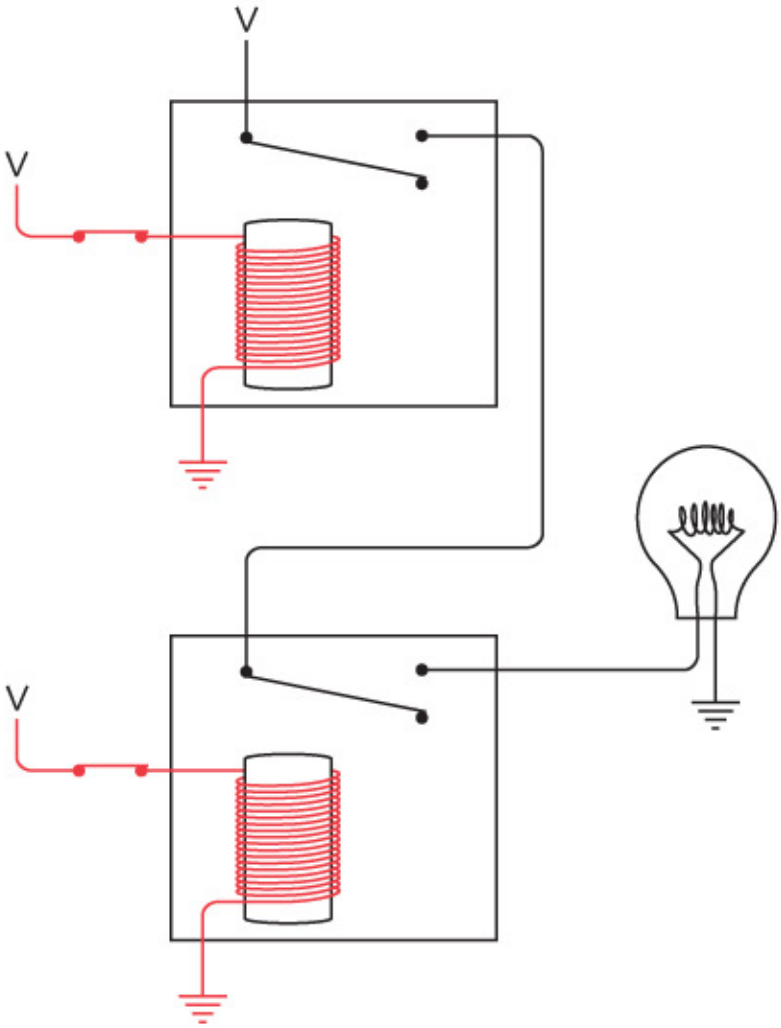
При замиканні верхнього вимикача лампочка гасне.



Лампочка гасне через те, що на друге реле не подається живлення. Так само лампочка гасне при замиканні нижнього вимикача.

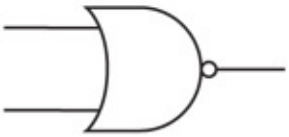


Коли замкнуті обидва перемикачі, лампочка теж не горить.



Ця поведінка прямо протилежна поведінці вентиля АБО. Така схема називається *вентилем АБО-НЕ.*

Його символ аналогічний символу вентиля АБО, за винятком того, що на виході зображено



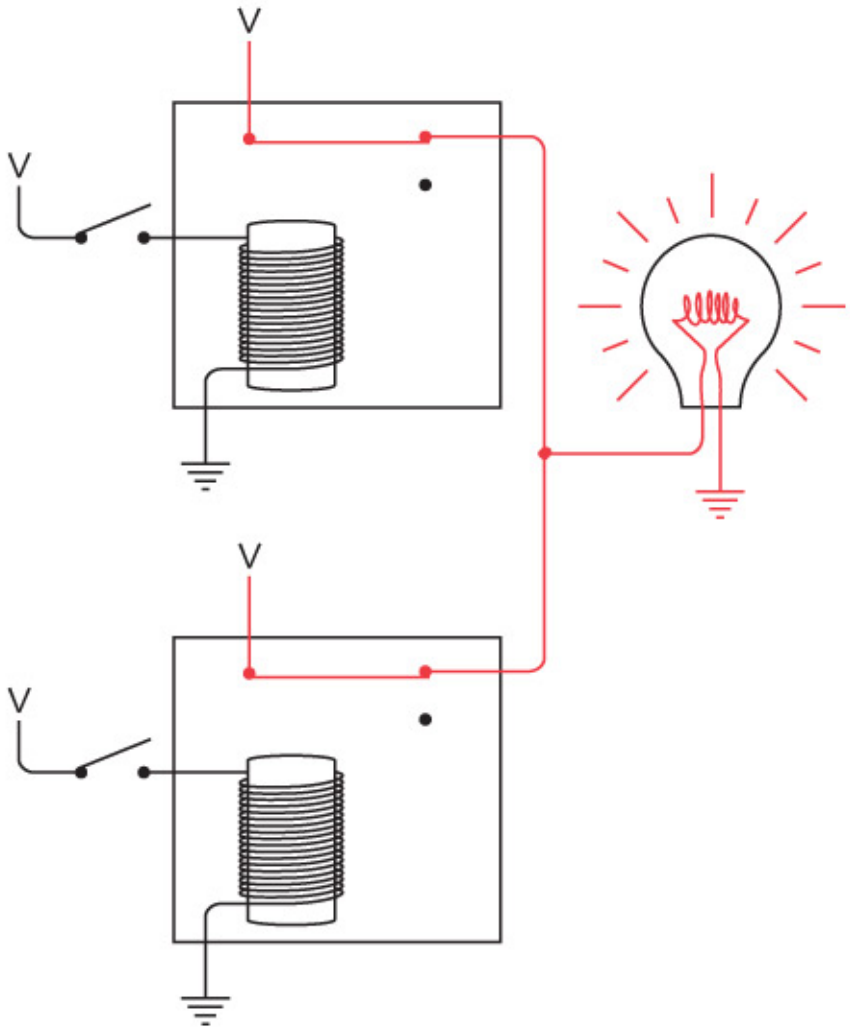
невеликий кружок, що означає *інвертувати.*

Вентиль АБО не відповідає наступній схемі.

Результати роботи вентиля АБО-НЕ представлені в таблиці.

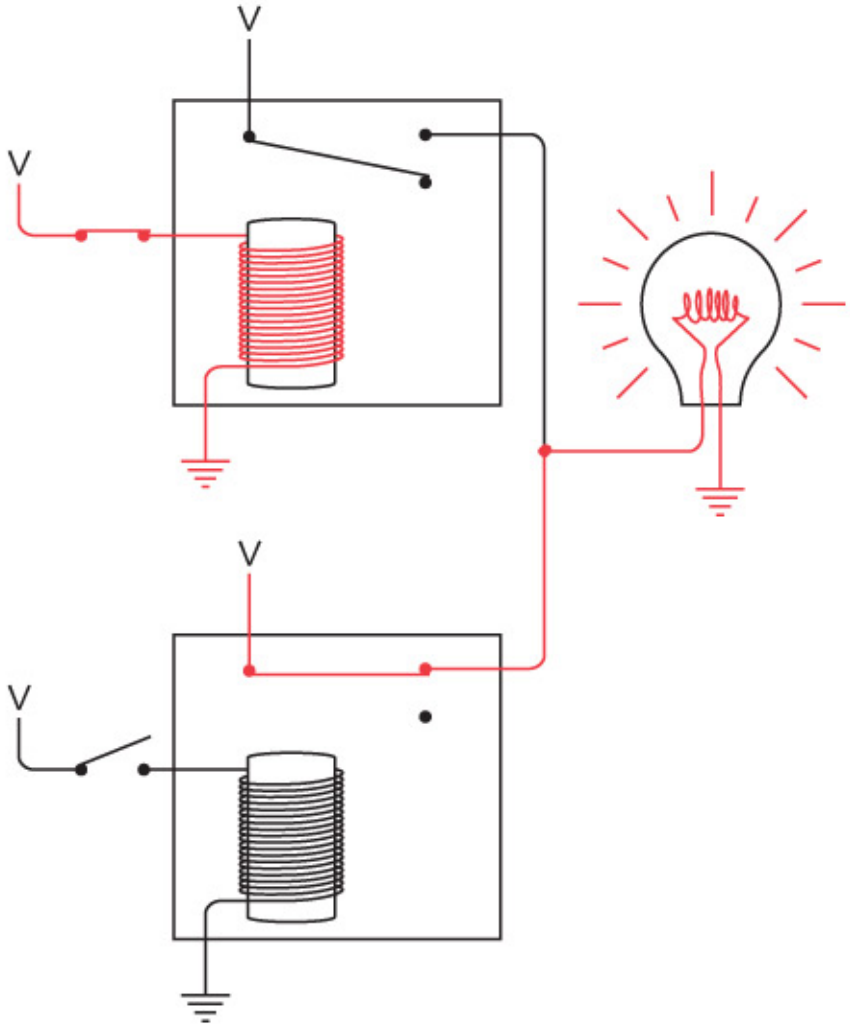
NOR	**0**	**1**
0	1	0
1	0	0

Вони протилежні результатам роботи вентиля АБО, вихід якого дорівнює 1, якщо один із двох його входів дорівнює 1, а 0 – тільки якщо обидва входи дорівнюють 0.



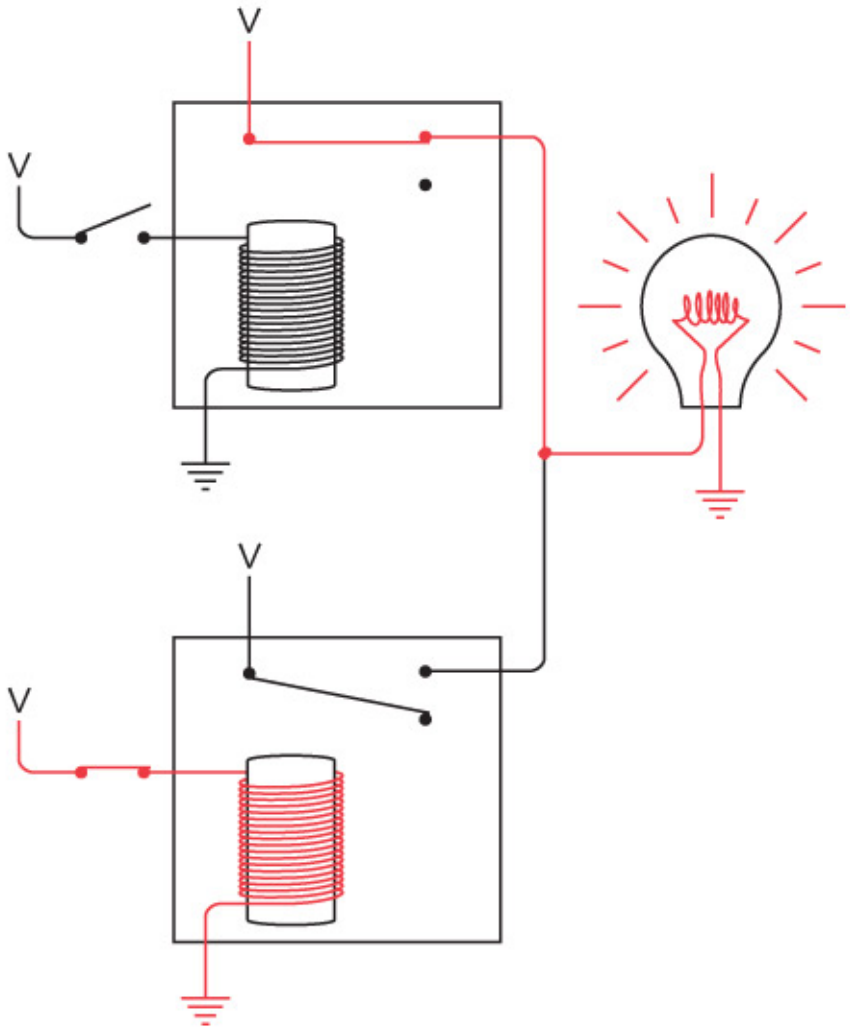
Далі показаний ще один спосіб з'єднання двох реле.

У цьому випадку два виходи з'єднані. Це схоже на конфігурацію вентиля АБО тільки тут використовуються інші контакти. Лампочка горить, коли обидва перемикачі розімкнені.

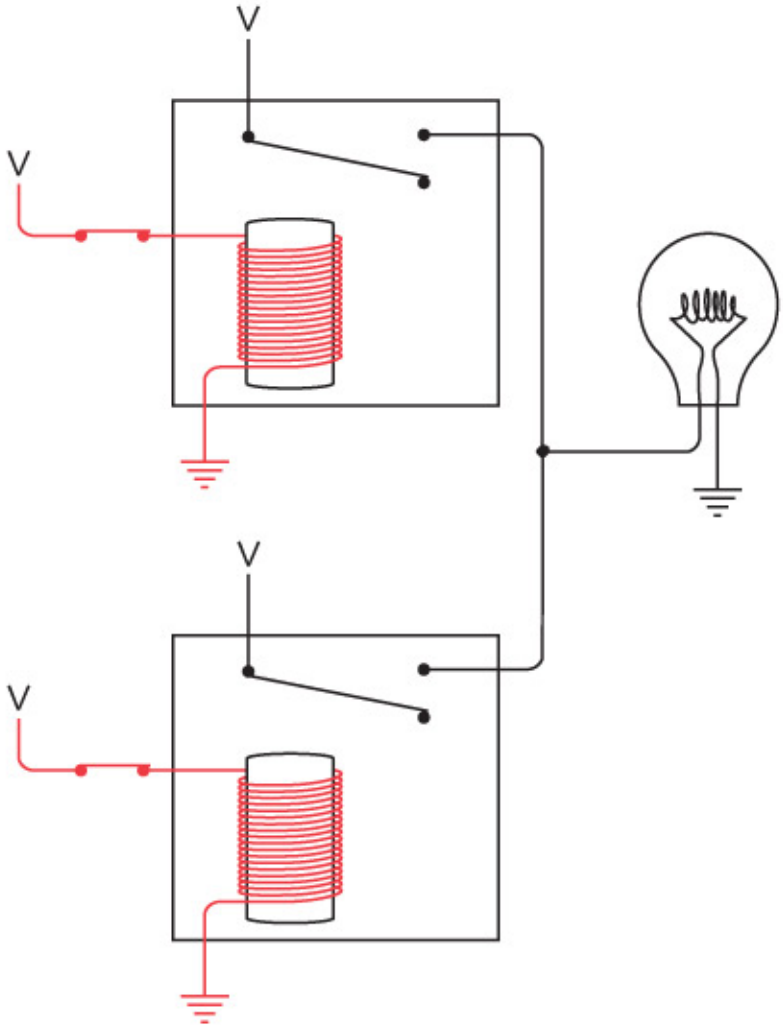


Лампочка продовжує горіти під час замикання верхнього перемикача.

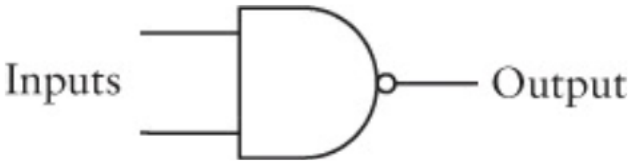
Так само лампочка продовжує горіти і при замиканні нижнього перемикача.



Тільки при замиканні обох перемикачів лампочка гасне.



Ця поведінка прямо протилежна поведінці вентиля І. Така схема називається *вентилем І-НЕ.* Вентиль І-НЕ зображується так само, як і вентиль І, але з кружком на виході, що означає, що вихідний сигнал протилежний вихідному сигналу вентиля І.



Вентиль І-НЕ демонструє таку поведінку.

NAND	**0**	**1**
0	1	1
1	1	0

Зверніть увагу: вихід вентиля І-НЕ протилежний виходу вентиля І. Вихід вентиля І дорівнює 1, тільки якщо обидва входи дорівнюють 1; в іншому випадку вихід дорівнює 0.

Отже, ми розглянули чотири різні способи підключення реле, які мають два входи та один вихід. Кожна конфігурація дає дещо різні результати. Для економії сил і часу ми назвали ці конфігурації логічними вентилями та вирішили позначати їх за допомогою символів, які використовуються інженерами-електриками. Вихідний сигнал конкретного логічного вентиля залежить від вхідного сигналу, як показано в таблицях.

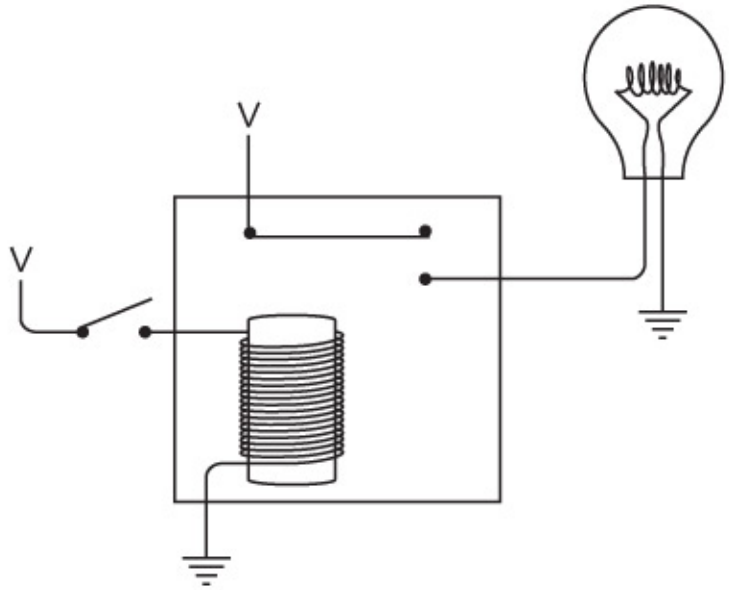
AND	0	1
0	0	0
1	0	1

OR	0	1
0	0	1
1	1	1

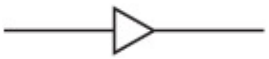
NAND	0	1
0	1	1
1	1	0

NOR	0	1
0	1	0
1	0	0

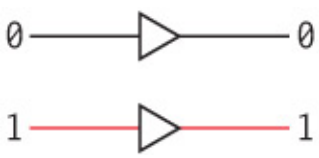
Тепер у нас є чотири логічні вентилі та інвертор. Залишилося доповнити інструментарій звичайним реле, яке називається *буфером.*



Буфер зображується так.



Цей символ подібний до символу інвертора, але без маленького круга. Буфер примітний тим,

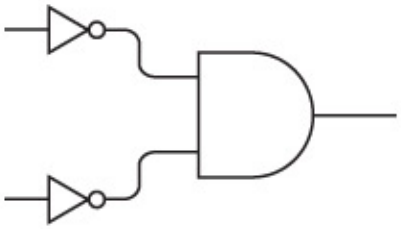


що майже нічого не робить. Вихідний сигнал буфера збігається з вхідним сигналом.

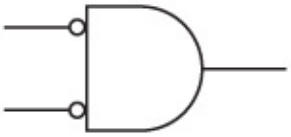
Однак, ви можете використовувати буфер за наявності слабкого вхідного сигналу. Як ви пам'ятаєте, саме через це реле використовувалися в телеграфній системі багато років тому. Крім того, буфер можна застосовувати для невеликої затримки сигналу. Справа в тому, що для спрацьовування реле потрібно небагато часу – невелика частка секунди.

Відтепер у далі рідко зустрічатимуться зображення реле. Натомість наступні схеми будуть складатися з буферів, інверторів, чотирьох основних логічних вентилів і складніших схем (дешифратора «2 на 4», наприклад), зібраних із цих вентилів. Зрозуміло, всі ці компоненти складаються з реле, проте немає необхідності їх розглядати.

Раніше коли ми конструювали дешифратор «2 на 4», нам зустрілася невелика схема наступного типу.



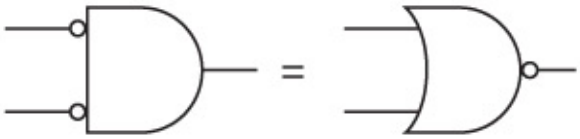
Два інвертовані входи стали входами вентиля І. Іноді така конфігурація зображується без



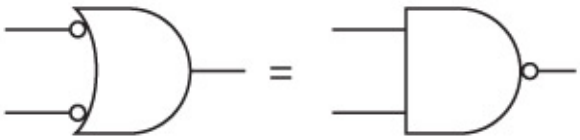
інверторів.

Зверніть увагу на маленькі кружки на вході вентиля І, які вказують, що сигнали в цій точці інвертуються: 0 (відсутність напруги) стає 1 (наявність напруги) і навпаки.

Вентиль І з двома інвертованими входами поводиться так само, як вентиль АБО-НЕ.



Вихід дорівнює 1, тільки якщо обидва входи дорівнюють 0.



Аналогічно вентиль АБО з двома інвертованими входами еквівалентний вентилю І-НЕ.

Вихід дорівнює 0, тільки якщо обидва входи дорівнюють 1.

Ці дві пари еквівалентних схем представляють електричне втілення законів Огастеса де Моргана, ще одного математика вікторіанської епохи, який був на дев'ять років старший за Буля. Його книга "Формальна логіка" була опублікована в 1847 році, згідно з переказами, в один день з книгою Буля "Математичний аналіз логіки". Насправді на заняття логікою, Буля спонукала відкрита ворожнеча між де Морганом та іншим британським математиком, пов'язана зі звинуваченнями у плагіаті (історія виправдала де Моргана). З самого початку де Морган усвідомив важливість прозріння Буля. Він безкорисливо заохочував Буля і допомагав йому в дослідженнях, проте сьогодні він, на жаль, майже забутий, а в пам'яті нащадків залишилися лише його славетні закони.

Закони де Моргана найпростіше висловити в такий спосіб.

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

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

A і B – два булеві операнди. У першому виразі вони інвертуються, а потім об'єднуються за допомогою булевого оператора І. Це еквівалентно об'єднанню двох операндів за допомогою булевого оператора АБО і подальшого інвертування результату (відповідає оператору АБО-НЕ). У другому виразі два оператори інвертуються, а потім об'єднуються за допомогою булевого оператора АБО. Це еквівалентно об'єднанню двох операндів за допомогою булевого оператора І та подальшого інвертування результату (відповідає оператору І-НЕ).

Закони де Моргана – важливий інструмент спрощення булевих висловів, отже, спрощення схем. Історія показала, що у цьому полягало значення роботи Клода Шеннона для інженерів-електриків. Однак нам важливо зібрати працюючу схему, а не зробити так, щоб вона працювала якомога простіше. Цим ми і займемося в наступній темі – зберемо лічильну машину, що працює.

Тема 8. Двійковий суматор

Додавання – найпростіша арифметична операція. Якщо ми хочемо створити комп'ютер (а саме в цьому полягає наша мета), спочатку потрібно знайти спосіб створення пристрою, що додає два числа. По суті, комп'ютери виконують лише операцію додавання. Якщо нам вдасться сконструювати механізм, що вміє додавати, ми виявимося здатні створити пристрій, який використовує операцію додавання для того, щоб віднімати, множити, ділити, розраховувати платежі по іпотеці, відправляти ракети на Марс, грати в шахи і вносити плутанину в телефонні рахунки.

Суматор, який ми збудуємо, буде великим, нескладним, повільним і галасливим порівняно із сучасними калькуляторами та комп'ютерами. Найцікавіше полягає в тому, що ми зберемо цю машину з простих електричних пристроїв, про які говорили в попередніх темах, – перемикачів, лампочок, дротів, батарейки та реле, об'єднаних у різні логічні вентилі. Цей суматор складатиметься виключно з деталей, які вже були винайдені 120 років тому. Особливо добре те, що нам не потрібно нічого збирати у своїй вітальні; натомість ми можемо конструювати на папері та в пам’яті.

Ця машина працюватиме виключно з двійковими числами, в ній будуть відсутні деякі сучасні функції. Ви не зможете використовувати клавіатуру для введення чисел, що підлягають додаванню; натомість буде низка перемикачів. Роль дисплея для відображення результатів у цьому суматорі виконає низка лампочок.

Але машина зможе додати два числа, і вона зробить це практично як комп'ютер.

Додавання двійкових чисел схоже на додавання десяткових. Якщо хочете додати два десяткові числа, наприклад 245 і 673, ви розбиваєте завдання на простіші етапи. На кожному етапі додаєте дві десяткові цифри. У цьому прикладі починаєте із додавання 5 і 3. Це завдання вирішується швидше, якщо ви знаєте таблицю додавання.

Велика різниця між додаванням десяткових і двійкових чисел у тому, що у випадку з двійковими числами використовується простіша таблиця.

+	**0**	**1**
0	0	1
1	1	10

Якщо ви виросли серед дельфінів, ймовірно, ви в школі вчили цю таблицю, голосно вимовляючи:

0 плюс 0 дорівнює 0,
0 плюс 1 дорівнює 1,
1 плюс 0 дорівнює 1,
1 плюс 1 дорівнює 0, 1 в пам’яті.

Ви можете додати до цієї таблиці нулі так, щоб кожен результат представляв 2-бітове

+	**0**	**1**
0	00	01
1	01	10

значення.

Таким чином, результатом додавання пари двійкових чисел є два біти, які називаються *розрядом суми* та *розрядом перенесення* (1 плюс 1 дорівнює 0, 1 в пам'яті). Тепер ми можемо розділити таблицю додавання двійкових чисел на дві таблиці. Перша – для розряду суми.

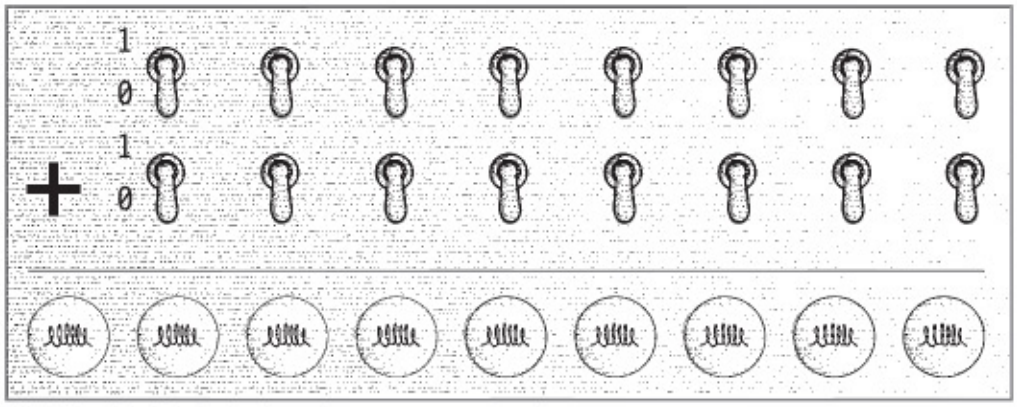
+ sum	0	1
0	0	1
1	1	0

Друга – для розряду перенесення.

+ carry	0	1
0	0	0
1	0	1

Додавання двійкових чисел зручно розглядати так, оскільки наш суматор виконує операції додавання та перенесення окремо. Для створення двійкового суматора потрібно сформулювати схему, що виконує ці операції. Робота виключно в двійковій системі числення значно спрощує завдання, оскільки всі частини схеми – перемикачі, лампочки та дроти – можуть представляти двійкові цифри.

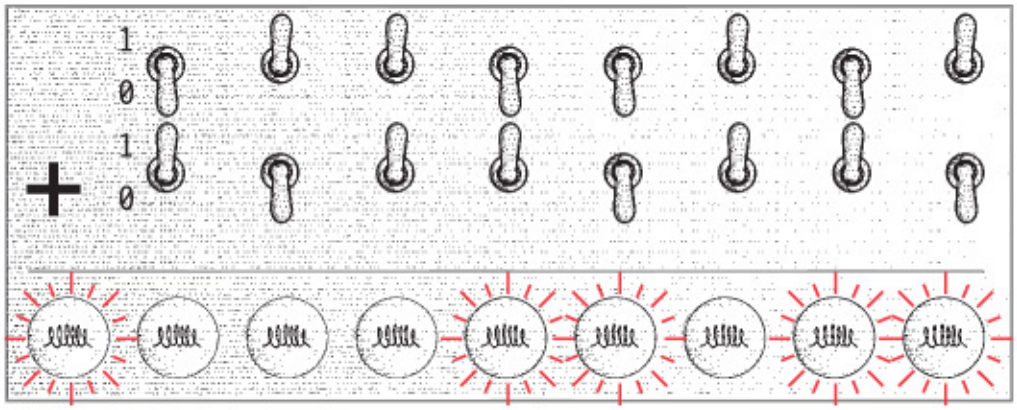
Як і при додаванні десяткових чисел, ми додаємо двійкові числа стовпець за стовпцем, починаючи з крайнього правого.



Зверніть увагу: при додаванні значень у третьому стовпці праворуч 1 переноситься в наступний стовпець. Це відбувається знову в шостому, сьомому та восьмому стовпцях праворуч.

Якого розміру двійкові числа хочемо додати? Оскільки ми створюємо суматор прямо в думці, то можемо зробити так, щоб він складав дуже довгі числа. Однак давайте будемо розсудливими і обмежимося двійковими числами довжиною до восьми біт, тобто додаватимемо двійкові числа в діапазоні від 0000 0000 до 1111 1111 (десяткові від 0 до 255). Сума двох 8-бітових чисел може досягати двійкового значення 111111110 (десяткового значення 510).

Пульт керування нашим двійковим суматором може мати такий вигляд.



На цьому пульті є два ряди по вісім перемикачів. Цей набір перемикачів – пристрій введення, який ми будемо використовувати для введення двох 8-бітових значень. У цьому пристрої вимкнений перемикач (положення вниз) відповідає 0, а включений (положення вгору) – 1, як у випадку з настінними перемикачами у вашому будинку. Пристрій виведення в нижній частині пульта – ряд дев'яти лампочок, які відобразять результат додавання. Лампочка без світла відповідає 0, світить – 1. Нам потрібно дев'ять лампочок, оскільки сума двох 8-бітних чисел може бути 9-бітним числом.

В іншому суматор складатиметься з логічних вентилів, з'єднаних різними способами. Перемикачі активуватимуть реле в логічних вентилях, які, у свою чергу, запалюватимуть потрібні лампочки. Наприклад, якщо ми хочемо додати числа 0110 0101 та 1011 0110 (з попереднього прикладу), включаємо відповідні перемикачі.

Лампочки, що загорілися, показують результат: 1 0001 1011. Принаймні, ми на це сподіваємося. Адже ми ще не зібрали пристрій!

У попередній темі я згадав, що ми будемо використовувати безліч реле. Для 8-бітного суматора, який ми створюємо, потрібно не менше 144 реле – по вісімнадцять для кожної з восьми пар бітів, які додаємо. Якби я показав готову схему, ви напевно злякалися б. Нікому не під силу розібратися у схемі, що складається зі ста сорока чотирьох хитро з'єднаних реле. Натомість ми вирішуватимемо таке завдання поетапно, використовуючи логічні вентилі.

Можливо, ви відразу помітили зв'язок між логічними вентилями та додаванням двійкових чисел, коли побачили таблицю для розряду переносу, який виникає в результаті додавання двох однобітних чисел.

+ carry	0	1
0	0	0
1	0	1

Ймовірно, ви помітили в ній результат роботи вентиля I.

AND	0	1
0	0	0
1	0	1

Таким чином, вентиль I обчислює значення розряду перенесення при додаванні двох двійкових цифр.

Ага! Ми безперечно робимо успіхи. Наш наступний крок, схоже, полягає в тому, щоб переконати деякі реле поводитися так:

+ sum	0	1
0	0	1
1	1	0

Це друга частина завдання при додаванні пари двійкових цифр. Обчислити значення розряду суми виявляється не так просто, як значення розряду перенесення, але ми впораємося і з цією складністю.

Перше, що потрібно зрозуміти, – це те, що результат роботи вентиля АБО близький до того, що нам потрібно, за винятком значення в нижньому правому кутку.

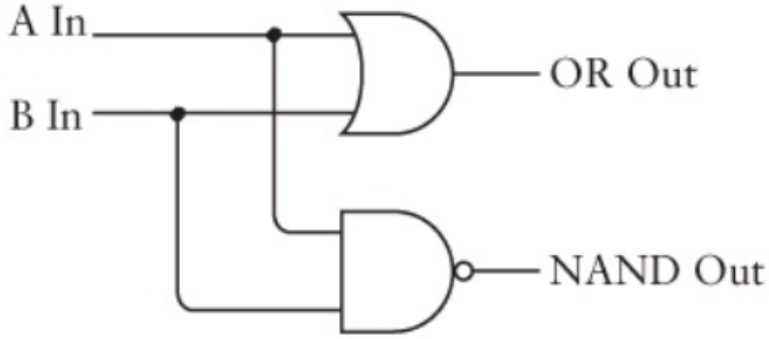
OR	0	1
0	0	1
1	1	1

Результат роботи вентиля І-НЕ також близький до того, що нам потрібно, за винятком значення у верхньому лівому куті:

NAND	0	1
0	1	1
1	1	0

A In	B In	OR Out	NAND Out	What we want
0	0	0	1	0

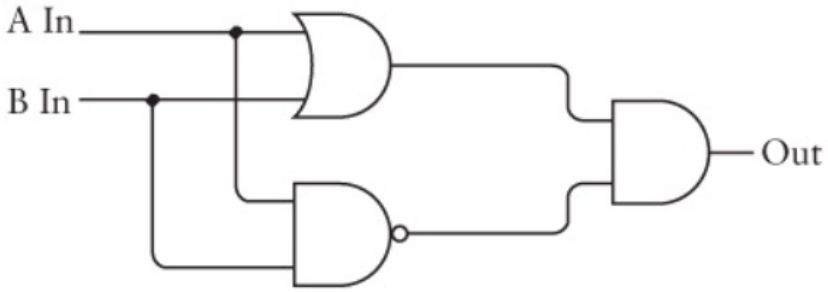
Отже, давайте підключимо вентилі АБО та І-НЕ до одних і тих же входів.



У наступній таблиці представлені вихідні сигнали вентилів АБО та І-НЕ та їх порівняння з тим, що ми хочемо отримати від суматора.

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

Зауважте, що ми хочемо отримати значення 1, тільки якщо вихідні сигнали обох вентилів АБО та І-НЕ рівні 1. Це говорить про те, що ці два вихідні сигнали можуть бути вхідними сигналами для вентиля І.

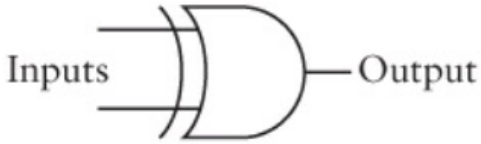


Те що потрібно.

Зверніть увагу: у всій цій схемі, як і раніше, є тільки два входи і один вихід. Два входи відносяться до обох вентилів АБО та І-НЕ. Вихідні сигнали вентилів АБО та І-НЕ подаються на вхід вентиля І, і це дає саме той результат, якого ми прагнемо.

A In	B In	OR Out	NAND Out	AND Out
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0

Насправді у цієї схеми є назва: *вентиль, що виключає АБО* (Викл-АБО, воно ж – додавання по модулю 2). Вона називається так тому, що вихід дорівнює 1, якщо вхід А дорівнює 1 *або* вхід В дорівнює 1, але не обидва одночасно. Замість малювати вентилі АБО, І-НЕ та І, ми можемо використовувати позначення, яким інженери-електрики показують вентиль Викл-АБО.



Це позначення дуже схоже на позначення вентиля АБО, але має додаткову криву лінію з боку входу.

XOR	0	1
0	0	1
1	1	0

Вентиль Викл-АБО – це останній логічний елемент, який тут докладно описаний. Іноді в електротехніці використовується шостий вентиль, що називається *вентилем збігу* або *еквівалентності,* оскільки вихід дорівнює 1 тільки за однакових сигналів на вході. Вентиль збігу на виході діє протилежно вентилю Викл-АБО, тому його позначення аналогічне позначення вентиля Викл-АБО, але доповнено кружком з боку виходу.

Давайте повторимо все, що знаємо. При додаванні двох двійкових чисел виходить біт суми і біт перенесення.

+ sum	0	1
0	0	1
1	1	0

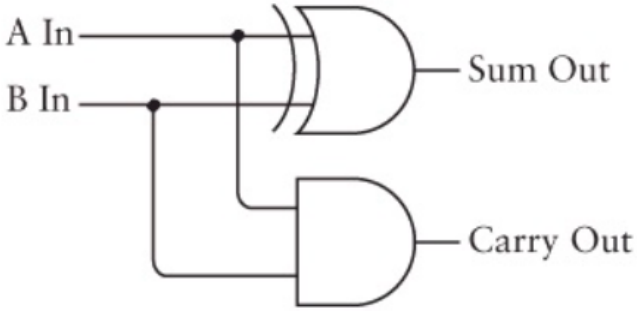
+ carry	0	1
0	0	0
1	0	1

Для отримання цих результатів можна використовувати наступні два вентилі:

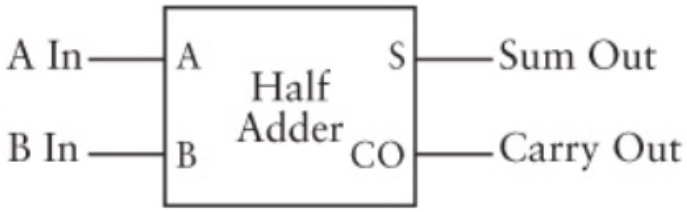
XOR	0	1
0	0	1
1	1	0

AND	0	1
0	0	0
1	0	1

Розряд суми двох двійкових чисел задається виходом вентиля Викл-АБО, а розряд перенесення – виходом вентиля І, тому можна комбінувати вентилі І та Викл-АБО для додавання двох двійкових цифр А і В.



Замість багаторазового перемальовування вентилів І та Викл-АБО можна просто намалювати схему, подібну до наступної.

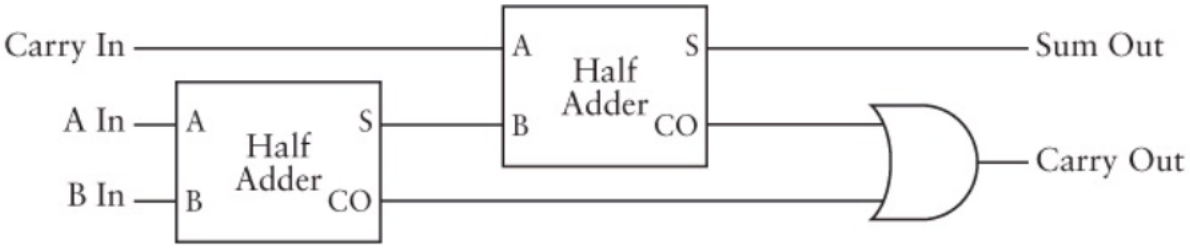


Існує причина, через яку ця схема називається *напівсуматором.* Зрозуміло, вона додає дві двійкові цифри та видає біт суми та біт перенесення. Однак довжина переважної більшості двійкових чисел перевищує один біт. Те, що напівсуматор не може зробити, так це додати можливий біт перенесення, що вийшов у результаті попередньої операції додавання. Уявіть, що додаємо два двійкові числа.

$$\begin{array}{r} 1111 \\ +\ 1111 \\ \hline 11110 \end{array}$$

Ми можемо використовувати напівсуматор тільки для додавання цифр у правому крайньому стовпці: 1 плюс 1 дорівнює 0, 1 переноситься. У разі другого стовпця праворуч нам, по суті, потрібно додати *три* двійкові цифри через перенесення. І це стосується решти стовпців. Кожна наступна операція додавання двох двійкових цифр може включати біт перенесення з попереднього стовпця.

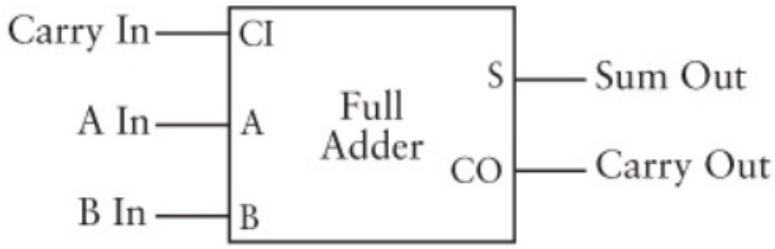
Для додавання трьох двійкових цифр знадобляться два напівсуматори та вентиль АБО, з'єднані наступним чином.



Щоб розібратися в цій схемі, почнемо зі входів А та В першого напівсуматора зліва. Результат – біт суми та біт перенесення. Ця сума повинна бути додана до перенесення з попереднього стовпця, тому вони є входами другого напівсуматора. Сума, отримана від другого напівсуматора, – остаточна. Два перенесення з напівсуматорів – входи для вентиля АБО. Може здатися, що тут потрібен другий напівсуматор і така схема, безумовно, спрацювала б. Однак якщо ви

проаналізуєте всі можливості, то виявите, що *обидва* перенесення з двох напівсуматорів ніколи не рівні 1. Вентиля АБО достатньо для їх додавання, оскільки він діє так само, як вентиль Викл-АБО, якщо обидва вхідні сигнали одночасно не рівні 1.

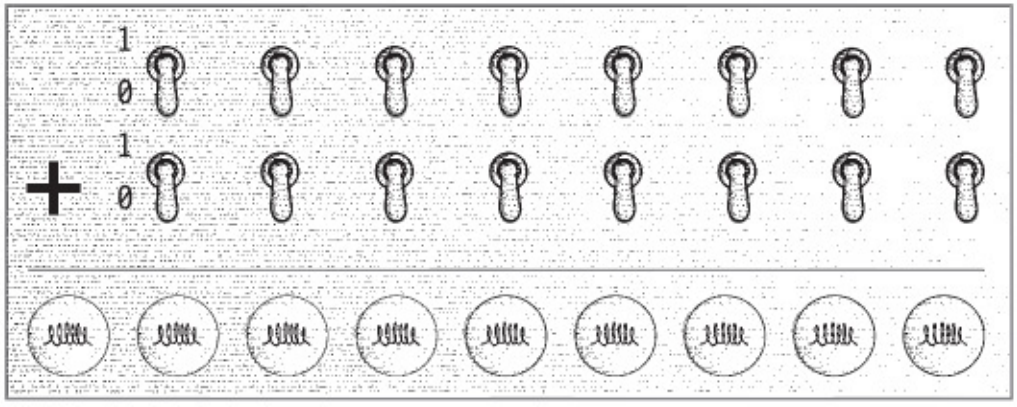
Замість багаторазового перемальовування цієї схеми можемо назвати її *повним суматором.*



У наведеній нижче таблиці представлені всі можливі комбінації входів для повного суматора і результуючі виходи.

A In	B In	Carry In	Sum Out	Carry Out
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

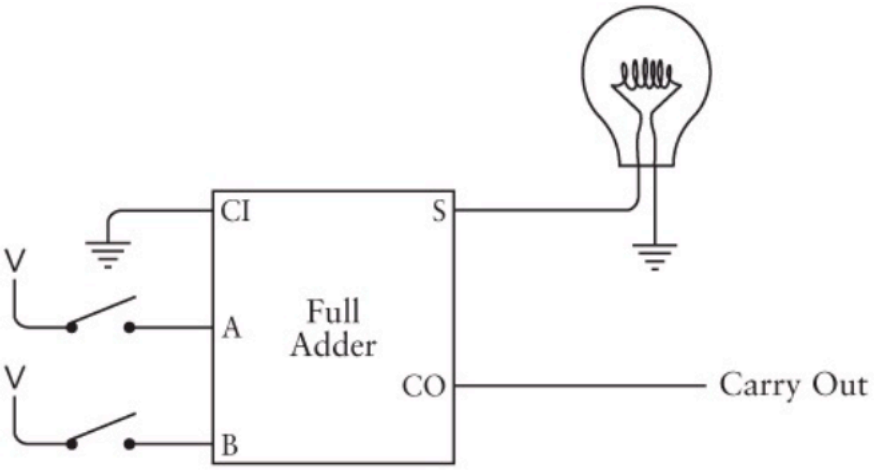
На початку цієї теми я сказав, що для створення суматора знадобиться 144 реле. Ось як я це зрозумів: для кожного вентиля І, АБО та І-НЕ потрібні по два реле. Таким чином, вентиль Викл-АБО складається з шести реле. Напівсуматор – це вентиль Викл-АБО та вентиль І, тому для його створення необхідні вісім реле. Кожен повний суматор – два напівсуматори та вентиль АБО, тобто 18 реле. Нам потрібні вісім повних суматорів для створення 8-бітної машини, або 144 реле.



Згадайте наш вихідний пульт керування з перемикачами та лампочками.

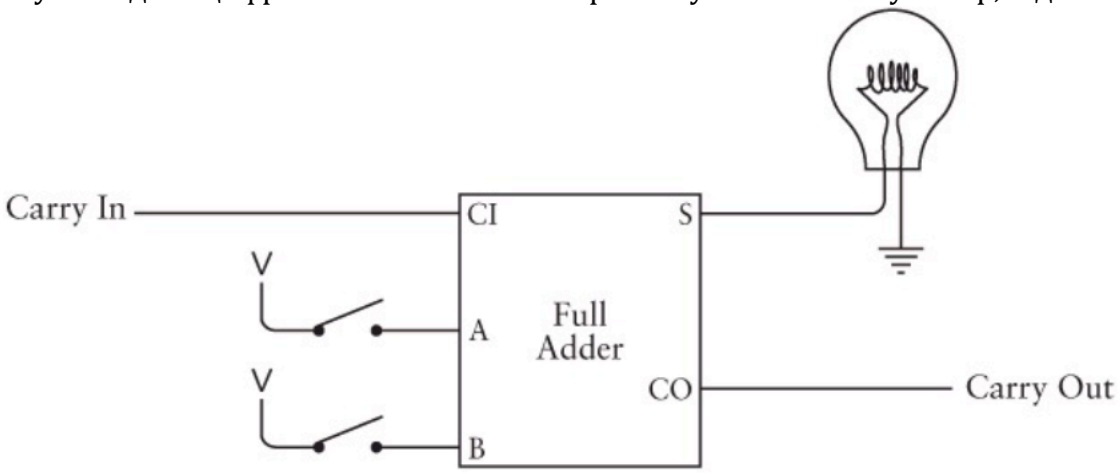
Тепер ми можемо почати приєднувати перемикачі та лампочки до повного суматора.

Спочатку підключимо два крайні праві перемикачі і крайню праву лампочку до повного суматора.



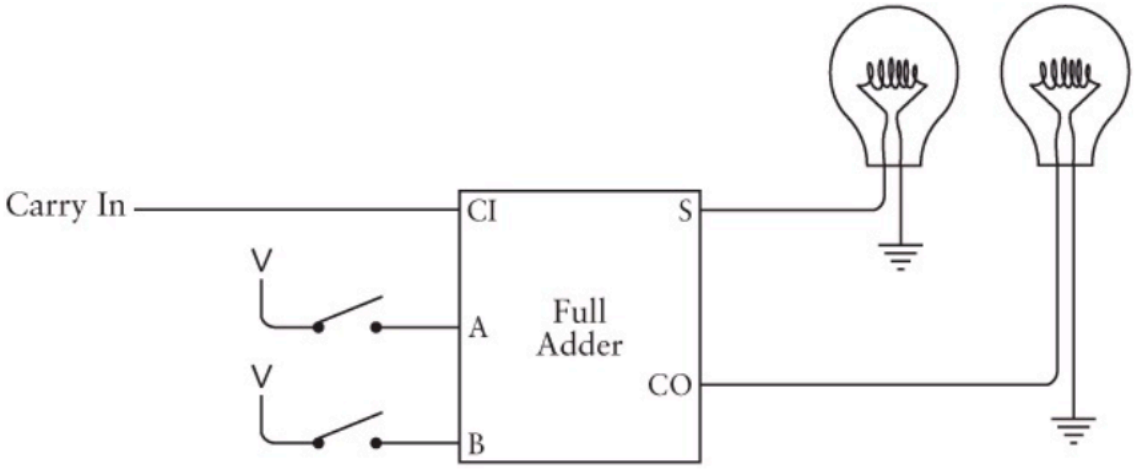
Коли ви починаєте додавати два двійкові числа, перший стовпець цифр відрізняється від інших тим, що не може містити біт переносу з попереднього стовпця. У першому стовпці немає біта перенесення, тому вхід для перенесення повного суматора з'єднується із землею, тобто його значенням є 0 біт. Зрозуміло, *в результаті* додавання першої пари двійкових цифр може вийти біт перенесення. Цей вихід перенесення – вхід для наступного стовпця.

Для наступних двох цифр та лампочки ви використовуєте повний суматор, підключений так.



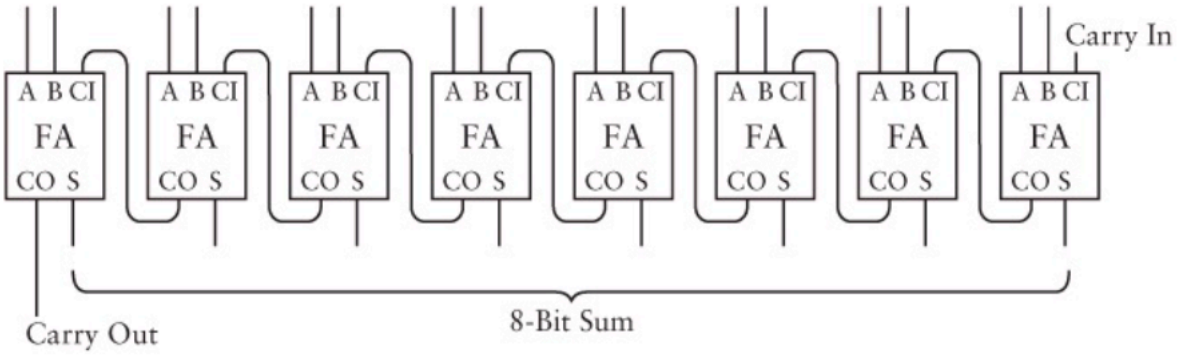
Вихід перенесення, отриманий від першого повного суматора є входом для другого повного суматора. Кожен наступний стовпець цифр складається за тією самою схемою. Кожен розряд перенесення з одного стовпця подається на вхід для перенесення наступного стовпця.

Нарешті, восьма та остання пара перемикачів підключена до останнього повного суматора.

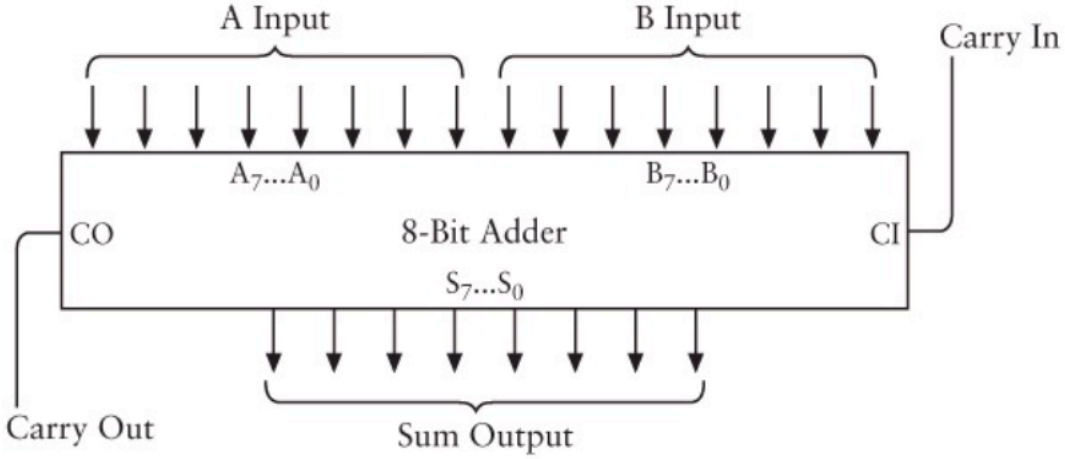


Тут останній вихід для перенесення підключений до дев'ятої лампочки.

Ось ще один спосіб зобразити схему восьми повних суматорів (full adder, FA), в якій кожен вихід для переносу (CO) підключений до наступного входу для переносу (CI).



Представимо єдине позначення 8-бітного суматора, входи позначимо літерами від A_0 до A_7 і від B_0 до B_7 виходи – літерами від S_0 до S_7 (від sum – «сума»).



Це найпоширеніший спосіб позначення окремих бітів багатобітного числа. Біти A_0, B_0 і S_0 є молодшими, а біти A_7, B_7 і S_7 – *старшими.* Наприклад, ось як за допомогою цих букв з індексами

$$A_7\ A_6\ A_5\ A_4\ A_3\ A_2\ A_1\ A_0$$
$$0\ 1\ 1\ 0\ 1\ 0\ 0\ 1$$

можна було б уявити двійкове число 0110 1001.

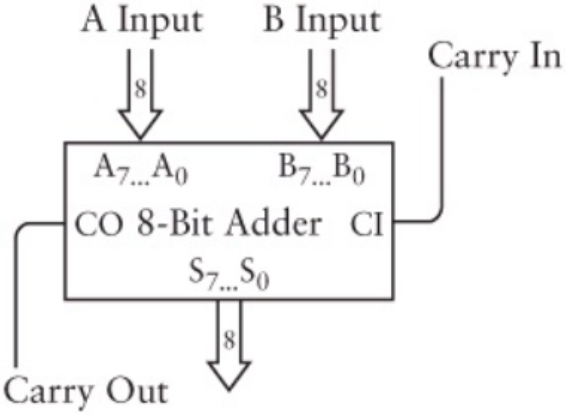
Індекси починаються з 0 і збільшуються в міру переходу до більш значущих цифр, оскільки

$$2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$$
$$0\ 1\ 1\ 0\ 1\ 0\ 0\ 1$$

вони відповідають показнику ступеня двійки.

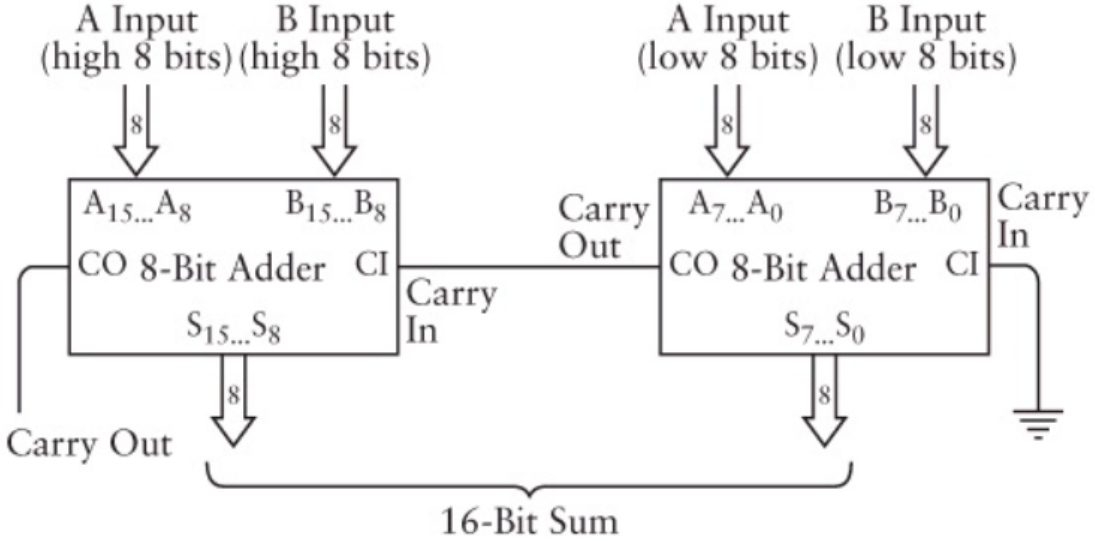
Якщо ви помножите кожний ступінь двійки на цифру, розташовану під нею, та додасте результати, отримайте десятковий еквівалент числа 0110 1001, який дорівнює 64 + 32 + 8 + 1, або 105.

Інакше 8-бітний суматор можна зобразити так.



Вісімки всередині стрілок вказують на те, що кожна з них – це група із восьми окремих сигналів. Індекси символів $A_7 \dots A_0$, $B_7 \dots B_0$ та $S_7 \dots S_0$ також позначають восьмирозрядність числа.

Як тільки зберете один 8-бітний суматор, ви зможете створити другий. Їх легко *розташувати каскадом,* щоб додати два 16-бітові числа.



Вихід для перенесення правого суматора пов'язаний із входом для перенесення лівого. Лівий суматор як вхідні значення приймає найстарші вісім цифр двох доданків і як вихідне значення видає найстарші вісім цифр.

Тепер ви можете запитати: «Невже комп'ютери *справді* додають цифри саме так?»

В принципі так. Але не зовсім.

По-перше, суматори можуть бути швидше за ті, які ми описали. Якщо ви подивитеся на те, як працює ця схема, то зрозумієте, що вихід перенесення від молодшої пари цифр необхідний для додавання з наступною парою, вихід перенесення від другої пари цифр – для додавання з третьою парою і т. д. Загальна швидкість суматора дорівнює кількості бітів, помноженого на швидкість одного повного суматора. Це називається *наскрізним перенесенням.* Швидкі суматори використовують додаткові схеми *прискореного перенесення.*

По-друге (і це найголовніше), комп'ютерам більше не потрібне реле! Однак перші цифрові комп'ютери, створені на початку 1930-х років, використовували реле, пізніше вакуумні лампи. Сучасні комп'ютери створюються з урахуванням транзисторів. Транзистори в основному функціонують так само, як і реле, проте (як ми побачимо далі) вони набагато швидше, компактніше, тихіше, дешевше і споживають набагато менше енергії. Для побудови 8-бітного суматора, як і раніше, потрібні 144 транзистори (або більше, якщо ви хочете замінити наскрізне перенесення схемою прискореного переносу), проте при цьому розмір схеми мікроскопічний.

Тема 9. Байти та шістнадцяткові числа

Дві вдосконалені лічильні машини, описані в попередній темі, добре ілюструють концепцію *потоків даних.* Восьмибітні значення переміщаються ланцюгом від одного компонента до іншого. Ці значення подаються на входи суматорів, клямок та селекторів, а також з'являються на виходах цих пристроїв. Крім того, 8-бітні значення задаються за допомогою перемикачів та відображаються поруч лампочок. Таким чином, потік даних у цих схемах має *ширину вісім біт.* Але чому? Чому не шість, не сім, не дев'ять і десять?

Можна було б відповісти, що основою наших удосконалених лічильних машин є вихідний суматор з теми 12, який працює з 8-бітними значеннями. Однак немає жодних особливих причин конструювати цю машину саме так. Просто при її створенні ми вважали цю величину зручною. Як би там не було, визнаю, що схитрував, оскільки від початку знав (можливо, і ви теж), що вісім біт даних відповідають одному байту.

Слово «байт» (byte) виникло в компанії IBM приблизно в 1956 році. Воно походить від слова bite («шматок»), але його було вирішено писати через букву *у,* щоб не плутати зі словом bit («біт»). Протягом деякого часу слово "байт" означало просто число бітів у конкретному потоці даних. Однак у середині 1960-х, у зв'язку з розробкою сімейства комп'ютерів System/360 у компанії IBM, це слово позначало групу з восьми біт.

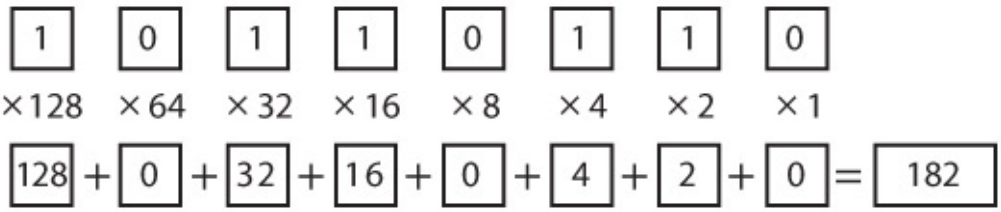
Як 8-розрядне число, байт може набувати значення в діапазоні від 00000000 до 11111111. Ці значення можуть описувати позитивні цілі числа від 0 до 255, а при використанні доповнення до двійки для подання негативних чисел вони можуть відображати як позитивні, так і негативні числа в діапазоні від −128 до 127. Крім того, конкретний байт може просто представляти одну з 2^8, або 256, різних речей.

Число 8 виявилося дуже зручною величиною. Компанія IBM віддала перевагу 8-бітним байтам у зв'язку з простотою зберігання чисел у форматі BCD (про який я розповім у темі 23). Однак, як ми побачимо далі, байт ідеально підходить для зберігання тексту, оскільки *більшу* частину мов світу (за винятком ідеограм, що використовуються в китайській, японській та корейській) можна уявити менш ніж 256 символами. Крім того, байт ідеально підходить для подання відтінків сірого на чорно-білих фотографіях, оскільки людське око розрізняє приблизно 256 відтінків цього кольору. А там, де не вистачає одного байту (наприклад, для представлення вищезгаданих ідеограм китайської, японської та корейської мов), як правило, можна використовувати два байти, які дозволяють висловити 2^{16}, або 65536, різних елементів.

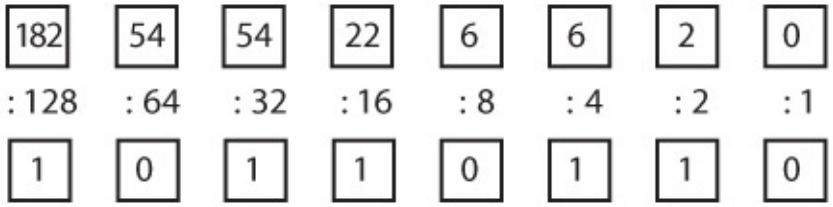
Половина байта, тобто чотири біти, іноді називається *тетрадою (nibble),* проте це слово вживається набагато рідше, ніж "байт".

Оскільки байти часто використовуються при описі роботи комп'ютерів, нам потрібно якомога коротший спосіб запису їх значення. Наприклад, запис числа, що складається з восьми двійкових цифр 10110110, безумовно, є коректним, але навряд чи лаконічним.

Зрозуміло, ми завжди можемо звертатися до байтів, використовуючи їх десяткові еквіваленти, але це вимагатиме перетворення двійкового числа в десяткове, що хоч і нескладно, але дуже обтяжливо. У темі 8 я продемонстрував один досить простий підхід. Оскільки кожна двійкова цифра відповідає ступеню 2, ми можемо просто записати цифри двійкового числа, а під ними – степені 2, після чого перемножити числа в кожному стовпці і додати результати. Далі подано процес перетворення числа 10110110.



Процес перетворення десяткового числа на двійкове менш зручний і передбачає розподіл десяткового числа на спадні ступені двійки. Частка від кожного поділу – двійкова цифра, а залишок поділяється на наступний у порядку зменшення ступеня двійки. Ось як десяткове число 182 перетворюється на двійкове.

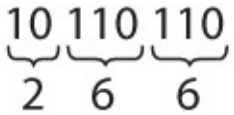


У темі 8 ця техніка докладно описана. Проте для перетворення двійкових чисел на десяткові і назад зазвичай потрібний папір, олівець та практика.

Ми вже дізналися про вісімкову систему числення – систему числення з основою 8, де використовуються лише цифри 0, 1, 2, 3, 4, 5, 6 і 7. Перетворити вісімкове число на двійкове легко. Все, що потрібно, – це запам'ятати 3-бітовий еквівалент кожної вісімкової цифри.

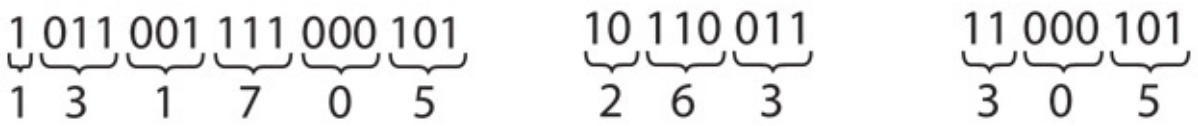
Binary	Octal
000	**0**
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Якщо у вас є двійкове число (наприклад, 10110110), починайте перетворення з правого краю. Кожна група з трьох біт відповідає вісімковій цифрі.



Таким чином, байт 10110110 можна виразити у вигляді вісімкового числа 266. Цей вислів, безумовно, є лаконічнішим, отже, вісімкова система дійсно підходить для представлення байтів. Проте вона має невеликий недолік.

У двійковій системі байти виражаються значеннями в діапазоні від 00000000 до 11111111, у вісімковій – значеннями в діапазоні від 000 до 377. Як було показано в попередньому прикладі, середній і крайній праворуч вісімковим цифрам відповідають групи з трьох біт, однак крайній зліва вісімковій цифрі відповідають тільки два біти. Це означає, що вісімкове уявлення 16-розрядного числа не збігається з вісімковими уявленнями двох байтів, що становлять це 16-розрядне число.



Щоб узгодити уявлення багатобайтних значень із уявленнями окремих байтів, потрібна система, у якій кожен байт ділиться на однакову кількість бітів. Отже, нам потрібно розділити кожен байт на чотири значення по два біти кожне (система обчислення з основою 4) або на два значення по чотири біти кожне (система обчислення з основою 16).

Систему числення з основою 16 ми ще не розглядали, і на те є причини. Система числення з основою 16 називається *шістнадцятковою*,* – навіть назва важковимовна.

Шістнадцяткова система числення використовується для позначення MAC-адрес (унікальних фізичних адрес мережевого обладнання) та для запису мережевих (IP) адрес у сучасному протоколі IPv6. З цим ви можете зіткнутися під час налаштування доступу Інтернет на своєму комп'ютері.

У десятковій системі числення рахуємо так:

0 1 2 3 4 5 6 7 8 9 10 11 12 …

У вісімковій системі, як ви пам'ятаєте, не використовуються цифри 8 і 9:

0 1 2 3 4 5 6 7 10 11 12 …

У системі з основою 4 не потрібні цифри 4, 5, 6 та 7:

0 1 2 3 10 11 12 …

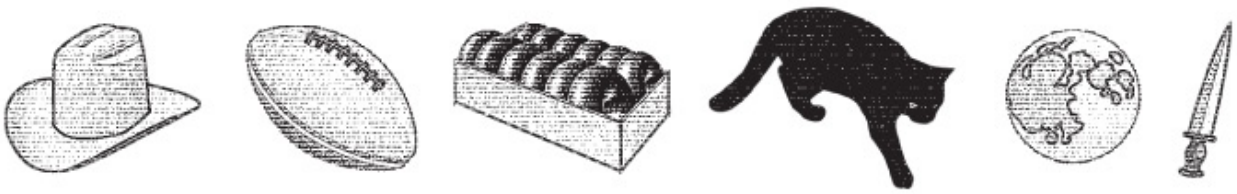
Нарешті, у двійковій системі достатньо лише 0 та 1:

0 1 10 11 100 …

Однак шістнадцяткова система відрізняється тим, що в ній використовується *більше* цифр, ніж у десятковій. У шістнадцятковій системі підрахунок відбувається приблизно так:

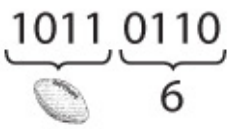
0 1 2 3 4 5 6 7 8 9?????? 10 11 12 …

У даному випадку 10 відповідає числу $16_{\text{десять.}}$ Знаки запитання говорять про те, що нам потрібні ще шість символів для представлення шістнадцяткових чисел. Що це за символи? Звідки їх брати? Що ж, оскільки вони не дісталися нам у спадок, подібно до інших традиційних числових символів, ми можемо придумати їх самостійно, наприклад такі.



На відміну від символів, що використовуються для позначення більшості чисел, ці позначення мають перевагу: вони легко запам'ятовуються і ототожнюються з тими величинами, які представляють. Існує так званий десятигалонний ковбойський капелюх, м'яч для американського футболу (11 гравців у команді), дюжина пончиків (12 штук), чорна кішка (з якою асоціюється нещасливе число 13), повний місяць (з'являється на небі через 14 днів після молодого) та кинжал (що нагадує про вбивство Юлія Цезаря 15-го дня березня). Кожен байт можна виразити у вигляді двох шістнадцяткових цифр. Іншими словами, шістнадцяткова цифра еквівалентна чотирьом бітам, або одній тетраді. У наступній таблиці показані відповідності двійкових, шістнадцяткових та десяткових чисел.

Binary	Hexadecimal	Decimal	Binary	Hexadecimal	Decimal
0000	0	0	1000	8	8
0001	1	1	1001	9	9
0010	2	2	1010		10
0011	3	3	1011		11
0100	4	4	1100		12
0101	5	5	1101		13
0110	6	6	1110		14
0111	7	7	1111		15



Ось як можна уявити двійкове число 10110110 у шістнадцятковій системі.

І не важливо, чи маємо ми справу з багатобайтними числами.

Один байт завжди представляється парою шістнадцяткових цифр.

На жаль (а може, на щастя), ми не збираємося використовувати футбольні м'ячі та пончики для запису шістнадцяткових чисел, хоча вони, безумовно, могли б пригодитися для цієї мети. Замість них у шістнадцятковій системі застосовуються позначення, що приводять багатьох у збентеження. Справа в тому, що шість відсутніх шістнадцяткових цифр репрезентують шість перших букв латинського алфавіту:

0 1 2 3 4 5 6 7 8 9 ABCDEF 10 11 12 ...

У наступній таблиці показано *реальну* відповідність між двійковими, шістнадцятковими та десятковими числами.

Binary	Hexadecimal	Decimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Таким чином, двійкове число 10110110 можна уявити шістнадцятковим числом B6, не малюючи футбольний м'яч. Як ви пам'ятаєте, у попередніх темах я вказував основу системи числення за допомогою нижнього індексу, наприклад: $10110110_{\text{ДВА}}$ для двійкової системи; $2312_{\text{Чотири}}$ – для четвіркової; $266_{\text{вісім}}$ – для вісімкової; $182_{\text{десять}}$ – для десяткової.

За аналогією ми можемо використовувати позначення $B6_{\text{шістнадцять}}$ для шістнадцяткової системи.

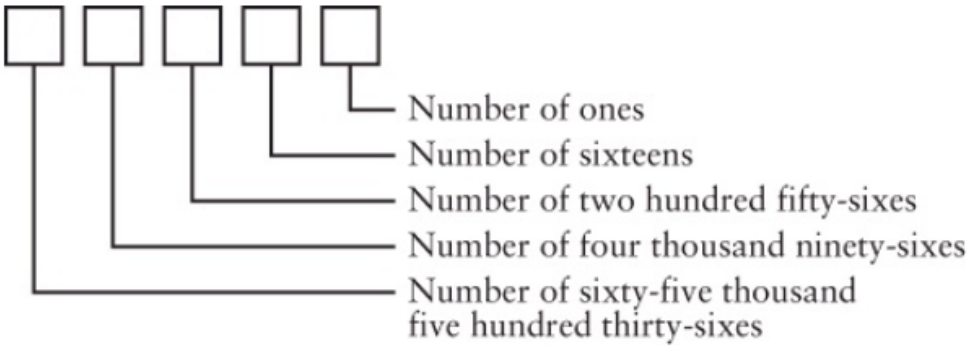
Однак такий вираз надто громіздкий. На щастя, для шістнадцяткових чисел існують і інші, більш короткі позначення. Ви можете записати таке число таким чином:

$B6_{\text{НЕХ.}}$

У цій книзі я використовуватиму поширений спосіб представлення шістнадцяткових чисел, що передбачає додавання до числа малої латинської літери *h:*

B6h.

У шістнадцятковому числі положення кожної цифри відповідає ступеню 16.



Шістнадцяткове число 9A48Ch можна представити так:

9A48Ch = 9 × 10000h +
A × 1000h +
4 × 100h +
8 × 10h +
C × 1h.

Цей вираз можна записати, використовуючи ступінь числа 16:

$9A48Ch = 9 \times 16^4 +$
$A \times 16^3 +$
$4 \times 16^2 +$
$8 \times 16^1 +$
$C \times 16^0.$

Або десяткові еквіваленти цих ступенів:

9A48Ch = 9 × 65 536 +
A × 4096 +
4 × 256 +
8 × 16 +
C × 1.

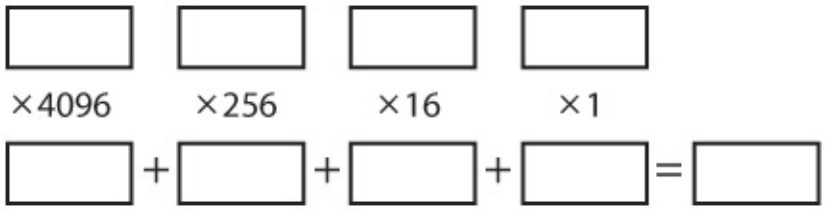
Зверніть увагу на відсутність двозначності при записі окремих цифр числа (9, A, 4, 8 та C) без нижнього індексу, що означає основу системи числення. Дев'ять – це 9, чи то десяткова чи шістнадцяткова система числення. З іншого боку, A очевидно представляє шістнадцятковий еквівалент десяткового числа 10.

По суті перетворення всіх цифр в десяткові числа дозволяє виконати розрахунок підсумкового значення:

9A48Ch = 9 × 65 536 +
10 × 4096 +
4 × 256 +
8 × 16 +
12 × 1.

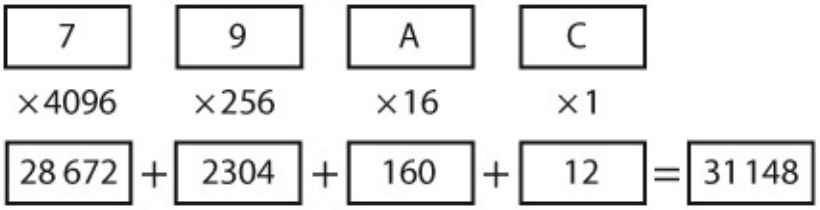
У результаті виходить число 631948. Таким чином шістнадцяткові числа перетворюються на десяткові.

Шаблон для перетворення будь-якого чотиризначного шістнадцяткового числа в десяткове

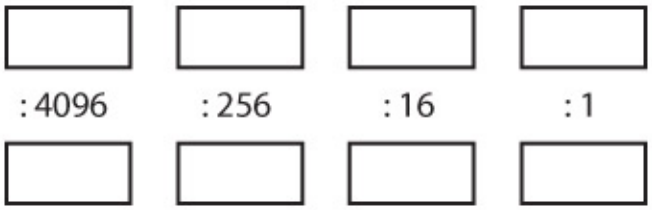


виглядає так.

Як приклад перетворимо число 79ACh. Майте на увазі, що шістнадцяткові цифри A і C еквівалентні десятковим числам 10 і 12.

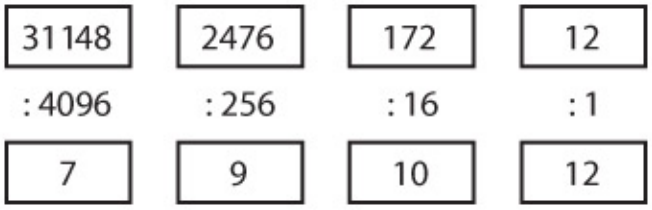


Перетворення десяткових чисел на шістнадцяткові зазвичай передбачає виконання операцій ділення. Число яке менше або дорівнює 255 можна представити одним байтом, що складається з двох шістнадцяткових цифр. Щоб обчислити ці дві цифри, потрібно поділити число на 16, у результаті вийде частка і залишок. Повернемося до прикладу із десятковим числом 182. 182 поділити на 16, отримаємо 11 (що відповідає цифрі B у шістнадцятковій системі) і 6 у залишку. Так, шістнадцятковим еквівалентом десяткового числа 182 є B6h. Якщо десяткове число, яке ви хочете перетворити, менше 65536, то шістнадцятковий еквівалент складатиметься не більше ніж



з чотирьох цифр. Шаблон для перетворення такого числа на шістнадцяткове наступний.

Спочатку помістіть десяткове число у верхній лівий прямокутник. Це наше перше ділене. Поділимо число на 4096 (перший дільник). Частку впишемо в прямокутник, розташований під ділимим, а залишок – в прямокутник праворуч від ділимого. Цей залишок – нове ділене, яке ми

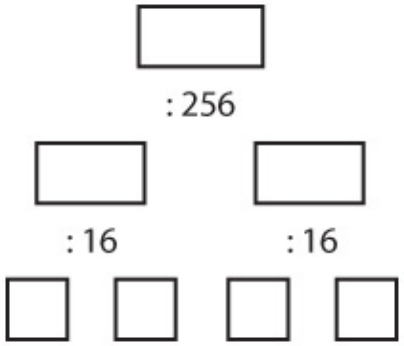


поділимо на 256. Ось як число 31148 перетворюється на шістнадцятковий формат.

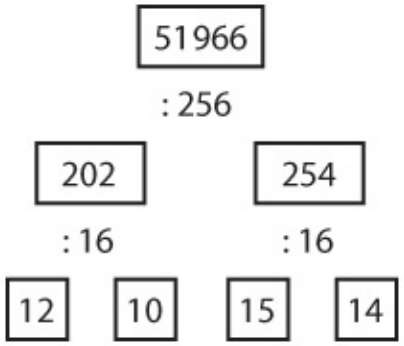
Десяткові числа 10 та 12 відповідають шістнадцятковим цифрам A і C, тому результат дорівнює 79ACh.

Одна з проблем цієї техніки полягає в тому, що для ділення ви, ймовірно, вирішите використовувати калькулятор, а калькулятори не показують залишок від ділення. Якщо ви поділите 31148 на 4096 на калькуляторі, то отримаєте 7,6044921875. Щоб розрахувати залишок, потрібно помножити 4096 на 7 (вийде 28672) і відняти це значення з 31148. Або помножити 4096 на 0,6044921875 – дробову частину результату від поділу. (Щоправда, деякі калькулятори передбачають функцію перетворення десяткових чисел у шістнадцяткові та назад.)

Інший спосіб перетворення десяткових чисел від 0 до 65535 в шістнадцяткові передбачає поділ числа на два байти шляхом його поділу на 256. Потім кожен байт ділиться на 16. Шаблон для цього наступний.



Почнемо згори. Після кожної операції ділення частка поміщається в прямокутник, розташований ліворуч від дільника, а залишок – у прямокутник праворуч. Наприклад, число



51966 перетворюється таким чином.

Шістнадцятковими еквівалентами чисел 12, 10, 15 і 14 є літери *C, A, F і E,* тому результат швидше нагадує слово, ніж число.

Далі представлена таблиця додавання для шістнадцяткової системи числення.

+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
7	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
8	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
9	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
A	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
B	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A
C	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

Використовуючи цю таблицю та звичайні правила додавання в стовпчик, можна додавати шістнадцяткові числа.

```
  4A3378E2
+ 877AB982
-----------
  D1AE3264
```

У темі 13 я згадував, що для представлення негативних чисел можна застосовувати доповнення до двійки. Якщо ви маєте справу з 8-бітовими двійковими числами зі знаком, то негативні числа починаються з 1. У шістнадцятковій системі числення двозначні числа зі знаком негативні, якщо вони починаються з цифр 8, 9, A, B, C, D, E або F, оскільки їх двійкові еквіваленти починаються з 1. Наприклад, число 99h може відповідати або десятковому числу 153 (якщо ви знаєте, що маєте справу з однобайтними числами без знака), або десятковому числу -103 (якщо ви знаєте, що це число зі знаком).

Крім того, байт 99h може відповідати і десятковій кількості 99. Це цікаво, але, схоже, суперечить усьому, про що ми говорили досі. У темі 23 поясню як це працює, а тепер зупинимося на пам'яті.

Тема 10. Складання пам'яті

Щоранку, коли ми прокидаємося, вмикається наша пам'ять. Ми згадуємо де знаходимося, що робили напередодні, що плануємо зробити сьогодні. Пам'ять повертається відразу або фрагментами, і протягом ще кількох хвилин людина може чогось не пам'ятати («Забавно, я не пам'ятаю, що ліг спати в шкарпетках»), проте загалом ми здатні відновити безперервність свого життя, щоб розпочати новий день.

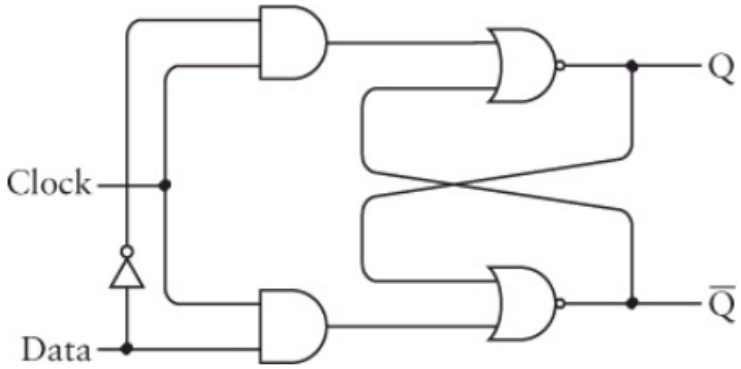
Зрозуміло, що людська пам'ять не цілком упорядкована. Спробуйте згадати щось зі шкільного курсу геометрії. Ймовірно, почнете думати про однокласника, що сидів перед вами, або про той день, коли спрацювала пожежна тривога якраз у той момент, коли вчитель збирався пояснити вам, що означає вираз «що й вимагалося довести».

Людська пам'ять не є цілком надійною. Писемність, можливо, було винайдено спеціально, щоб компенсувати недоліки людської пам'яті. Можливо, минулої ночі ви раптово прокинулися о 3:00 з чудовою ідеєю для сценарію. Схопили ручку та папір, які тримаєте біля ліжка спеціально для таких випадків, та записали ідею. Наступного ранку ви читаєте свою блискучу думку і починаєте роботу над сценарієм («Хлопець зустрічає дівчину, погоня на машинах та вибухи»… і це все?). Або не починаєте.

Ми *пишемо,* а потім *читаємо.* Ми *зберігаємо,* а потім *витягаємо.* Ми *зберігаємо* інформацію, а надалі *отримуємо* до неї *доступ.* Функція пам'яті полягає в тому, щоб зберігати інформацію без спотворень між двома подіями. Щоразу, коли ми зберігаємо інформацію, ми використовуємо різні типи пам'яті. Папір – добрий носій для зберігання текстової інформації, магнітна стрічка підходить для зберігання музики та фільмів.

Телеграфні реле, об'єднані у вентилі, а потім у тригери, також можуть зберігати інформацію. Як ми бачили раніше, тригер здатний зберігати один біт. Це не дуже багато, але це початок. Як тільки ми навчимося зберігати один біт, ми легко зможемо впоратися з двома, трьома та більше.

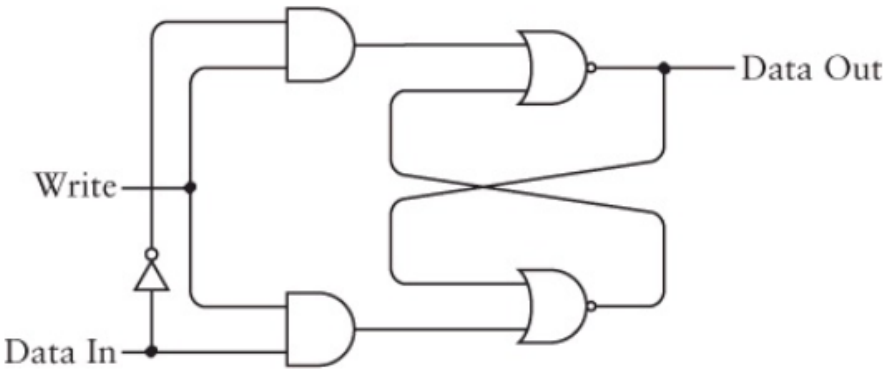
У Темі 14 ми познайомилися з D-тригером зі спрацьовуванням за рівнем, який складається з інвертора, двох вентилів І та двох вентилів АБО-НЕ.



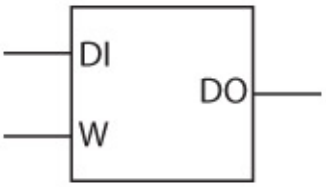
Коли вхід Clk дорівнює 1, вихідний Q сигнал збігається з вхідним сигналом «Дані». Коли значення входу Clk змінюється на 0, вихід Q зберігає останнє значення входу «Дані». Подальші зміни вхідного сигналу "Дані" не впливають на виходи до тих пір, поки значення входу Clk знову не зміниться на 1. Ось таблиця логіки для тригера.

Inputs		Outputs	
D	**Clk**	**Q**	**Q-bar**
0	1	0	1
1	1	1	0
X	0	Q	Q-bar

У Темі 14 цей тригер використовувався в різних схемах, а зараз він застосовуватиметься виключно для зберігання одного біта. З цієї причини я маю намір перейменувати входи та виходи, щоб вони відповідали меті.

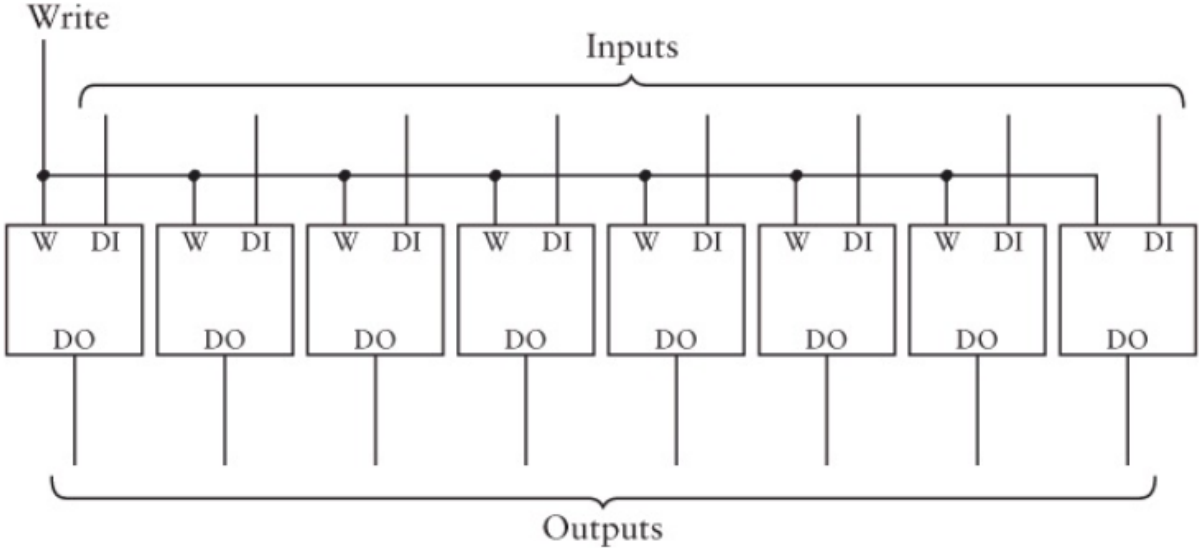


Це той самий тригер, тільки тепер вихід Q називається "Виведення даних" (Data Out, DO), а вхід Clk ("Запам'ятати цей біт") став "Записом" (Write, W). Так само, як ми можемо записати деяку інформацію на папері, сигнал "Запис" призводить до *запису* або *збереження* сигналу "Введення даних" (Data In, DI) у схемі. Зазвичай вхід "Запис" дорівнює 0, а сигнал "Введення даних" не впливає на вихід. Однак кожного разу, коли ми хочемо зберегти значення сигналу «Введення даних» у тригері, подаємо на вхід «Запис» 1, а потім знову 0. Як я згадував у Темі 14 схема такого типу також називається засувкою, оскільки вона як би «*закриває*» дані. Ось як ми можемо уявити

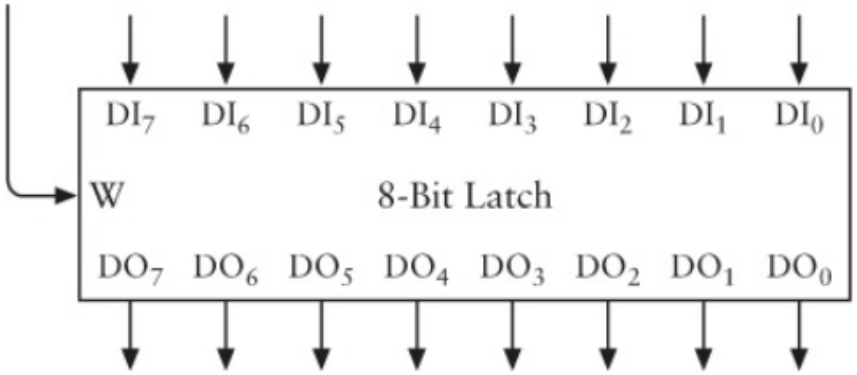


однобітну клямку без зображення всіх окремих компонентів.

Ми можемо досить легко об'єднати кілька однобітних клямок у багатобітну. Все, що для цього потрібно зробити, – з'єднати входи «Запис».



Ця 8-бітна клямка містить вісім входів і вісім виходів. Крім того, клямка має один вхід під назвою «Запис», який зазвичай дорівнює 0. Щоб зберегти 8-бітове значення в цій клямці, подайте на вхід «Запис» 1, а потім 0. Цю клямку також можна зобразити наступним чином.



Або так, щоб вона більше нагадувала зображення однобітної.

Інший спосіб з'єднання вісім однобітних клямок більш складний. Припустимо, нам потрібен лише один сигнал "Введення даних" та один сигнал "Виведення даних". Однак ми хочемо зберегти значення сигналу "Введення даних" вісім разів протягом дня або вісім разів протягом наступної хвилини. Крім того, ми хочемо мати можливість надалі переглянути ці вісім значень лише кинувши погляд на один сигнал «Виведення даних».

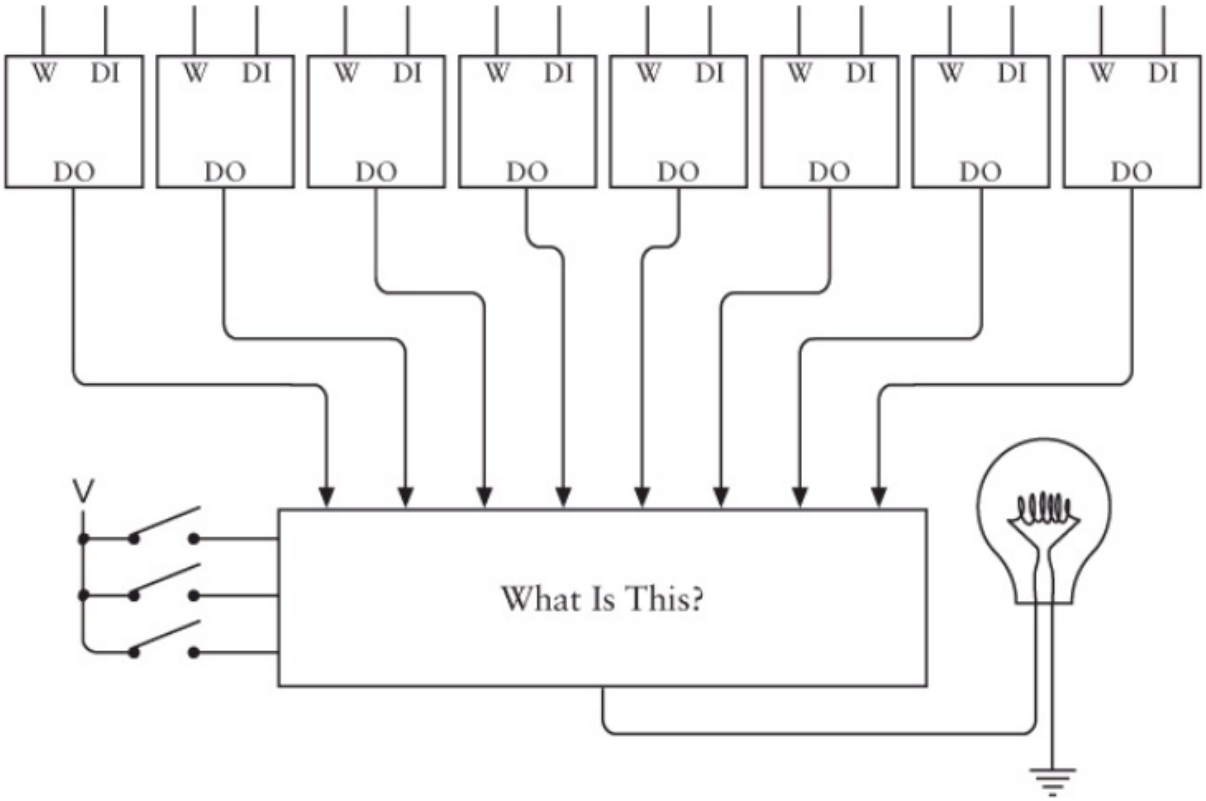
Іншими словами, замість збереження одного 8-бітного значення, як у випадку з 8-бітною клямкою, ми хочемо зберегти вісім окремих однобітних значень.

Чому ми хочемо зробити саме так? Мабуть, тому, що ми маємо лише одну лампочку.

Ми знаємо, що нам потрібно вісім однобітних клямок. Давайте поки не хвилюватимемося, як саме дані в них зберігаються. Зосередимося на перевірці сигналів «Виведення даних» цих восьми клямок, використовуючи лише одну лампочку. Звичайно, ми могли б перевіряти вихід кожної клямки, *вручну* переносячи лампочку від однієї клямки до іншої, але ми віддали б перевагу більш автоматизованому способу. Фактично ми хочемо вибирати одну з восьми однобітних клямок, використовуючи перемикачі.

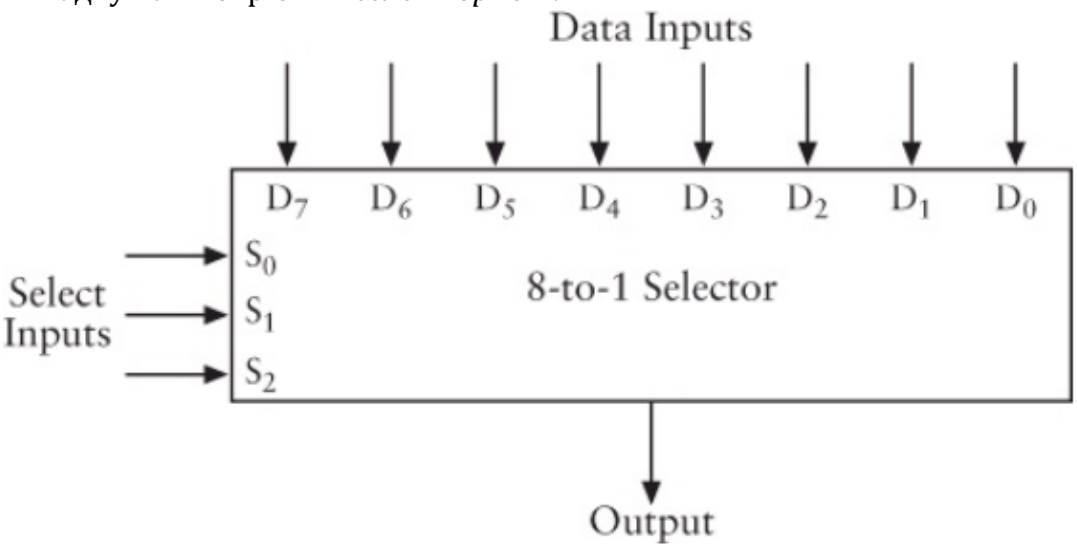
Скільки перемикачів потрібно? Якщо хочемо вибрати один з восьми елементів, потрібно три перемикачі. Три перемикачі можуть представляти вісім різних значень: 000, 001, 010, 011, 100, 101, 110 та 111.

Отже, ось наші вісім однобітних клямок, три перемикачі, лампочка та пристрій, який необхідно помістити між перемикачами та лампочкою.



Пристрій – це якийсь корпус з вісьмома входами зверху та трьома входами зліва. Замикаючи і розмикаючи три перемикачі, ми можемо вибрати, який із восьми входів повинен бути перенаправлений на вихід у нижній частині корпусу. Цей вихід запалює лампочку.

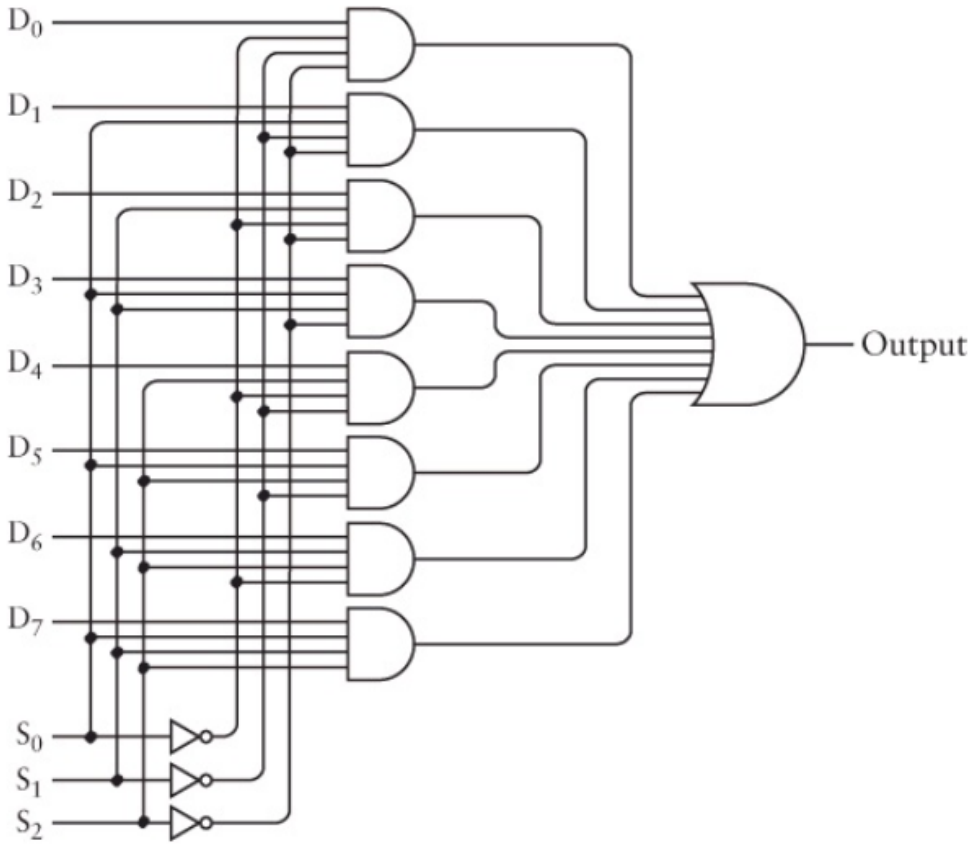
То що це за пристрій? Ми вже бачили щось подібне, хоч і не з такою кількістю входів. Воно схоже на схему, яку ми використовували в Темі 14 першої модифікованої версії суматора. У тому випадку нам потрібно було щось, що дозволяє вибрати, звідки повинен надходити сигнал на вхід суматора: від ряду перемикачів або з виходу клямки. Тоді ми назвали пристрій селектором "2 на 1". В даному випадку нам потрібний *селектор "8 на 1".*



Селектор восьми ліній на одну має вісім входів для даних (вгорі) та три входи для вибірки (Select, Sel) (ліворуч). Входи Select дозволяють вибрати сигнал якого входу для даних з'явиться на виході. Наприклад, якщо сигнали Select дорівнюють 000, то вихідний сигнал збігається з D_0. Якщо входи Select дорівнюють 111, то вихідний сигнал збігається з D_7. Якщо входи Select – 101, вихідний сигнал збігається з D_5. Наведемо таблицю логіки цього селектора.

Inputs			Outputs
S_2	S_1	S_0	**Q**
0	0	0	$\mathbf{D_0}$
0	0	1	$\mathbf{D_1}$
0	1	0	$\mathbf{D_2}$
0	1	1	$\mathbf{D_3}$
1	0	0	$\mathbf{D_4}$
1	0	1	$\mathbf{D_5}$
1	1	0	$\mathbf{D_6}$
1	1	1	$\mathbf{D_7}$

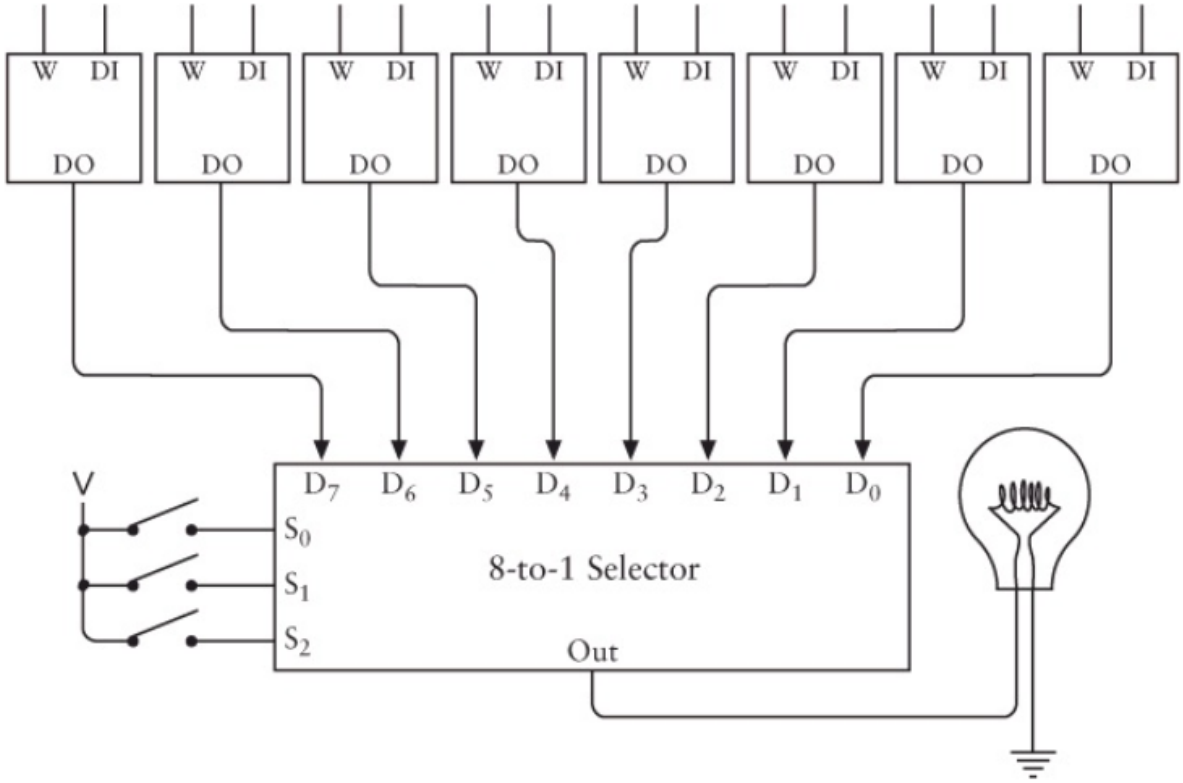
Селектор «8 на 1» складається з трьох інверторів, восьми чотиривходових вентилів І та одного восьмивходового вентиля АБО.



Ця схема може здатися досить заплутаною, проте за допомогою наступного прикладу намагатимуся переконати вас, що вона працює. Припустимо, сигнали Sel_2 і Sel_0 рівні 1, а сигнал S_1 – 0. На входи шостого зверху вентиля І подаються сигнали Sel_0, Sel_1 (інвертований), Sel_2, кожен з яких дорівнює 1. Ні на один інший вентиль І ці три сигнали не подаються, тому вихід всіх інших вентилів І дорівнюватиме 1. Вихід шостого вентиля І – 0 при D_5, рівному 0, або 1 при D_5, рівному 1. Те ж саме стосується крайнього праворуч вентиля АБО. Таким чином, якщо сигнали для вибірки дорівнюють 101, вихідний сигнал збігається з сигналом D_5.

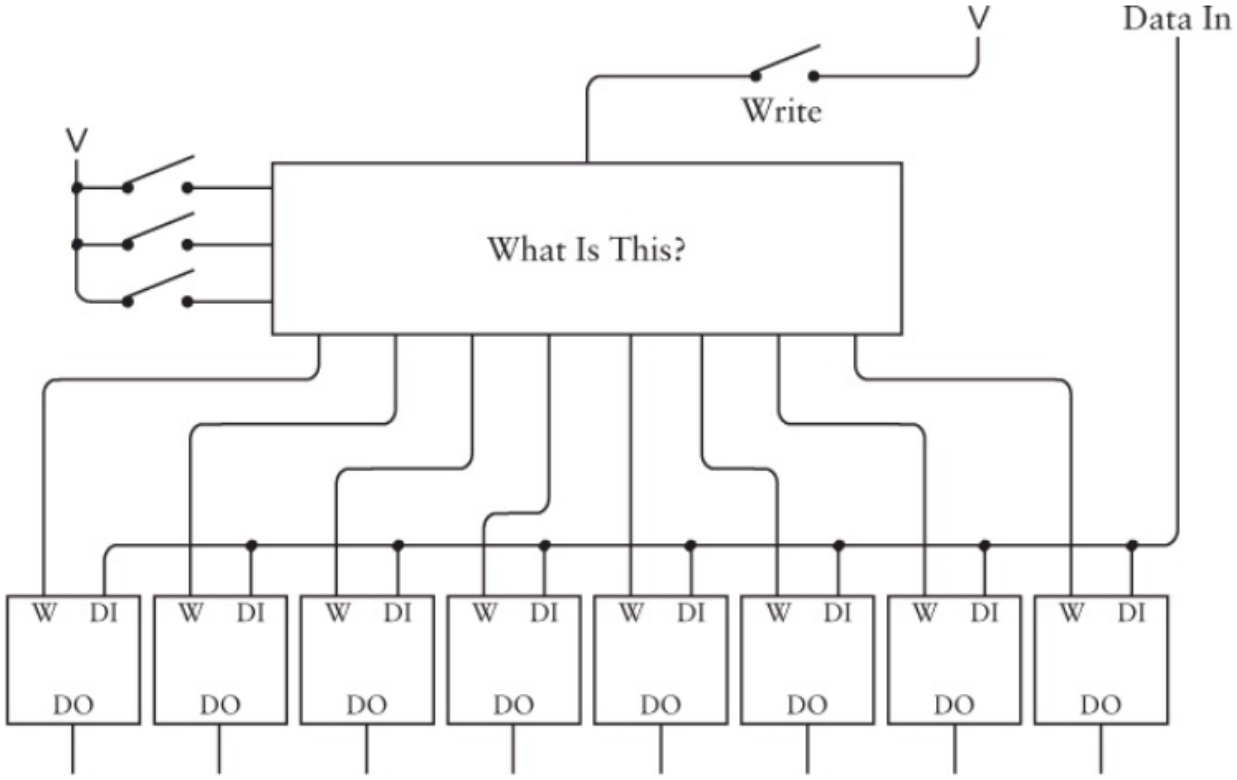
Давайте ще раз повторимо, чого хочемо досягти. Ми намагаємося з'єднати вісім однобітних засувок так, щоб у них можна було записувати та зчитувати дані окремо, використовуючи один

сигнал «Введення даних» та один сигнал «Виведення даних». Ми вже з'ясували, що можна вибрати сигнал "Виведення даних" однієї з восьми клямок, використовуючи селектор "8 на 1".



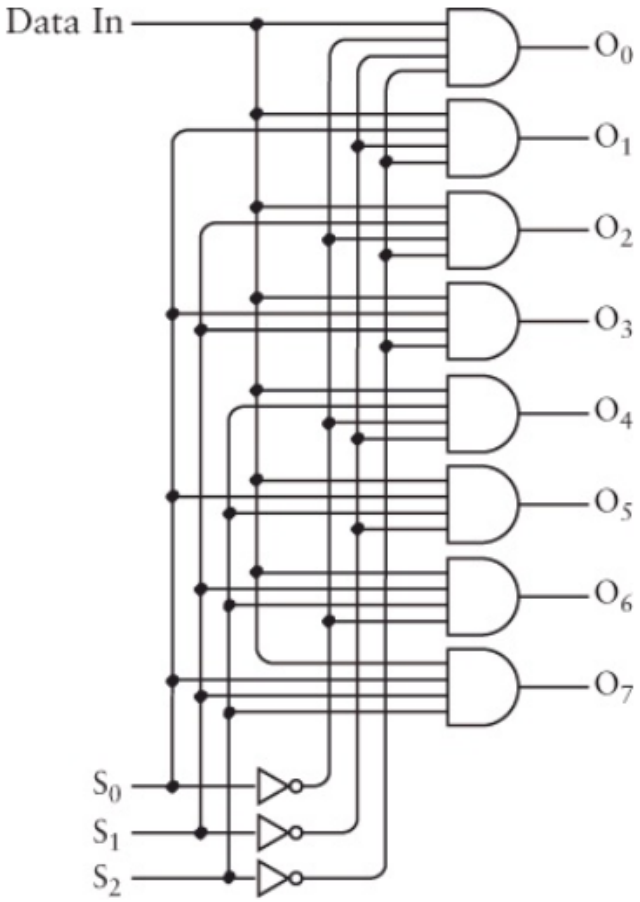
Отже, наше завдання наполовину вирішене. Тепер, коли ми визначилися з пристроєм на виході, займемося входом.

На вході ми маємо сигнали "Дані" та "Запис". Входи «Дані» клямки можна з'єднати між собою. Однак ми не можемо зробити те саме з сигналами «Запис», оскільки хочемо записувати дані окремо, отже, маємо подавати один сигнал «Запис» на одну (і лише одну!) клямку.



Тепер потрібна інша схема, яка трохи схожа на селектор "8 на 1", але виконує прямо протилежну дію. Ця схема називається *дешифратор "3 на 8".* У Темі 11 ми бачили простий дешифратор даних, коли з'єднували перемикачі для вибору кольору нашої ідеальної кішки.

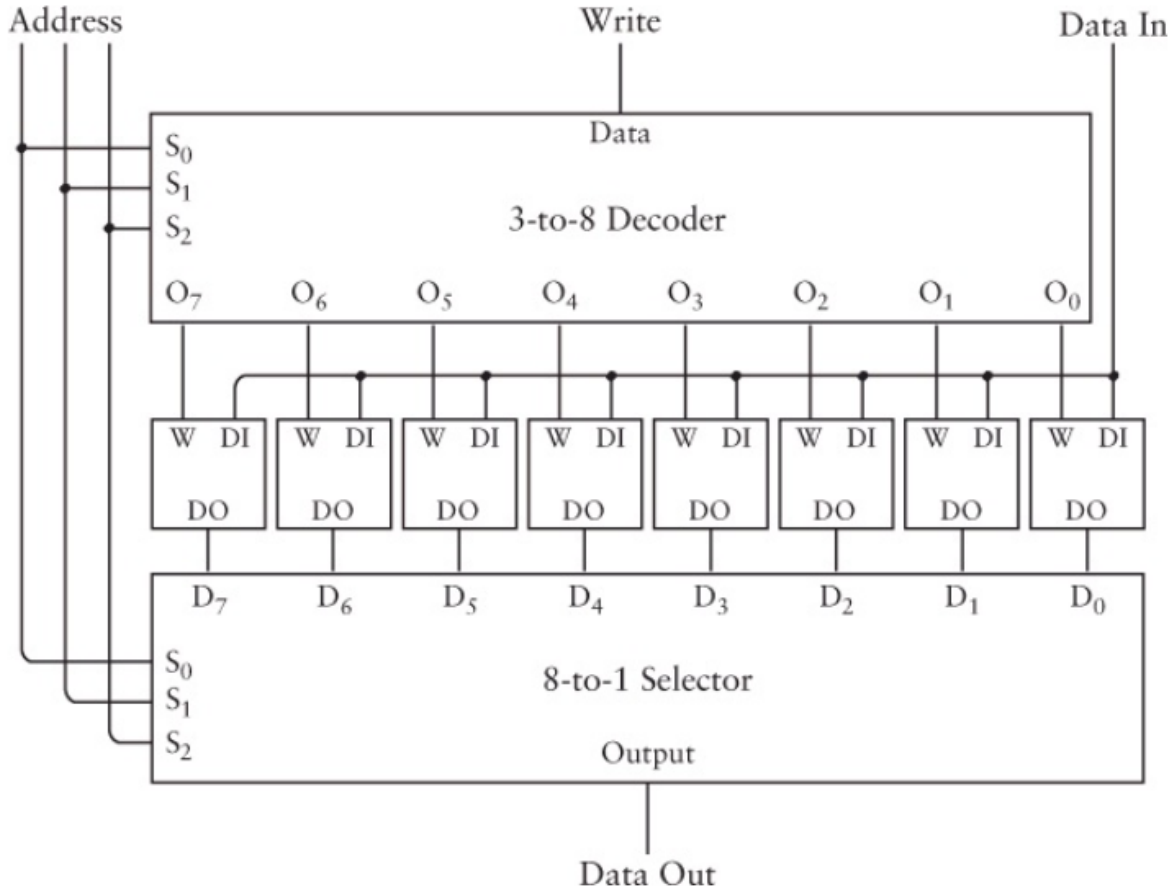
Дешифратор "3 на 8" має вісім виходів. У будь-який момент усі ці виходи дорівнюють 0, крім одного, який обраний вхідними сигналами Sel_0, Sel_1 і Sel_2. Значення цього виходу збігається зі значенням входу "Введення даних".



Зверніть увагу: вхідними сигналами шостого вентиля І зверху є Sel_0, Sel_1, Sel_2. Вони не подаються на жоден інший вентиль І, тому якщо на входи для вибірки подається значення 101, то виходи всіх інших вентилів І дорівнюватимуть 0. Вхід шостого вентиля І може мати значення 0, якщо вхід «Введення даних» дорівнює 0, або 1, якщо вхід «Введення даних» дорівнює 1. Повна таблиця логіки має такий вигляд.

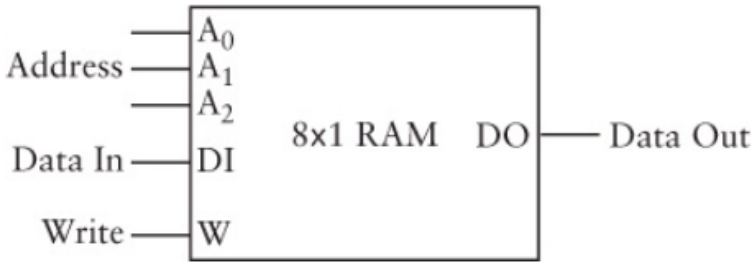
Inputs			**Outputs**							
S_2	S_1	S_0	O_7	O_6	O_5	O_4	O_4	O_2	O_1	O_0
0	0	0	0	0	0	0	0	0	0	Data
0	0	1	0	0	0	0	0	0	Data	0
0	1	0	0	0	0	0	0	Data	0	0
0	1	1	0	0	0	0	Data	0	0	0
1	0	0	0	0	0	Data	0	0	0	0
1	0	1	0	0	Data	0	0	0	0	0
1	1	0	0	Data	0	0	0	0	0	0
1	1	1	Data	0	0	0	0	0	0	0

Повна схема з вісьмома клямками виглядає таким чином.



Важливо: три сигнали вибірки для дешифратора та селектора є однаковими (вони позначені словом "адреса" – address, Addr). Подібно до поштового індексу, ця 3-бітна адреса визначає, до якої з восьми однобітних клямок ми звертаємося. На вході сигнал «Адреса» визначає, яка засувка збереже сигнал «Дані» під впливом сигналу «Запис». На виході (у нижній частині схеми) вхід «Адреса» управляє селектором «8 на 1» для того, щоб рахувати вихідний сигнал однієї з восьми клямок.

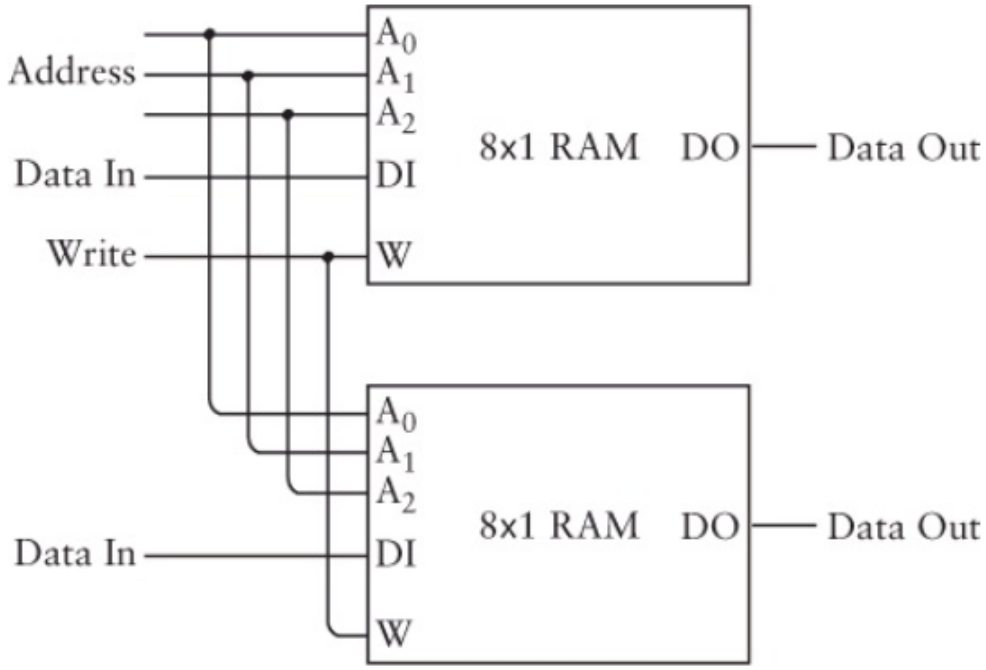
Ця конфігурація клямок іноді називається *пам'яттю із записом/читанням,* але частіше – *пам'яттю з довільним доступом,* або *довільною вибіркою* (random access memory, RAM). Ця конфігурація RAM зберігає вісім окремих однобітних значень. Її можна зобразити в такий спосіб.



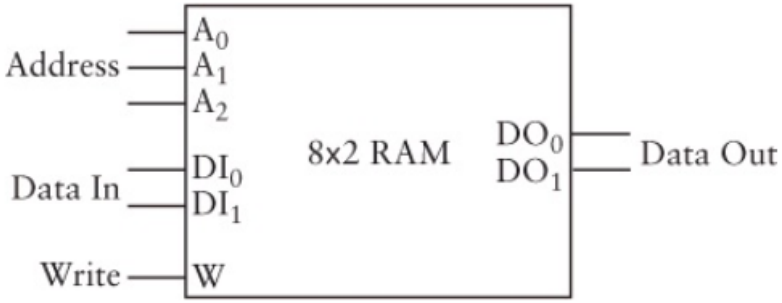
Пристрій називається *пам'яттю,* тому що він зберігає інформацію. Можливість *читання/запису* говорить про те, що ви можете зберегти нове значення в будь-якій клямці (*записати* значення), а також дізнатися, що зберігається в кожній із клямок (*прочитати* значення). Термін «довільний доступ» означає, що запис та зчитування інформації із засувок можуть здійснюватися шляхом зміни вхідних сигналів «Адреса». Крім пам'яті з довільним доступом, є пам'ять з послідовним доступом, при використанні якої для зчитування значення, що зберігається за адресою 101, потрібно спочатку прочитати значення, що зберігається за адресою 100.

Описана вище конфігурація RAM часто називається *масивом RAM.* Цей конкретний масив RAM організований за схемою, що іноді скорочено позначається «8 × 1» – вісім однобітних значень. Щоб визначити загальну кількість бітів, які можна зберегти в масиві RAM, потрібно перемножити ці два числа.

Масиви RAM можна комбінувати. Наприклад, об'єднати два масиви RAM «8×1».

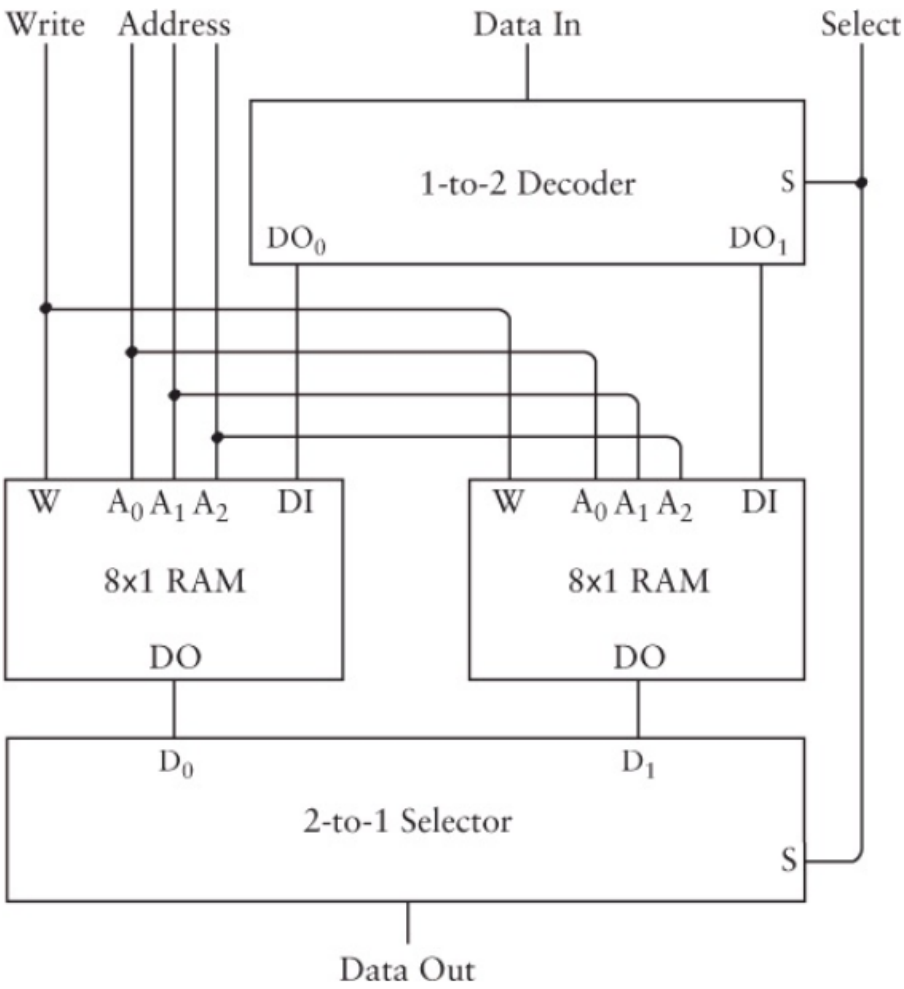


В даному випадку входи «Адреса» та «Запис» двох масивів RAM «8 × 1» з'єднані, тому в результаті виходить масив RAM «8 × 2».

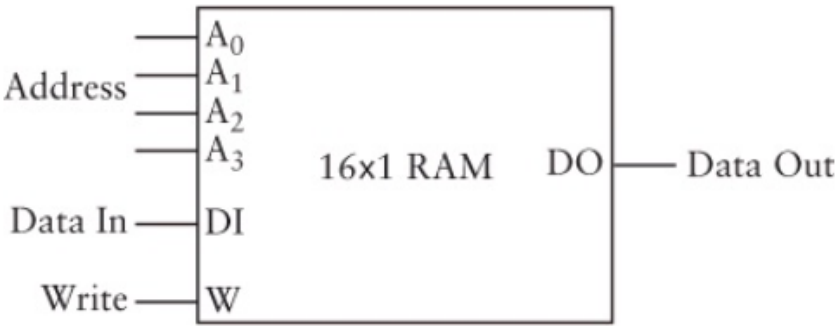


Цей масив RAM зберігає вісім значень, розмір кожного з яких становить два біти.

Крім того, два масиви RAM «8×1» можна об'єднати як окремі клямки, використовуючи селектор «2 на 1» та дешифратор «1 на 2».



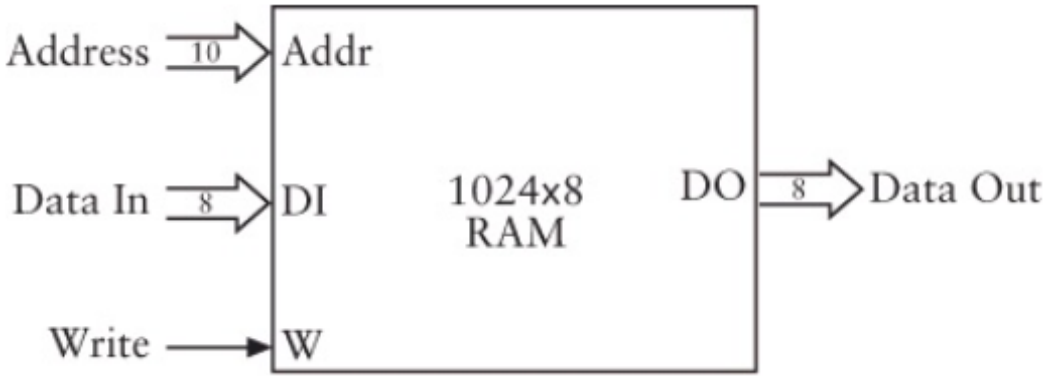
Сигнал "Вибірка", який подається як на дешифратор, так і на селектор, по суті, вибирає один із двох масивів RAM "8×1". Насправді він є четвертою адресною лінією. Таким чином, ми маємо справу з масивом RAM «16×1».



Цей масив RAM зберігає 16 значень, розмір кожного з яких становить один біт. Кількість значень, що зберігаються в масиві RAM, безпосередньо залежить від кількості входів "Адреса". При відсутності таких входів (як у випадку з однобітною та 8-бітною клямками) може бути збережено лише одне значення. За наявності одного входу «Адреса» можна зберегти два значення. Два входи «Адреса» дозволяють зберігати чотири значення, три входи «Адреса» – вісім, чотири входи – шістнадцять. Таке відношення можна виразити за допомогою рівняння:

Кількість значень у масиві RAM = $2^{\text{кількість входів «Адреса».}}$

Я показав, як можна сконструювати невеликі масиви RAM, тому вам неважко буде уявити набагато більші. Наприклад, такий.



Цей масив RAM зберігає в цілому 8196 біт інформації, які організовані у вигляді 1024 значень по вісім біт кожне. Цей масив має десять входів "Адреса", так як 2^{10} дорівнює 1024, вісім входів і вісім виходів для даних.

Інакше кажучи, цей масив RAM зберігає 1024 байт. Він схожий на поштове відділення із 1024 абонентськими скриньками. У кожному їх зберігається значення розміром один байт (яке, щоправда, може представляти просто спам).

Одна тисяча двадцять чотири байти – *кілобайт,* і тут виникає велика плутанина. Найчастіше приставка " *кіло-"* (від грецького "тисяча") використовується в метричній системі. Наприклад, кілограм – це 1000 грамів, кілометр – 1000 метрів. Проте кілобайт становить 1024 байт, а чи не 1000 байт.

Проблема в тому, що метрична система заснована на 10 ступенях, а двійкові числа – на 2 ступенях, і цим системам ніколи не зійтися. Ступенями 10 є 10, 100, 1000, 10 000, 100 000 і т. д., а ступенями 2–2, 4, 8, 16, 32, 64 і т. д. Не існує ступеня 10, який дорівнював би деякому ступеню 2.

Однак іноді ці дві системи зближуються. Так, значення 1000 досить близько до значення 1024. Висловлюючись математичною мовою, 2 у ступені 10 «приблизно дорівнює» 10 у ступені 3:

$2^{10} \approx 10^{3}$.

У цьому співвідношенні немає нічого чарівного. Воно лише передбачає, що конкретна ступінь 2 приблизно дорівнює конкретній ступені 10. Цей збіг дозволяє людям використовувати термін «кілобайт пам'яті», маючи на увазі 1024 байти.

Кілобайт скорочено позначається Кбайт (міжнародне позначення – Kb). Описаний вище масив RAM може зберігати 1024 байт або один кілобайт (1 Кбайт).

Ми не маємо на увазі, що в масиві RAM ємністю один кілобайт зберігається 1000 байт. У ньому зберігається більше тисячі байт, а саме 1024. Щоб справити враження знаючої людини, слід говорити "один кілобайт".

Один кілобайт пам'яті має вісім входів та вісім виходів для даних, а також десять входів «Адреса». Оскільки доступ до байтів здійснюється за допомогою десяти входів "Адреса", масив RAM зберігає 2^{10} байт. Щоразу, коли додаємо ще один вхід «Адреса», ми подвоюємо обсяг пам'яті. Кожен рядок наступної послідовності є подвоєнням пам'яті:

1 кілобайт = 1024 байт = 2^{10} байт ≈ 10^{3} байт;
2 кілобайти = 2048 байт = 2^{11} байт;
4 кілобайти = 4096 байт = 2^{12} байт;
8 кілобайт = 8192 байт = 2^{13} байт;
16 кілобайт = 16384 байт = 2^{14} байт;
32 кілобайти = 32 768 байт = 2^{15} байт;
64 кілобайти = 65536 байт = 2^{16} байт;
128 кілобайт = 131072 байт = 2^{17} байт;
256 кілобайт = 262144 байт = 2^{18} байт;
512 кілобайт = 524288 байт = 2^{19} байт;
1024 кілобайт = 1048576 байт = 2^{20} байт ≈ 10^{6} байт.

Зверніть увагу: вказані зліва кількості кілобайтів – ступені 2.

Ту ж логіку, яка дозволяє називати 1024 байт кілобайтом, ми можемо використовувати для того, щоб назвати 1024 кілобайт *мегабайтом* (приставка мега-від грецького «великий»). Мегабайт скорочено позначається Мбайт (Mbyte, рідше MB). Подвоєння пам'яті продовжується:

1 мегабайт = 1048576 байт = 2^{20} байт ≈ 10^{6} байт;

2 мегабайти = 2097152 байт = 2^{21} байт;
4 мегабайти = 4 194 304 байт = 2^{22} байт;
8 мегабайт = 8388608 байт = 2^{23} байт;
16 мегабайт = 16777216 байт = 2^{24} байт;
32 мегабайти = 33 554 432 байт = 2^{25} байт;
64 мегабайти = 67 108 864 байт = 2^{26} байт;
128 мегабайт = 134217728 байт = 2^{27} байт;
256 мегабайт = 268435456 байт = 2^{28} байт;
512 мегабайт = 536870912 байт = 2^{29} байт;
1024 мегабайт = 1073741824 байт = 2^{30} байт ≈ 10^{9} байт.

Одна тисяча двадцять чотири мегабайти складають *гігабайт* (приставка "гіга-" – від грецького "гігантський"), який позначається літерами Гб (GB).

Один *терабайт* (від грецького «жахливий») дорівнює 2^{40}байт (приблизно 10^{12}), або 1099511627776 байт. Терабайт позначається буквами Тб (TB).

Кілобайт дорівнює приблизно тисячі байтів, мегабайт – мільйону, гігабайт – мільярду, терабайт – трильйону байтів.

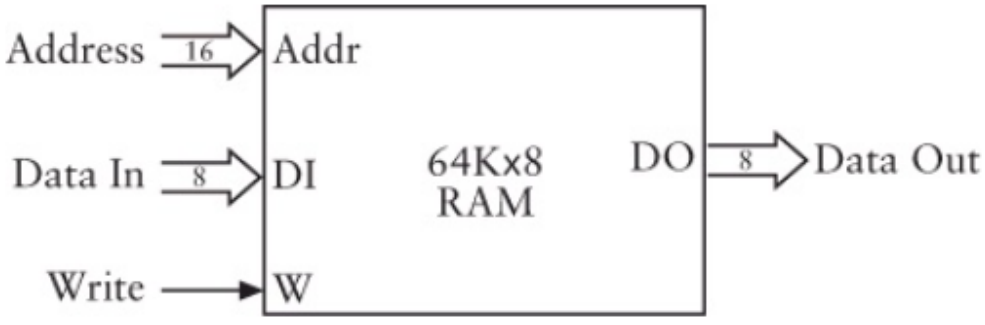
Петабайт дорівнює 2^{50}, або 1 125 899 906 842 624 байт (приблизно 10^{15}, або квадрильйон), а *екзабайт* дорівнює 2^{60}, або 1 152 921 504 606 846 976 байт (приблизно 10^{18}, або квінтильйон).

Зверніть увагу, що домашні комп'ютери, куплені в період написання цієї книги (1999 рік), зазвичай мали 32 або 64 (іноді 128) мегабайта пам'яті з довільним доступом. (Зауважте, я говорю про пам'ять RAM, а не про ємність жорстких дисків.) Це 33554432 байт, або 67108864 байт, або 134217728 байт.

Зрозуміло, у розмові люди використовують скорочення. Власник 65536 байтів пам'яті скаже: «У мене 64 кілобайти (і я гість з далекого 1980 року)». Користувач комп'ютера з пам'яттю 33554432 байт уточнить: «У мене 32 мега». А щасливчик, що має 1073741824 байт пам'яті, підкреслить: «У мене цілий гіг».

Іноді згадують *кілобіти* або *мегабіти* (зверніть увагу на використання слова "біти" замість "байти"), але це буває рідко. У більшості випадків, коли йдеться про пам'ять, повідомляють кількість байтів, а не бітів. (Щоб перетворити байти на біти, потрібно помножити їх на 8.) Зазвичай коли в розмові згадуються кілобіти або мегабіти, вони мають відношення до швидкості передачі даних по кабелю і використовуються в таких словосполученнях, як «кілобіти за секунду» та «мегабіти за секунду». Наприклад, коли йдеться про модем 56 К, мається на увазі саме 56 кілобіт на секунду, а не кілобайт; технологія Gigabit Ethernet забезпечує передачу одного гігабіту інформації на секунду тощо.

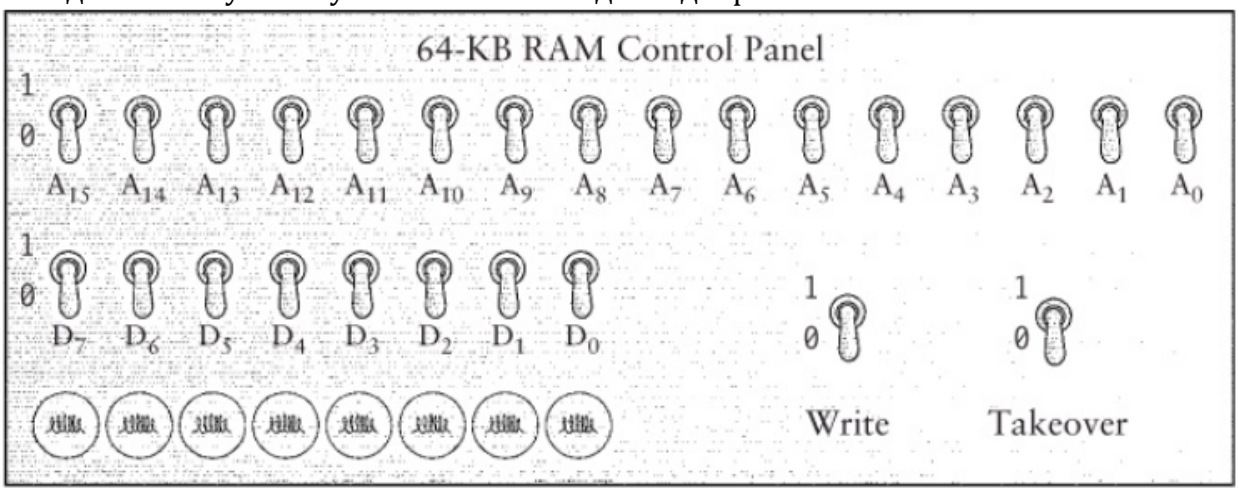
Тепер, навчившись збирати масиви RAM будь-якого потрібного розміру, намагатимемося не сильно захоплюватися. Давайте просто представимо масив RAM ємністю 65536 байт.



Чому 64 кілобайти? Чому не 32 кілобайти чи не 128 кілобайт? *Тому що 65536 – хороше кругле число.* Воно дорівнює 2^{16}. Цей масив RAM має 16-бітну адресу. Іншими словами, ця адреса дорівнює двом байтам. У шістнадцятковій системі числення значення цієї адреси знаходиться в діапазоні від 0000h до FFFFh.

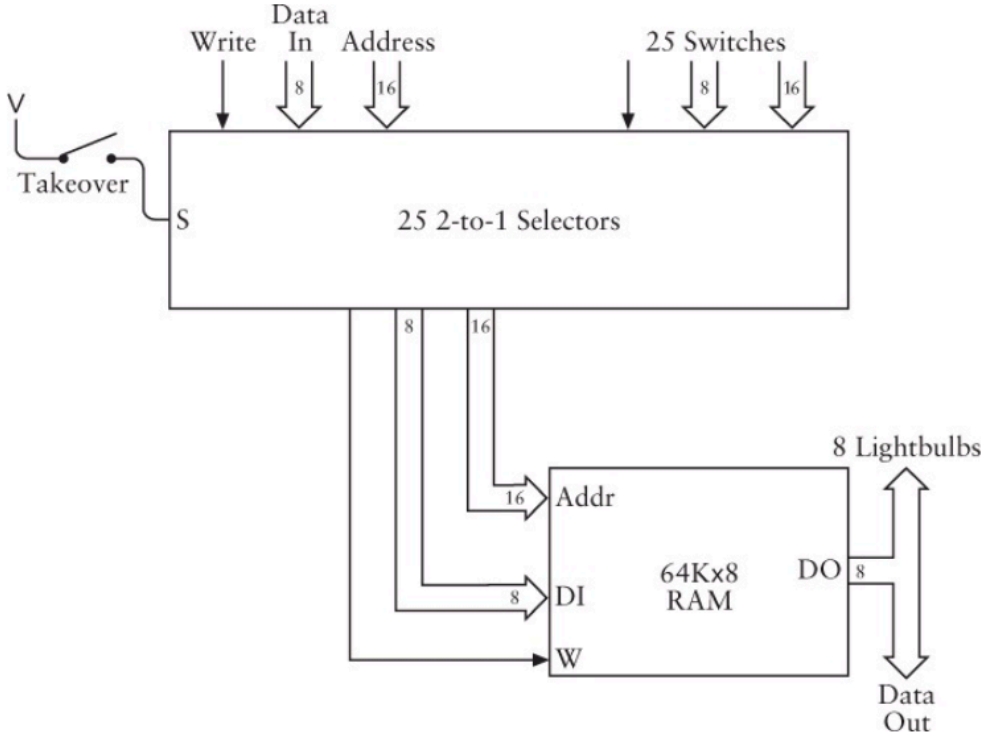
Раніше я натякнув на те, що обсяг пам'яті, що дорівнює 64 кілобайтам, був характерний для персональних комп'ютерів, куплених у 1980-х роках, хоча ця пам'ять збиралася і не з телеграфних реле. Чи вдасться створити такий пристрій, використовуючи реле? Сподіваюся, ви не розглядаєте таку можливість серйозно. Наша конструкція передбачає використання дев'яти реле для кожного біта пам'яті, тому для створення масиву RAM 64 Кб × 8 знадобиться майже п'ять мільйонів реле.

Нам також не завадить пульт керування, що дозволяє користуватися цією пам'яттю: записувати значення або зчитувати їх. Такий пульт повинен мати 16 перемикачів для вказівки адреси, вісім перемикачів для задання 8-бітного значення, яке ми хочемо записати, ще один перемикач для сигналу запису і вісім лампочок для відображення певного 8-бітного значення.



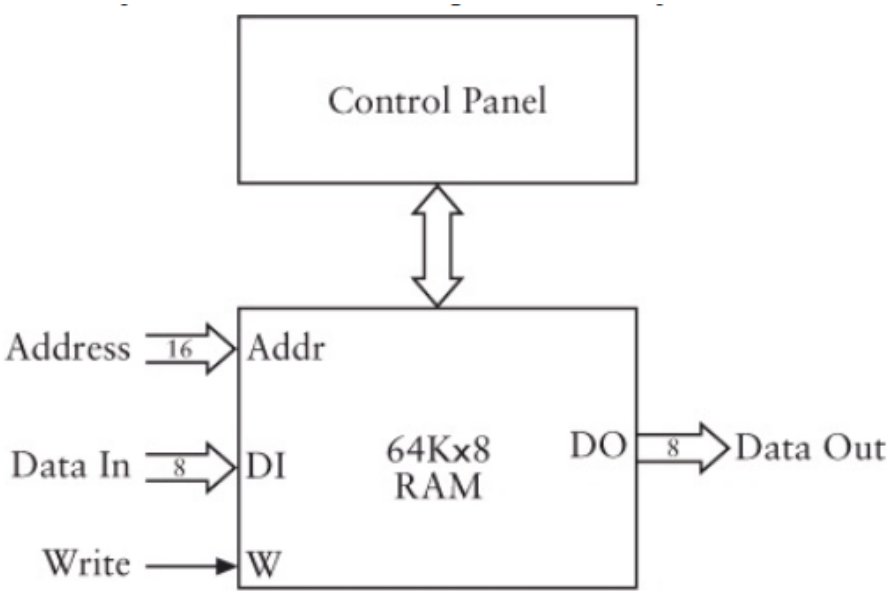
Усі перемикачі зображені у положенні «Вимкнено» (0). Я також додав перемикач із написом «Перехоплення». Мета цього перемикача – дозволити іншим схемам використовувати ту саму пам'ять, до якої підключено пульт керування. Коли цей перемикач встановлено у положення 0 (як показано малюнку), інші перемикачі пульта управління нічого не роблять. Однак, коли перемикач встановлений у положення 1, управління пам'яттю здійснюється виключно за допомогою пульта.

Для створення перемикача "Перехоплення" буде потрібно набір селекторів "2 на 1", а саме 25 штук: шістнадцять – для сигналів "Адреса", вісім – для перемикачів, що дозволяють вводити дані, і ще один – для перемикача "Запис". Нижче показано схему.



Коли перемикач "Перехоплення" розімкнуто (як показано на малюнку), на входи масиву RAM 64 Кбайт × 8 "Адреса", "Дані" і "Запис" надходять сигнали ззовні, як показано над лівим верхнім кутом блоку селекторів "2 на 1". Коли перемикач «Перехоплення» замкнено, сигнали на входи масиву RAM «Адреса», «Дані» та «Запис» надходять від перемикачів на пульті управління. У будь-якому випадку сигнали «Виведення даних» з масиву RAM надходять на вісім лампочок і, ймовірно, ще кудись.

Масив RAM 64 Кбайт × 8 з таким пультом управління можна зобразити в такий спосіб.



Коли перемикач «Перехоплення» замкнено, ви можете використовувати 16 перемикачів «Адреса» для вибору будь-якої з 65 536 адрес. Лампочки показують 8-бітове значення, яке зараз зберігається в пам'яті за цією адресою. Ви можете використовувати вісім перемикачів "Дані" для встановлення нового значення, а записати це значення в пам'ять за допомогою перемикача "Запис".

Масив RAM 64 Кб × 8 і пульт управління, безумовно, допоможуть вам відстежити будь-яке з 65536 8-бітних значень, яке може знадобитися. Однак ця конструкція дозволяє деяким іншим схемам використовувати значення, які ми зберегли в пам'яті, і записувати в неї нові.

Є ще одна *важлива* деталь, яку необхідно знати про пам'ять: коли в Темі 11 ви познайомилися з концепцією вентилів, я перестав малювати окремі реле, з яких складаються ці вентилі. Зокрема, відтоді я більше не вказував на схеми, що кожне реле підключено до якогось джерела живлення. Щоразу при спрацьовуванні реле електрика тече через котушку електромагніту і утримує металевий контакт на місці.

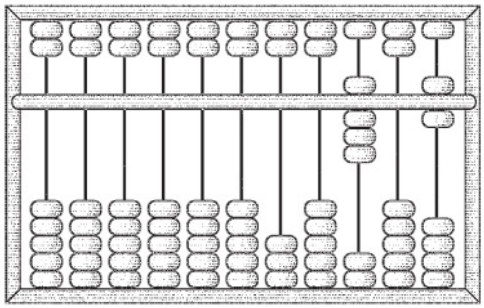
Отже, у вас є масив RAM 64 Кб × 8, весь обсяг якого заповнений вашими улюбленими байтами. Що станеться, якщо ви відключите його від джерела живлення? Всі електромагніти втратить свої магнітні властивості, і з гучним клацанням всі контакти реле повернуться у вихідне положення. А вміст цієї пам'яті? Він зникне назавжди!

Ось чому пам'ять з довільним доступом також називається *енергозалежною.* Для зберігання вмісту їй потрібне постійне енергопостачання.

Тема 11. Від рахівниць до мікросхем

Протягом усієї записаної історії люди винаходили різні розумні пристрої та машини, прагнучи хоч трохи спростити процес математичних обчислень. Незважаючи на те, що людський вид, мабуть, має вроджені здібності до обчислення, в цьому нам часто потрібна допомога. Нерідко ми ставимо такі складні завдання, з якими не можемо впоратися самостійно.

Розвиток систем числення можна вважати найбільш раннім інструментом, який допомагав людям вести облік товарів та майна. Представники багатьох культур, у тому числі давні греки та американські індіанці, мабуть, використовували для рахунку дрібні камінці та зерна. У Європі це призвело до винаходу рахункових дощок, на сході – рахівниць.



Незважаючи на те, що рахівниці зазвичай асоціюються з азіатськими культурами, вони, мабуть, були завезені торговцями в Китай приблизно 1200 року н. е.

Ніхто ніколи по-справжньому не отримував задоволення від множення та ділення, проте мало хто робив якісь дії для вирішення цієї проблеми. Шотландський математик Джон Непер (1550-1617) був одним із таких людей. Він винайшов логарифми для спрощення лічильних операцій. Добуток двох чисел – це сума їхніх логарифмів. Отже, якщо потрібно перемножити два числа, ви знаходите їх у таблиці логарифмів, додаєте числа з таблиці, а потім шукаєте у таблиці число, логарифм якого відповідає отриманій сумі. Побудова таблиць логарифмів протягом наступних 400 років займала одні найбільші уми, тоді як інші розробляли невеликі пристрої, які замінюють такі таблиці. Довга історія логарифмічної лінійки розпочалася з рахункової лінійки, створеної Едмундом Гюнтером (1581–1626) та вдосконаленою Вільямом Відредом (1574–1660). Історія цієї лінійки практично завершилася 1976 року, коли компанія Keuffel & Esser презентувала останню вироблену лінійку Смітсонівському інституту у Вашингтоні (округ Колумбія). Причиною її заходу став винахід ручного калькулятора.

Крім того, Непер винайшов ще один лічильний пристрій для полегшення множення, що складається з рядів чисел, вигравіруваних на кістці або розі, завдяки чому воно отримало назву кістки Непера. Найперший механічний калькулятор певною мірою представляв автоматизовану версію кісток Непера, створену приблизно 1620 року Вільгельмом Шиккардом (1592–1635). Інші калькулятори, сконструйовані з шестерінок і важелів, що зчіпляються, є майже такими ж старими пристосуваннями. Серед найвидатніших винахідників механічних калькуляторів можна виділити двох математиків та філософів: Блеза Паскаля (1623–1662) та Готфріда Вільгельма Лейбніца (1646–1716).

Напевно ви пам'ятаєте, яку складність створювало перенесення біта як у початковому 8-бітному суматорі, так і в комп'ютері, який, крім іншого, автоматизував операцію додавання чисел, що мають більше восьми розрядів. Спочатку це перенесення здається просто невеликим трюком, що використовується при додаванні, проте у випадку з суматорами перенесення біта – основна проблема. Якщо ви розробили суматор, який робить все, окрім перенесення біта, вважайте, що ви ні на крок не наблизилися до мети!

Ключовий фактор в оцінці старих обчислювальних машин – наскільки успішно вони справлялися з перенесенням. Наприклад, конструкція механізму перенесення, розроблена Паскалем, не дозволяла машині робити віднімання. Щоб відняте одне число з іншого, необхідно було доповнення до дев'яти, яке я продемонстрував у темі 13. Ефективні механічні калькулятори, якими могли користуватися люди, з'явилися лише наприкінці XIX століття.

Одним із цікавих винаходів, який мав значно вплинути на історію обчислень, а також на текстильну промисловість, був автоматичний ткацький верстат, розроблений Жозефом Жаккаром (1752–1834). У верстаті Жаккара, створеному в 1804 році, для задання візерунка тканини використовувалися металеві карти з пробитими в них отворами (на зразок перфокарт для самограючих піаніно). Найбільшим досягненням Жаккара став його чорно-білий автопортрет у шовку, на створення якого знадобилося близько десяти тисяч карт.

У XVIII столітті *обчислювачем* називалася людина, найнята спеціально для того, щоб робити обчислення. Великий попит мали таблиці логарифмів, а таблиці тригонометричних функцій широко застосовувалися для астрономічної навігації. Якщо вам потрібно було опублікувати новий набір таблиць, доводилося наймати численних «комп'ютерів», організовувати їх роботу, а потім систематизувати отримані дані. Зрозуміло, помилки могли виникнути на будь-якій стадії, починаючи від розрахунків та закінчуючи підготовкою до друку.

Прагненням позбавити математичні таблиці помилок керувався у своїй роботі Чарльз Беббідж (1791–1871), британський математик і економіст, практично сучасник Семюела Морзе.

У той час математичні таблиці (наприклад, логарифмів) не створювалися шляхом обчислення фактичного логарифма для кожного запису в таблиці. Це зайняло б занадто багато часу. Замість цього логарифми обчислювали для вибраних чисел, а потім числа між ними обчислювали шляхом інтерполяції, використовуючи те, що у відносно простих обчисленнях називають *різницями*.

Приблизно з 1820 року Беббідж вірив, що зможе спроектувати та створити машину, яка автоматизувала б процес побудови таблиці, аж до підготовки її до друку. Це усунуло б помилки. Він придумав «різницеву машину», яка по суті була великим механічним суматором. Багаторозрядні десяткові числа представлялися за допомогою зубчастих коліс, кожне з яких могло знаходитися в будь-якому з десяти положень. Негативні числа оброблялися з допомогою доповнень до десяти. Незважаючи на те, що ранні моделі довели працездатність конструкції Беббіджа, різницева машина так ніколи і не була завершена, оскільки створювалася на гранти британського уряду, яких, зрозуміло, ніколи не вистачало. У 1833 році Беббідж припинив роботу над нею.

Однак до цього часу у Беббіджа виникла найкраща ідея. Вона полягала у створенні так званої аналітичної машини, розробка конструкції якої займала вченого до самої смерті, причому йому вдалося фактично зібрати кілька невеликих моделей та частин цього механізму.

Аналітична машина – пристрій, найбільш наближений до комп'ютера з усього, що було створено в XIX столітті. Конструкція Беббіджа передбачала *сховище* (концептуально нагадує пам'ять комп'ютера) та *млин* (пристрій для виконання арифметичних операцій). Множення можна було робити шляхом багаторазового додавання, а ділення – шляхом багаторазового віднімання.

Найцікавіше полягає в тому, що аналітичну машину можна було запрограмувати за допомогою карт на кшталт тих, що використовувалися в ткацькому верстаті Жаккара. Як висловилася Августа Ада Байрон, графиня Лавлейс (1815–1852), у примітках до свого перекладу статті, написаної італійським математиком про аналітичну машину Беббіджа: «Можна сказати, що аналітична машина плете алгебраїчні візерунки подібно до того, як ткацький верстат».

Беббідж, здається, був першим, хто усвідомив важливість умовних переходів у комп'ютерах. Знову наведемо слова Ади Байрон: «Таким чином, *цикл* операцій слід розуміти як *будь-який набір операцій,* що виконується *більше одного разу. Цикл* є циклом незалежно від того, повторюється він *двічі* або невизначена кількість разів, оскільки саме *факт повторення* робить цикл тим, чим він є. У багатьох аналізованих випадках існують *рекурентні групи,* що складаються з одного чи кількох циклів, тобто *цикл циклу* чи *цикл циклів*».

Незважаючи на те, що різницева машина зрештою була сконструйована Георгом Шютцем та його сином Едвардом у 1853 році, машини Беббіджа залишалися забутими, про них згадали лише у 1930-х, коли люди почали досліджувати основи інформатики. На той час усе, чого досяг Беббідж, вже було перевищено пізнішими технологіями, і мало що міг запропонувати комп'ютерному інженеру XX століття, крім значно випереджаючого свій час передбачення автоматизації.

Ще одним поштовхом у розвиток інформатики послужила Конституція Сполучених Штатів Америки. Крім того, в ній міститься заклик до проведення перепису населення кожні десять

років. При проведенні перепису 1880 року збиралася інформація про вік, стать та національність. На аналіз даних пішло близько семи років.

Побоюючись того, що аналіз перепису 1890 року займе більше десятиліття, Бюро перепису населення вивчило можливість його автоматизації та обрало механізм, придуманий Германом Холлеритом (1860–1929), який працював статистом у 1880 році.

Холлерит планував використовувати картонні перфокарти розміром 168,278 × 82,551 мм. Малоймовірно, що Холлерит знав про те, як Чарльз Беббідж використав карти для програмування своєї аналітичної машини, проте він майже напевно був знайомий з використанням карток у ткацькому верстаті Жаккара. Отвори в цих картках були організовані в 24 стовпці по 12 позицій, що загалом давало 288 позицій. Ці позиції відповідали певним характеристикам людини, яка бере участь у переписі. Переписувач вказував ці особливості, пробиваючи прямокутні отвори розміром чверть дюйма у місці карти.

Читаючи книгу, ймовірно, ви настільки звикли мислити в термінах двійкових кодів, що могли припустити, що картка з 288 можливими отворами здатна зберігати 288 біт інформації. Проте ці карти використовувалися не так.

Наприклад, перфокарта, що застосовується під час перепису в суто двійковій системі, мала б одну позицію для статі. Вона була або пробита – у випадку, якщо опитуваний – чоловік, або не пробита – у випадку, якщо це жінка (або навпаки). Проте карти Холлерита передбачали дві позиції для статі: одна пробивалася чоловікам, інша – жінкам. Аналогічним чином переписувач вказував вік суб'єкта, пробиваючи два отвори. Перше позначало п'ятирічний діапазон: від 0 до 4, від 5 до 9, від 10 до 14 і т. д. Другий отвір пробивався в одній із п'яти позицій для позначення точного віку в цьому діапазоні. Для кодування віку були потрібні загалом 28 позицій на карті. При використанні двійкової системи потрібні були лише сім позицій для кодування будь-якого віку від 0 до 127 років.

Ми маємо пробачити Холлерита через те, що не впровадив двійкову систему для запису інформації, зібраної під час перепису населення. Перетворення віку на двійкові числа було непосильним завданням для тих, хто проводив перепис 1890 року. Крім того, існує практична причина, через яку використання перфокарт не може бути повністю заснована на двійковій системі. Двійкова система передбачає ймовірність того, що будуть пробиті всі (або майже всі) отвори, що зробить карту надзвичайно крихкою.

Дані перепису збирають так, щоб їх можна було підрахувати, тобто узагальнюють у *таблиці.* Зрозуміло, ви хочете знати, скільки людей живе у тому чи іншому районі, проте також цікаво отримати відомості про розподіл населення за віком. Для цього Холлерит сконструював табулятор – машину, в якій ручне керування поєднувалося з автоматизацією. Оператор притискав до кожної перфокарти прес із 288 пружними штирями. У тих місцях картки, де були пробиті отвори, ці штирі занурювалися в резервуар з ртуттю, що призводило до замикання електричного ланцюга, який активував електромагніт, який потім збільшував на одиницю значення десяткового лічильника.

Холлерит використовував електромагніти і в машині для сортування перфокарт. Наприклад, вам може знадобитися зібрати окрему вікову статистику за кожною професією. Спочатку потрібно сортувати карти за професіями, потім окремо для кожної з них зібрати дані за віком. Сортувальна машина використовувала той же ручний прес, що і табулятор, проте сортувальник застосовував електромагніти для того, щоб відкривати засувки одного з 26 відділень. У це відділення оператор опускав картку та вручну закривав засувку.

Цей експеримент з автоматизації перепису 1890 виявився надзвичайно успішним. Загалом було опрацьовано понад 62 мільйони карток. Вони містили вдвічі більше даних порівняно з тим, що вдалося зібрати в ході перепису 1880 року, а оброблені ці відомості були приблизно втричі швидшими. Холлерит та його винаходи стали відомі у всьому світі.

Герман Холлерит започаткував довгу послідовність подій. У 1896 році він заснував компанію Tabulating Machine Company, яка займалася здачею в оренду та продажем обладнання для роботи з перфокартами. До 1911 року в результаті пари злиттів вона перетворилася на Computing-Tabulating-Recording Company, або C-T-R. У 1915 році її президентом став Томас Джон Вотсон (1874–1956), який у 1924 році змінив назву на International Business Machines Corporation, або IBM.

До 1928 оригінальні карти, що використовувалися в переписі 1890, перетворилися на знамениті перфокарти IBM з 80 стовпцями і 12 рядками. Вони продовжували активно

використовуватися протягом понад 50 років, і навіть у наступні роки їх іноді називали *картами Холлерита.* Про еволюцію цих карт розповім докладніше у темах 20, 21 та 24.

Перш ніж перенестись у двадцяте століття, давайте переконаємося, що в нас склалося правильне уявлення про цю епоху. За очевидними причинами в цій книзі я приділяв пильну увагу винаходам, які є цифровими за своєю природою. До них відносяться телеграф, абетка Брайля, машини Беббіджа та карти Холлерита. При роботі з цифровими концепціями та пристроями ви можете легко подумати, що цифровим є весь світ. Однак відкриття та винаходи XIX століття були явно *не* цифровими. Дійсно, дуже мала частина природного світу, який ми сприймаємо за допомогою органів чуття, цифрова. Скоріше світ – це континуум, який нелегко уявити за допомогою чисел.

Незважаючи на те, що Холлерит використовував реле у своїх карткових табуляторах та сортувальниках, комп'ютери, створені на основі реле, які згодом стали називатися *електромеханічними,* з'явилися лише в середині 1930-х років. У цих машинах зазвичай використовувалися телеграфні реле, а реле, розроблені для маршрутизації телефонних викликів.

Ці перші релейні комп'ютери були схожі те що, що ми збирали у попередній темі (їх конструкція заснована на мікропроцесорах, створених 1970-х). Сьогодні для нас очевидно, що комп'ютери мають використовувати двійкові числа, але так було не завжди.

Інша відмінність нашого релейного комп'ютера від перших справжніх машин у тому, що ніхто в 1930-х роках не був настільки божевільним, щоб зібрати з реле пам'ять об'ємом 524288 біт! Вартість і вимоги до простору і потужності унеможливлювали створення такої пам'яті. Убогий обсяг доступної пам'яті використовувався виключно для зберігання проміжних результатів. Самі програми перебували на фізичному носії, наприклад, на паперовій стрічці з перфорацією. Справді, наш процес введення коду та даних на згадку – більш сучасна концепція.

Хронологічно перший релейний комп'ютер, мабуть, сконструював Конрад Цузе (1910-1995), який у 1935 році, будучи студентом-інженером, почав збирати машину у квартирі своїх батьків у Берліні. Ця машина використовувала двійкові числа, але у ранніх версіях застосовувалася механічна пам'ять, а не реле. Для програмування своїх комп'ютерів Цузе пробивав отвори у старій 35-міліметровій кіноплівці.

У 1937 році Джордж Стібіц (1904-1995) з Bell Telephone Laboratories приніс додому пару телефонних реле і зібрав на своєму кухонному столі однобітний суматор, який його дружина пізніше назвала "К-машиною" ("К" – означає "кухня"). Цей експеримент ліг в основу комп'ютера Complex Number Computer, створеного Bell Labs в 1939 році.

Тим часом студент випускного курсу Гарварда Ейкен (1900–1973) шукав спосіб виконання безлічі одноманітних обчислень, що призвело до співпраці Гарварда та IBM, в результаті якого було створено автоматичний обчислювач, керований послідовностями, який згодом отримав ім'я «Марк I». Роботу над цим пристроєм було завершено у 1943 році. Цей перший цифровий комп'ютер, здатний друкувати таблиці, зрештою реалізував мрію Чарльза Беббіджа. Комп'ютер "Марк II" був найбільшою релейною машиною, яка використовувала 13 тисяч реле. У Гарвардській обчислювальній лабораторії, яку очолював Ейкен, вперше було прочитано курс інформатики.

Реле підходили до створення комп'ютерів, але були неідеальні. Оскільки вони були механічними, їхня робота ґрунтувалася на згинанні металевої пластини. Після тривалої роботи реле могли зламатися, а також вийти з ладу через частинки бруду або паперу, що застрягли між контактами. Відомий випадок, коли в 1947 з реле комп'ютера «Марк II» в Гарварді була витягнута мошка. Грейс Хоппер (1906–1992), яка співпрацювала з Ейкеном з 1944 року, а пізніше стала відомим фахівцем у галузі мов програмування, приклеїла цю мошку в журнал із позначкою: «Перший відловлений баг».

Можлива заміна для реле – вакуумна лампа, розроблена Джоном Флемінгом (1849–1945) та Лі де Форестом (1873–1961) для радіо. На початку 1940-х років вакуумні лампи повсюдно використовувалися для посилення телефонних сигналів. Практично в кожному будинку був радіоприймач, наповнений трубками, що світяться, які посилювали радіосигнали, забезпечуючи їх чутність. Як і у випадку з реле, з вакуумних ламп можна зібрати вентилі І, АБО, І-НЕ та АБО-НЕ.

Не важливо, з чого зібрані вентилі – з реле або вакуумних ламп. Вентилі завжди можна об'єднати у суматори, селектори, дешифратори, тригери та лічильники. Все, що я говорив про

компоненти на основі реле в попередніх темах, залишається чинним при заміні реле вакуумними лампами.

Проте у вакуумних ламп спостерігалися свої недоліки: вони були дорогими, споживали багато електрики та виділяли багато тепла. Найбільша проблема полягала в тому, що лампи перегоряли. То справді був факт, з яким людям доводилося миритися. Власники лампових радіоприймачів звикли до необхідності періодично замінювати в них лампи. Телефонна система була спроектована з надмірною надійністю, тому лампи, що періодично перегорають, не становили великої проблеми (у будь-якому випадку ніхто не очікує від телефонної системи бездоганної роботи). А от якщо лампа перегорає у комп'ютері, це можна знайти не відразу. Крім того, в комп'ютері використовується так *багато* вакуумних ламп, що вони можуть перегорати в середньому кожні кілька хвилин.

Велика перевага використання вакуумних ламп у порівнянні з реле, це те що лампи можуть перемикатися з одного стану в інший приблизно за одну мільйонну частку секунди – за одну *мікросекунду.* Вакуумна лампа змінює стан (включається або вимикається) в тисячу разів швидше, ніж реле, яке перемикається з одного в інший стан в кращому випадку приблизно за одну мілісекунду, тобто тисячну частку секунди. Цікаво відзначити, що швидкість не представляла серйозної проблеми на ранніх етапах розвитку комп'ютерної індустрії, оскільки загальна швидкість обчислень була пов'язана зі швидкістю, з якою машина зчитувала програму з паперової або пластикової стрічки. Поки комп'ютери працювали на цьому принципі, перевага вакуумних ламп у порівнянні з реле не мала значення.

Починаючи з 1940-х років, вакуумні лампи стали витісняти реле в конструкції нових комп'ютерів. До 1945 року реле остаточно перестали використовуватися. Коли релейні машини називалися електромеханічними комп'ютерами, вакуумні лампи стали основою для перших *електронних* комп'ютерів.

У 1943 році у Великій Британії почав працювати комп'ютер «Колосс», що використовувався для розшифрування повідомлень, створених за допомогою німецької шифрувальної машини «Енігма». Над цим проектом (і деякими пізнішими британськими комп'ютерами) серед інших працював Алан Т'юрінг (1912–1954), який у наші дні відомий двома статтями. У першій, опублікованій 1937 року, він запровадив поняття «обчислюваність» – аналіз того, що можуть і чого не можуть зробити комп'ютери. Він також розробив абстрактну модель комп'ютера, яка відома під назвою машини Тьюринга. Друга відома стаття була присвячена штучному інтелекту. Автор представив тест для машинного інтелекту – тест Тьюринга.

В Електротехнічній школі Мура при Пенсільванському університеті Джон Екерт (1919–1995) та Джон Моучлі (1907–1980) розробили комп'ютер ENIAC (Electronic Numerical Integrator and Computer, електронний числовий інтегратор та обчислювач). У ньому використовувалися 18 тисяч вакуумних ламп, і комп'ютер закінчили наприкінці 1945 року. ENIAC, вага якого становила близько 30 тонн, можна вважати найбільшим комп'ютером в історії. До 1977 року у продажу вже були набагато швидші комп'ютери. Однак Еккерт і Моучлі не змогли запатентувати машину через заявку їхнього конкурента Джона Атанасова (1903-1995), який зібрав електронний комп'ютер раніше, який, проте, так ніколи і не заробив.

Комп'ютер ENIAC привернув увагу математика Джона фон Неймана (1903–1957). фон Нейман, що народився в Угорщині, проживав у Сполучених Штатах з 1930 року. Видатна людина, відома своєю здатністю виконувати в голові найскладніші арифметичні операції, фон Нейман був професором математики в Прінстонському інституті перспективних досліджень і вивчав усе – від квантової механіки до застосування теорії ігор в економіці.

Джон фон Нейман допоміг розробити комп'ютер EDVAC (Electronic Discrete Variable Automatic Computer, електронний автоматичний обчислювач з дискретними змінними), що був удосконаленою версією комп'ютера ENIAC. У статті 1946 року «Попереднє обговорення логічної конструкції електронної обчислювальної машини», написаної у співавторстві з Артуром Берксом і Германом Голдстайном, він описав кілька особливостей комп'ютера, завдяки яким машина EDVAC значно перевершувала ENIAC. Розробники EDVAC вирішили, що комп'ютер повинен використовувати двійкову систему числення. У машині ENIAC використовувалася десяткова. Крім того, комп'ютер повинен мати максимально можливий об'єм пам'яті, і ця пам'ять повинна зберігати і програмний код, і дані, що отримуються в процесі роботи. З комп'ютером ENIAC було не так. Програмування ENIAC здійснювалося за допомогою перемикачів та з'єднання кабелів. Ці

інструкції мали зберігатися у пам'яті послідовно і адресуватися з допомогою лічильника команд, у своїй допускалися умовні переходи. Такий принцип став відомий як *концепція програми, що запам'ятовується.*

Ці принципи були таким важливим кроком у розвитку інформатики, що сьогодні ми говоримо про них як про *архітектуру фон Неймана.* Комп'ютер, який ми зібрали в попередній темі, є класичною машиною фон Неймана. Однак архітектура фон Неймана має вузьке місце. Машина фон Неймана зазвичай витрачає значний час на вилучення інструкцій з пам'яті під час їх виконання. Нагадаємо, що остаточна конструкція комп'ютера з теми 17 передбачала, що при роботі з тією чи іншою інструкцією три чверті часу витрачається на її вилучення.

За часів комп'ютера EDVAC було недоцільно створювати із вакуумних ламп пам'ять великого об'єму. Натомість було запропоновано кілька дуже дивних рішень. Серед успішних було використання *пам'яті з ртутною лінією затримки,* у якій застосовувалися п'ятифутові трубки з ртуттю. З одного кінця трубки з інтервалом близько однієї мікросекунди в ртуть надсилалися слабкі імпульси. За одну мілісекунду вони досягали іншого кінця трубки, де детектувалися як звукові хвилі та вирушали назад. Таким чином кожна трубка з ртуттю могла зберігати близько 1024 біт інформації.

Тільки в середині 1950-х років було розроблено пам'ять, що складалася з великих масивів маленьких намагнічених металевих кілець, через які проходили дроти. Кожне таке кільце могло зберігати один біт інформації.

Джон фон Нейман був не єдиною людиною, яка міркувала про природу комп'ютерів у 1940-х роках.

Клод Шеннон також був впливовим мислителем. У темі 11 я обговорював його магістерську дисертацію 1938 року, в якій було встановлено взаємозв'язок між перемикачами, реле та булевою алгеброю. У 1948 році, працюючи в Bell Telephone Laboratories, він опублікував у Bell System Technical Journal статтю "Математична теорія зв'язку", де не тільки вперше вжив у пресі слово "біт", але й заклав основи розділу науки, відомої сьогодні як теорія *інформації.*

Теорія інформації вивчає можливість передачі цифрової інформації за наявності шуму (який зазвичай перешкоджає передачі всієї інформації), а також способи його компенсації. В 1949 Шеннон написав першу статтю про програмування комп'ютера для гри в шахи, а в 1952 розробив механічну мишу, керовану реле, яка могла знаходити вихід з лабіринту. Крім усього іншого, у Bell Labs Шеннон був добре відомий ще й своїм умінням їздити на одноколісному велосипеді, при цьому жонглюючи.

Норберт Вінер (1894–1964), який у віці 18 років у Гарварді отримав ступінь доктора філософії з математики, найбільш відомий завдяки своїй книзі «Кібернетика, або Управління та зв'язок у тварині та машині» (1948). Вінер придумав назву «*кібернетика*» (від грецького «керманич») для теорії про взаємозв'язок біологічних процесів у людях та тваринах з механікою комп'ютерів та роботів. У поп-культурі всюдисуща приставка «*кібер-*» тепер позначає все, що стосується комп'ютерів. Зокрема, мільйони пов'язаних через інтернет комп'ютерів називаються *кіберпростором.* Це слово було придумано письменником Вільямом Гібсоном і з'явилося в романі 1984 «Нейромант», написаному в жанрі *кіберпанку.*

У 1948 році компанія Eckert-Mauchly Computer Corporation (яка пізніше увійшла в Remington Rand) почала роботу над тим, чого мало стати першим доступним широкої аудиторії комп'ютером, – UNIVAC (Universal Automatic Computer, універсальний автоматичний комп'ютер). Він був закінчений у 1951 році, а перший екземпляр був доставлений до Бюро перепису населення. Комп'ютер UNIVAC дебютував в ефірі каналу CBS, де використовувався для прогнозування результатів президентських виборів 1952 року. Уолтер Кронкайт називав його електронним мозком. У 1952 році компанія IBM оголосила про випуск першої комерційної комп'ютерної системи – 701.

Так почалася довга історія використання комп'ютерів корпораціями та урядом. Якою б цікавою не була ця історія, зараз ми перейдемо до обговорення іншої тенденції, що зародилася 1947 року, яка дозволила зменшити вартість та розмір комп'ютерів, перетворивши їх на побутову техніку. Цей прорив у галузі електроніки ледь не залишився непоміченим.

Корпорація Bell Telephone Laboratories протягом багатьох років була місцем, де розумні люди мали можливість працювати практично над будь-яким проектом, що їх цікавив. На щастя, деякі з них захоплювалися комп'ютерами. Я вже згадував Джорджа Стібіца і Клода Шеннона, які зробили

значний внесок у розвиток обчислювальної техніки, працюючи в Bell Labs. Пізніше, у 1970-х роках, у Bell Labs була розроблена комп'ютерна операційна система Unix та мова програмування C, про які розповім пізніше.

Корпорація Bell Labs виникла 1 січня 1925 року, коли компанія American Telephone and Telegraph офіційно відокремила свої наукові та технічні дослідні підрозділи від решти бізнесу, створивши дочірнє підприємство. Основна мета Bell Labs полягала у розробці технологій поліпшення роботи телефонної системи. На щастя, це доручення було досить туманним і передбачало всілякі напрями досліджень, у тому числі очевидне і не втрачаюче своєї актуальності завдання, пов'язане з посиленням звукового сигналу, що передається по проводах, без його спотворення.

Починаючи з 1912 року компанія Bell System працювала над ламповими підсилювачами. Значна частина досліджень та розробок була спрямована на вдосконалення вакуумних ламп з метою їх використання у телефонній системі. Незважаючи на виконану роботу, вакуумні лампи, як і раніше, залишали бажати кращого. Вони були більшими, споживали багато електроенергії та згодом перегоряли. Проте вони не мали альтернативи.

Все змінилося 16 грудня 1947 року, коли два фізики з Bell Labs, Джон Бардін (1908–1991) та Уолтер Браттейн (1902–1987) зібрали підсилювач іншого типу з германієвої пластини – елемента, відомого як напівпровідник – і смужки золотої *фольги.* Через тиждень вони продемонстрували підсилювач своєму шефу Вільяму Шоклі (1910–1989). Це був перший *транзистор,* пристрій, який дехто вважає найважливішим винаходом XX століття.

Транзистор з'явився не на порожньому місці. За вісім років до цього, 29 грудня 1939 року, Шоклі написав у своєму записнику: «Сьогодні мені спало на думку, що в принципі можна створити підсилювач, який використовує замість вакуумних ламп напівпровідники». Після демонстрації першого транзистора багато років пішло на його доопрацювання. Лише у 1956 році Шоклі, Бардін та Браттейн отримали Нобелівську премію з фізики «за дослідження напівпровідників та відкриття транзисторного ефекту».

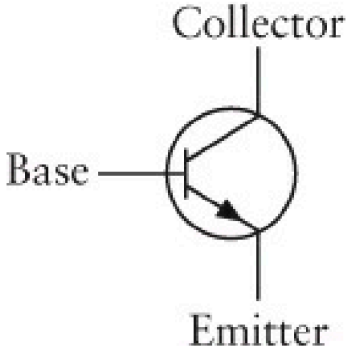
Як ви пам'ятаєте, електрони в атомі розподілені по оболонках, що оточують ядро. Мідь, золото та срібло характеризуються наявністю лише одного електрона на зовнішній оболонці їх атомів. Цей електрон може легко відірватися від решти атома і рухатися, створюючи електричний струм. Протилежністю провідників є ізолятори, наприклад гума та пластик, які практично не проводять електрику.

Германій і кремній (а також деякі з'єднання) називаються *напівпровідниками* не тому, що вони проводять електрику вдвічі гірше, ніж провідники, а тому, що їх провідністю можна керувати різними способами. Напівпровідники мають чотири електрони на зовнішній оболонці атома, що становить половину від їх максимально можливої кількості. У чистому напівпровіднику атоми утворюють дуже стійкі зв'язки, створюючи кристалічну решітку, подібну до кристалічних ґрат алмазу. Такі напівпровідники проводять електрику не дуже добре.

Однак напівпровідники можна *легувати,* тобто додати до них деякі домішки. Один тип домішок додає додаткові електрони до тих, які необхідні створення зв'язку між атомами. Вони називаються *напівпровідниками n-типу* (n – від англійської negative – «негативний»). В результаті додавання іншого типу домішок виходить *напівпровідник p-типу* (p – від positive – "позитивний").

Щоб створити підсилювач із напівпровідників, потрібно між двома шарами напівпровідника n-типу розташувати прошарок із напівпровідника p-типу. Пристрій, що вийшов, називається NPN-транзистором, а трьома його складовими частинами є *колектор, база* і *емітер.*

На малюнку наведено схематичне зображення NPN-транзистора.



Невелика напруга на базі управляє набагато більшим струмом, що проходить від колектора до емітера. За відсутності напруги на базі транзистор закривається.

Як правило, транзистори мають вигляд невеликих металевих циліндрів діаметром приблизно 6,4 міліметра з трьома проводами.

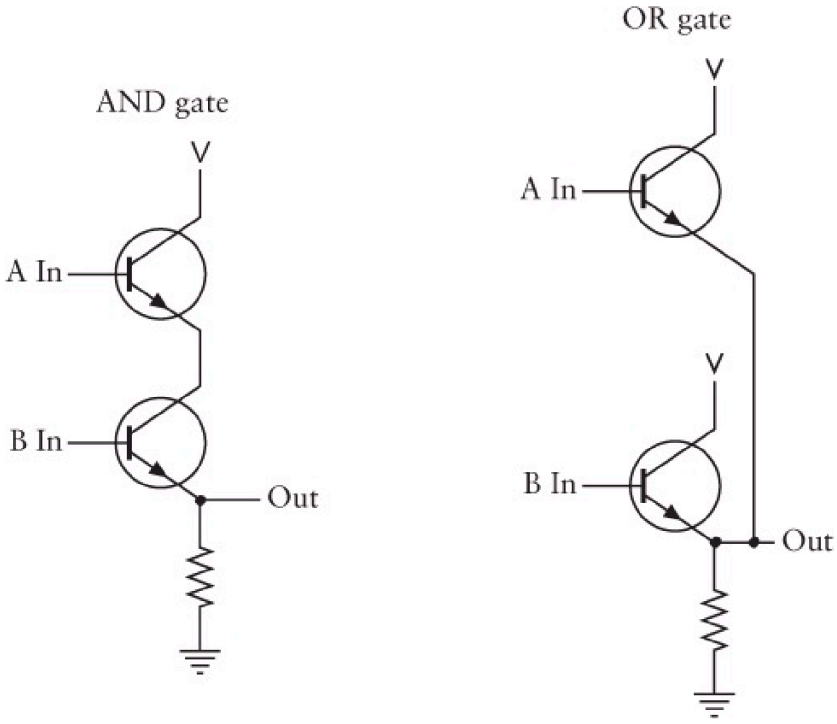
Розробка транзистора ознаменувала початок ери *твердотільної* електроніки, яка називається так тому, що транзистори не вимагають використання вакуумних ламп і створюються з твердих речовин, зокрема напівпровідників, найчастіше з кремнію (в наші дні). Крім того, що в порівнянні з вакуумними лампами транзистори мають куди менші розміри, вони споживають набагато менше електроенергії, генерують менше тепла і служать довше. Носити в кишені ламповий радіоприймач було неможливо. Транзисторний радіоприймач може живитися від невеликої батарейки та, на відміну від лампового, не нагрівається. Носіння транзисторного радіоприймача в кишені стало можливим для деяких щасливчиків, які розпакували подарунки різдвяного ранку 1954 року. У цих перших кишенькових радіоприймачах використовувалися транзистори, випущені компанією Texas Instruments, яка відіграла важливу роль напівпровідникової революції.

Однак *першим* комерційним застосуванням транзисторів було їх використання у слухових апаратах. На згадку про багаторічну роботу Олександра Белла з глухими людьми корпорація AT&T дозволила виробникам слухових апаратів застосовувати транзисторні технології, не сплачуючи за використання патентів. Перший транзисторний телевізор було випущено 1960 року, і сьогодні лампових приладів практично не знайти. (Хоча деякі меломани та електрогітаристи, як і раніше, віддають перевагу ламповим підсилювачам, а ніж їхнім транзисторним аналогам).

У 1956 році Шоклі залишив Bell Labs, щоб заснувати компанію Shockley Semiconductor Laboratories. Він переїхав до Пало-Альто (Каліфорнія), де виріс. Його компанія стала першим місцевим підприємством, що працювала у цьому напрямі. Згодом у цій місцевості з'явилися інші напівпровідникові та комп'ютерні компанії, а область на південь від Сан-Франциско тепер неофіційно називається Кремнієвою долиною.

Вакуумні лампи спочатку розроблялися для застосування в підсилювачах, проте їх також можна було використовувати як перемикачі в логічних вентилях. Те саме стосується транзистора. На наступному малюнку зображено вентиль І з урахуванням транзисторів, структура якого нагадує версію з реле. Тільки коли входи А і В дорівнюють логічній одиниці, тобто на базу подається позитивна напруга, обидва транзистори проводять струм, а вихід дорівнює 1. Резистор при цьому запобігає короткому замиканню.

З'єднавши два транзистори так, як показано на схемі праворуч, ви отримаєте вентиль АБО. У вентилі І емітер верхнього транзистора з'єднаний з колектором нижнього. У вентилі АБО колектори обох транзисторів підключені до джерела живлення. Емітери з'єднані між собою.



Як бачите, все, що ми дізналися про створення логічних вентилів та інших компонентів з реле, є справедливим і для транзисторів. Реле, лампи та транзистори спочатку розроблялися в основному для підсилювачів, проте з них можна зібрати логічні вентилі для комп'ютерів. Перші транзисторні комп'ютери було створено 1956 року, і через кілька років лампи перестали застосовуватися.

Транзистори, безумовно, роблять комп'ютери надійнішими, компактнішими та економічнішими. Але чи полегшують вони процес *складання*?

Насправді ні. Зрозуміло, транзистор дозволяє вмістити більше логічних вентилів у меншому просторі, проте вам, як і раніше, доведеться турбуватися про *з'єднання* всіх цих компонентів. З'єднати транзистори в логічні вентилі так само складно, як реле та вакуумні лампи. Цей процес ускладнюється ще меншим розміром, а також тим, що транзистори важче тримати. Якби ви вирішили зібрати комп'ютер, описаний у темі 17, і масив RAM ємністю 64 кілобайт з транзисторів, то *більша* частина часу на етапі проектування була б витрачена на розробку певної структури, де кріпилися всі компоненти. Основна фізична праця зводилася б до стомлювального з'єднання мільйонів транзисторів.

Як ми вже з'ясували, існують певні комбінації транзисторів, що часто зустрічаються. Пари транзисторів майже завжди з'єднані у вентилі. З вентилів часто збираються тригери, суматори, селектори чи дешифратори. Тригери об'єднуються в багатобітні клямки або масиви RAM. Зібрати комп'ютер було б простіше, якби транзистори були попередньо об'єднані у найпоширеніші конфігурації.

Цю ідею, мабуть, вперше запропонував британський фізик Джеффрі Даммер, який під час виступу у травні 1952 року сказав: «Я хотів би зазирнути у майбутнє. З появою транзистора і робіт щодо напівпровідників загалом сьогодні, очевидно, можна порушувати питання створення електронного устаткування у вигляді твердого блоку без будь-яких сполучних проводів. Цей блок може складатися з шарів ізолюючих, провідних, що перетворюють сигнал із змінного в постійний і посилюють сигнал матеріалів. Задання електронних функцій компонентів та їх з'єднання належним чином може бути виконане шляхом вирізування ділянок окремих шарів».

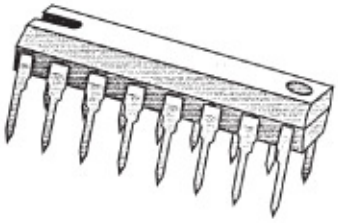
Проте на створення працюючого продукту пішло ще кілька років.

Нічого не знаючи про прогноз Даммера, у липні 1958 року Джек Кілбі з компанії Texas Instruments подумав, що на одному кристалі кремнію можна поєднати кілька транзисторів, а також резистори та інші електричні компоненти. Через шість місяців, у січні 1959 року, практично та ж ідея виникла у Роберта Нойса (1927-1990). Спочатку Нойс працював у компанії Shockley Semiconductor Laboratories, але в 1957 році він і ще семеро вчених покинули її, щоб заснувати корпорацію Fairchild Semiconductor Corporation.

У сфері технологій одночасний винахід – досить поширене явище. Незважаючи на те, що Кілбі винайшов свій пристрій за шість місяців до Нойса, а компанія Texas Instruments подала заявку на патент раніше, ніж Fairchild Semiconductor, Нойс отримав патент першим. Судові позови, що послідували за цим, завершилися з результатом, що влаштовує всіх, лише через десять років. Незважаючи на те, що Кілбі і Нойс ніколи не працювали разом, сьогодні вони вважаються співавторами *інтегральної мікросхеми* (ІС), яка зазвичай називається ч*іпом.*

Створення інтегральних схем – складний процес, який передбачає нашаровування тонких плівок легованого кремнію, протруєних у різних місцях для утворення мікроскопічних компонентів. Незважаючи на те, що розробка нової інтегральної мікросхеми передбачає великі витрати, масове виробництво дозволяє знизити ціни: чим більше виробляється мікросхем, тим дешевшими вони стають.

Кремнієвий чіп дуже тонкий і крихкий, тому він повинен бути надійно захищений корпусом, що дозволяє одночасно з'єднати його компоненти з іншими чіпами. Найчастіше інтегральні мікросхеми поміщаються в прямокутний пластиковий корпус *DIP* (dual inline package, корпус з дворядним розташуванням штиркових виводів) з 14, 16 або навіть 40 виводами.



Ось чіп із 16 виводами. Якщо ви візьмете його так, щоб невелика виїмка знаходилася зліва (як показано на малюнку), то виводи нумеруватимуться з 1 по 16 проти годинникової стрілки, починаючи з виводу в нижньому лівому куті і закінчуючи виводом в лівому верхньому куті. Штирки розташовані на відстані 2,5 мм один від одного.

Протягом 1960-х років ринок інтегральних мікросхем розвивався завдяки космічній програмі та гонці озброєнь. Першим масовим комерційним продуктом, що включав інтегральну мікросхему, був слуховий апарат, що розповсюджувався компанією Zenith 1964 року. У 1971 році компанія Texas Instruments почала продавати перший кишеньковий калькулятор, а компанія Pulsar – перший цифровий годинник. (Очевидно, в цифровому годиннику корпус ІС відрізняється від того, що ми обговорювали у наведеному вище прикладі.) Слідом за ними з'явилося безліч інших товарів, в конструкцію яких входили інтегральні мікросхеми.

У 1965 році Гордон Мур (тоді співробітник компанії Fairchild Semiconductor, а пізніше співзасновник корпорації Intel) зауважив, що технологія розвивається так, що починаючи з 1959 року кількість транзисторів, які можуть уміститися в одній мікросхемі, щорічно подвоюється, і передбачив збереження цієї тенденції. Фактично така технологія розвивалася трохи повільніше, тому закон Мура (як він став зрештою називатися) був скоригований і прогнозував подвоєння кількості транзисторів у мікросхемі кожні 18 місяців. Це, як і раніше, напрочуд швидкий розвиток, і закон Мура пояснює, чому домашні комп'ютери старіють лише за кілька років.

На початкових етапах розвитку технології про мікросхеми, що включають менше десяти логічних вентилів, говорили як про схеми з *малим рівнем інтеграції.* Схеми із *середнім рівнем інтеграції* (середні інтегральні схеми, СІС) включали від 10 до 100 вентилів, а схеми з *високим рівнем інтеграції* (великі інтегральні схеми, ВІС) – від 100 до 5000 вентилів. Потім було запроваджено такі поняття, як *надвисокий рівень інтеграції* (надвелика інтегральна схема, НВІС) – від 5 до 50 тисяч вентилів, *супернадвисокий рівень інтеграції* – від 50 до 100 тисяч вентилів та *ультрависокий рівень інтеграції* – понад 100 тисяч вентилів.

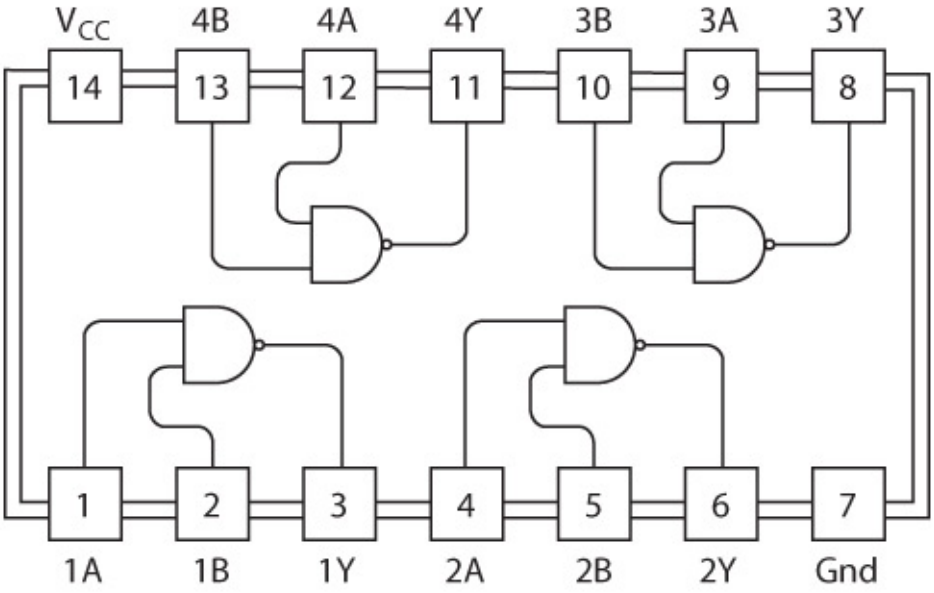
Решту цієї теми і всю наступну я пропоную провести в середині 1970-х, у тій давній епосі, коли ніхто ще не чув про фільм «Зоряні війни», а схеми НВІС ще тільки маячили на горизонті. Тоді виготовлення компонентів інтегральних схем використовувалося кілька різних технологій, кожна з яких визначає *сімейство* ІС. До середини 1970-х років переважали сімейства ТТЛ та КМОП.

Абревіатура ТТЛ розшифровується як *транзисторно-транзисторна логіка.* Якби в середині 1970-х ви працювали інженером-розробником цифрових ІС (збирали з ІС більші схеми), то вашою настільною книгою був би довідник з ТТЛ-мікросхем The TTL Data Book for Design Engineers,

вперше опублікований 1973 року компанією Texas Instruments. Він містив докладний опис інтегральних мікросхем ТТЛ серії 7400, що продаються Texas Instruments і деякими іншими компаніями, так званих тому, що номер кожної ІС в цьому сімействі починався з 74.

Кожна інтегральна схема серії 7400 складається з логічних вентилів, налаштованих певним чином. Деякі мікросхеми – прості логічні вентилі, з яких можна створити більші компоненти; інші – готові компоненти: тригери, суматори, селектори та дешифратори.

Перша ІС серії 7400, що має номер 7400, описана в довіднику The TTL Data Book як «чотирьохкратна двовходова позитивна схема І-НЕ». Це означає, що дана конкретна інтегральна схема має чотири двовходові вентилі І-НЕ. Вентилі І-НЕ називаються *позитивними,* оскільки наявність напруги відповідає значенню 1, а відсутність – значенню 0. На наступному малюнку



зображена мікросхема з 14 виводами і показано, як ці виводи співвідносяться з входами і виходами.

Діаграма – це вид мікросхеми зверху (виводи спрямовані вниз), при цьому виїмка в корпусі (згадана трохи раніше) розташована зліва.

Вивід 14 позначений символами V_{cc} і еквівалентний символу V, який використовував для позначення напруги. За традицією, будь-який подвійний підрядковий літерний індекс поруч із літерою V – джерело живлення. Літера *C* у цьому індексі – це вхід *колектора* транзистора, на який подається напруга. Вивід 7 позначений літерами GND, що означає земля (ground). Кожна інтегральна мікросхема, яку ви використовуєте, має бути підключена до джерела живлення та землі.

Для мікросхем ТТЛ серії 7400 значення V_{cc} має становити від 4.75 до 5.25 вольта. Іншими словами, напруга живлення – це п'ять вольт ± 5%. Якщо напруга впаде нижче 4.75 вольт, чіп може перестати працювати. Якщо вона перевищить значення 5.25, чіп може вийти з ладу. Зазвичай для живлення мікросхем ТТЛ не можна використовувати батарейки. Навіть якщо вдасться знайти п'ятивольтову батарейку, недостатня точність напруги зробить її невідповідним джерелом живлення цих чіпів. Як правило, мікросхеми ТТЛ потребують живлення від розетки.

Кожен із чотирьох вентилів І-НЕ мікросхеми 7400 має два входи та один вихід. Вони працюють незалежно один від одного. У попередніх темах ми говорили, що вхідний сигнал може мати значення 1 (за наявності напруги), або значення 0 (при відсутності напруги). Насправді вхідний сигнал одного з цих вентилів І-НЕ може змінюватись від нуля вольт (земля) до п'яти вольт (V_{cc}). У мікросхемі ТТЛ напруга у діапазоні від 0 до 0.8 вольта, відповідає логічному нулю, а напруга від двох до п'яти вольт – логічній одиниці. Напруги від 0.8 до 2.0 вольта слід уникати.

Напруга на виході вентиля ТТЛ, що становить близько 0.2 вольта, зазвичай відповідає логічному нулю, а 3.4 вольта – логічній одиниці. Оскільки ці значення можуть дещо відхилятися, то, говорячи про входи та виходи інтегральних схем, іноді замість 0 і 1 люди використовують такі поняття, як *низький* та *високий* рівень сигналу. Більше того, низька напруга може означати логічну одиницю, а висока логічний нуль. Така конфігурація характеризується *негативною*

логікою. У назві «чотирьохкратна двовходова позитивна схема І-НЕ» слово «позитивна» означає схему з позитивною логікою.

Значення напруги на виході вентиля ТТЛ 0.2 вольта (логічний нуль) та 3.4 вольта (логічна одиниця) знаходяться в допустимих межах – від 0 до 0.8 для логічного нуля та від двох до п'яти вольт для логічної одиниці. Таким чином мікросхеми ТТЛ ізолюються від *шумів.* Одиничний вихідний сигнал може зменшитися приблизно на 1.4 вольта, але, як і раніше, залишиться досить високим, щоб його можна було кваліфікувати як одиничний вхідний сигнал. Нульовий вихідний сигнал може збільшитися на 0.6 вольта, але залишиться досить низьким, щоб категоризувати вхідний нульовий сигнал.

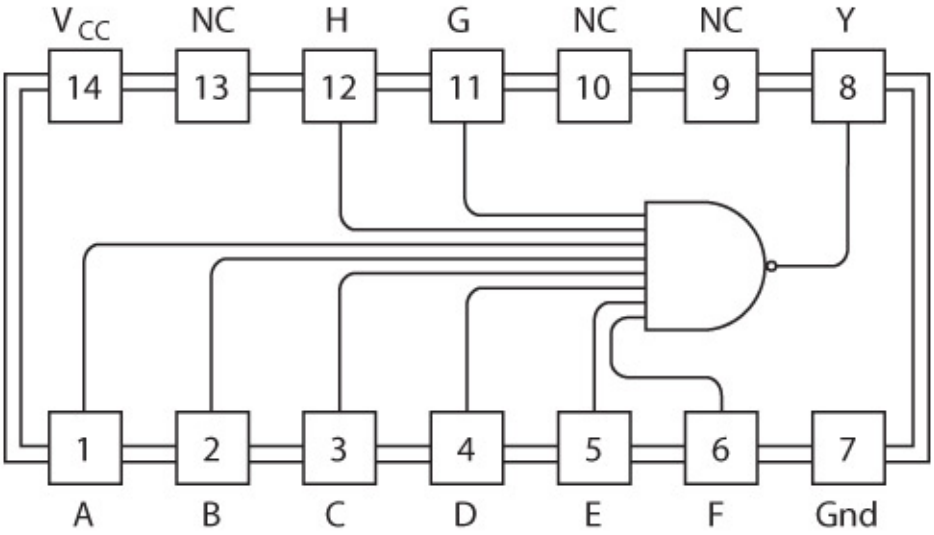
Найважливішим параметром конкретної інтегральної схеми є *час установки.* Цей час, необхідний для того, щоб зміна вхідного сигналу призвела до зміни вихідного сигналу.

Час установки мікросхем зазвичай вимірюється у *наносекундах.* Наносекунда дуже короткий проміжок часу. Одна тисячна частина секунди – це мілісекунда. Мільйонна частина секунди – мікросекунда. Наносекунда – це одна мільярдна частина секунди. Час установки для вентилів І-НЕ в мікросхемі 7400 гарантовано не перевищує 22 наносекунд. Це 0.000000022 секунди, або 22 мільярдні частки секунди.

Якщо вам важко уявити такий маленький проміжок часу, ви не самотні. Ми можемо охопити його лише думкою. Наносекунди набагато коротше всього, що є людським досвідом, тому вони назавжди залишаться за межами нашого розуміння. Кожне пояснення лише робить наносекунду більш незбагненною. Наприклад, я можу сказати, що якщо ви тримаєте цю книгу на відстані 30 сантиметрів від обличчя, то наносекунда – це час, за який світло долає відстань від сторінки до ока. Однак чи ви стали краще розуміти, що таке наносекунда?

Проте саме завдяки таким коротким проміжкам часу, як наносекунда, можливе існування комп'ютерів. Як ви бачили в темі 17 комп'ютерний процесор виконує дуже прості дії: переміщає байт з пам'яті в регістр, додає з іншим байтом і повертає результат назад в пам'ять. Єдина причина, через яку результат роботи комп'ютера – щось істотне (йдеться про реальний комп'ютер, а не той, що описувався у темі 17), у тому, що це операції відбуваються дуже швидко. Як сказав Роберт Нойс: «Якщо змиритися з поняттям наносекунди, то комп'ютерні операції концептуально досить прості».

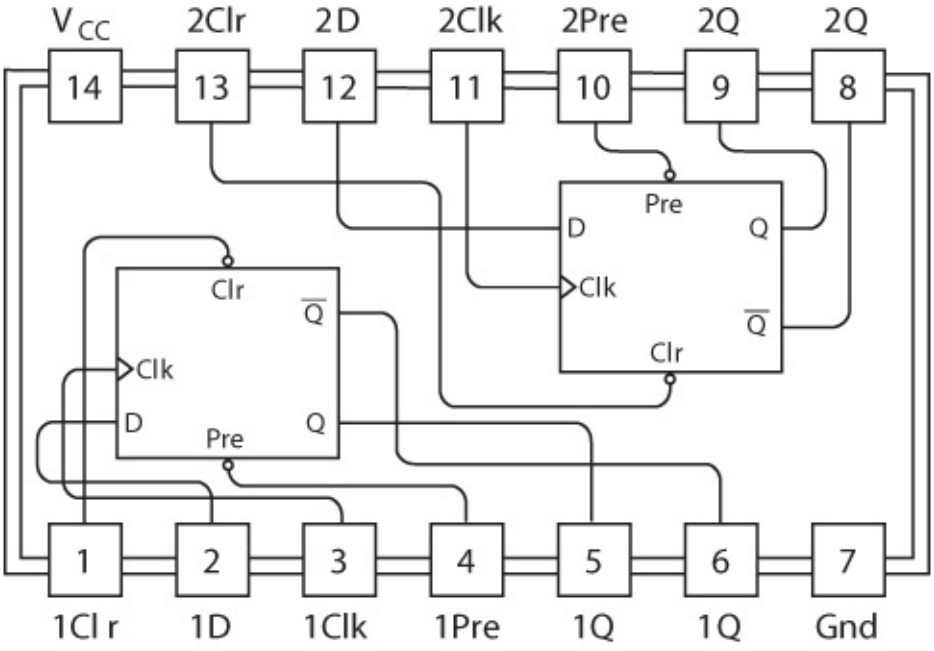
Давайте продовжимо вивчення довідника з мікросхем ТТЛ. У цій книзі ви побачите багато вже знайомих компонентів. Мікросхема 7402 містить чотири двовходові вентилі АБО-НЕ, мікросхема 7404 – шість інверторів, мікросхема 7408 – чотири двовходові вентилі І, мікросхема



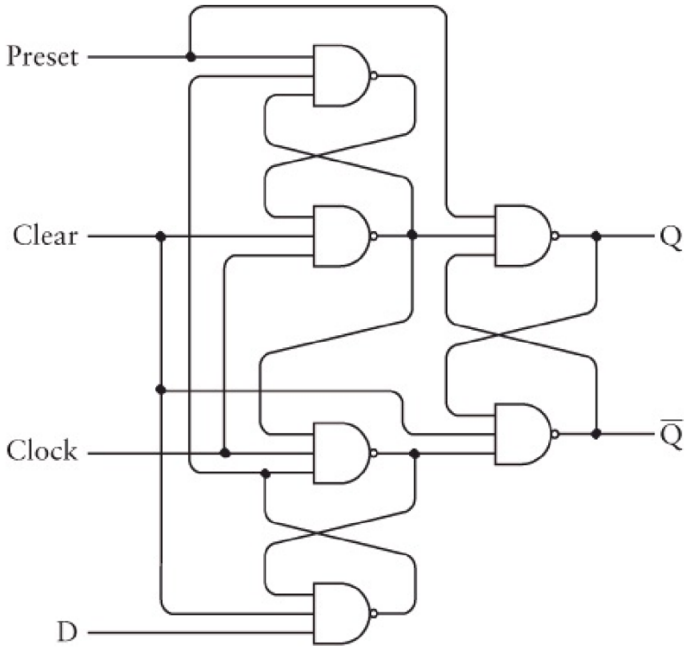
7432 – чотири двовходові вентилі АБО, а мікросхема 7430 – восьмивходовий вентиль І-НЕ.

Абревіатура *NC* означає no connection – "не підключено".

Мікросхема 7474 також може здатися знайомою. Це здвоєний D-тригер зі скиданням та передустановкою, що спрацьовує по фронту, схема якого виглядає так.



У довідник з мікросхем ТТЛ включена логічна схема для кожного з тригерів.



Ця схема може здатися схожою на схему, наведену в кінці, за винятком того, що я використовував вентилі АБО. Наведена в довіднику з мікросхем ТТЛ таблиця логіки також трохи відрізняється.

Inputs				Outputs	
Pre	**Clr**	**Clk**	**D**	**Q**	$\overline{Q}$
L	H	X	X	H	L
H	L	X	X	L	H
L	L	X	X	H*	H*
H	L	↑	H	H	L
H	H	↑	L	L	H
H	H	L	X	Q_0	$\overline{Q}_0$

У таблиці *H* означає *високий* (high), а *L* – *низький* (low) рівень сигналу. Якщо хочете, можете вважати ці позначення одиницею та нулем. У моєму тригері входи «Скидання» та «Установка» зазвичай дорівнюють 0; у разі вони зазвичай рівні 1.

Далі в довіднику до мікросхем ТТЛ ви виявите, що мікросхема 7483 – це 4-бітний двійковий повний суматор, мікросхема 74151 – селектор з вісьмома входами і одним виходом, 74154 – дешифратор з чотирма входами і 16 виходами, 74175 – зчетверний D-тригер зі скиданням. Ви можете використовувати дві з перерахованих мікросхем для створення 8-бітної клямки.

Отже, тепер ви знаєте, звідки взялися різні компоненти, які я використовував у Темі 11 – з довідника із мікросхем ТТЛ.

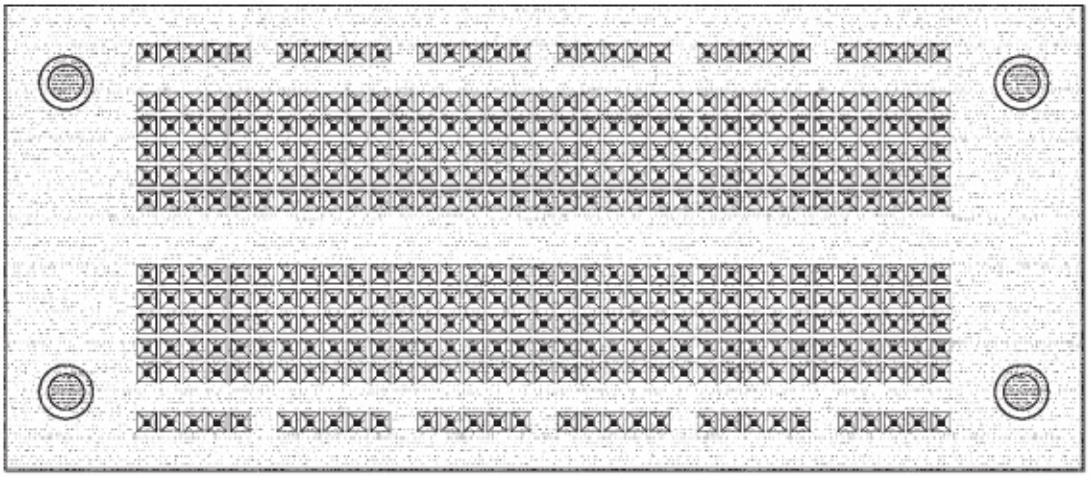
Як інженер-розробник цифрових ІС, ви витратили б безліч годин на читання довідника з мікросхем ТТЛ і вивчення існуючих чіпів. Освоївши інструменти, ви можете зібрати з мікросхем ТТЛ комп'ютер, описаний у темі 17. Поєднати між собою мікросхеми набагато простіше, ніж окремі транзистори. Однак ви навряд чи захотіли б використовувати схеми ТТЛ для створення масиву RAM об'ємом 64 кілобайт. Об'єм найбільш ємного чіпа RAM, описаного в довіднику The TTL Data Book for Design Engineers 1973, становив всього 256 × 1 біт. Для створення масиву RAM об'ємом 64 кілобайт вам знадобилося б 2048 таких чіпів! Мікросхеми ТТЛ ніколи не були оптимальною технологією створення пам'яті. До цієї теми я повернуся у темі 21.

Ймовірно, ви вирішите використати осцилятор. Незважаючи на можливість підключення виходу ТТЛ-інвертора до його входу, краще мати осцилятор з більш передбачуваною частотою. Такий осцилятор можна легко зібрати, використовуючи кристал кварцу, який поміщається у невеликий циліндричний плоский корпус з двома виводами. Ці кристали вібрують з певною частотою, яка зазвичай становить щонайменше мільйон циклів на секунду. Мільйон циклів за секунду відповідає частоті один *мегагерц.* Якби комп'ютер, описаний у Темі 17 був зібраний з мікросхем ТТЛ, він би нормально працював з тактовою частотою десять мегагерц. На виконання кожної інструкції витрачалося б 400 наносекунд. Це, безумовно, в багато разів перевищує швидкість роботи релейних пристроїв.

Іншим популярним сімейством чіпів є *КМОП* (комплементарна структура метал – оксид – напівпровідник), або CMOS (complementary metal-oxide-semiconductor). Якби в середині 1970-х у свій вільний час ви збирали схеми з чіпів КМОП, то як довідник могли б використовувати книгу CMOS Databook, опубліковану компанією National Semiconductor. Ця книга містить інформацію про мікросхеми КМОП серії 4000.

Необхідна потужність мікросхем ТТЛ – від 4,75 до 5,25 вольта, для мікросхем КМОП – від 3 до 18 вольт. Досить великий діапазон! Крім того, мікросхеми КМОП споживають набагато менше енергії порівняно з ТТЛ-чіпами, що уможливлює створення на їх основі невеликих пристроїв, що працюють від батарейок. Недолік мікросхеми КМОП – низька швидкість роботи. Наприклад, гарантований час установки 4-бітного повного суматора КМОП 4008, що працює від напруги 5 вольт, – 750 наносекунд. Швидкість збільшується зі зростанням напруги і становить 250 наносекунд при десяти вольтах і 190 наносекунд – при 15 вольтах. Однак, за цим показником пристрій на основі мікросхем КМОП сильно відстає від 4-бітного ТТЛ-суматора, час установки якого 24 наносекунди. (Двадцять п'ять років тому компроміс між швидкістю мікросхеми ТТЛ та

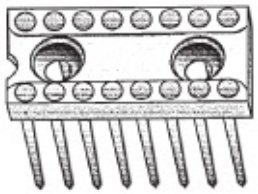
низьким енергоспоживанням мікросхеми КМОП був досить явним. Сьогодні існують версії ТТЛ-чіпів з малим енергоспоживанням та високошвидкісні версії мікросхем КМОП.)



Насправді з'єднання мікросхем починається на пластиковій *макетної платі.*

Кожні п'ять отворів електрично з'єднані під пластмасовою основою. Мікросхема вставляється в макетну плату так, щоб вона спиралася на довгу центральну борозну, а її виводи потрапляли в отвори з обох боків. Кожен вивід ІС при цьому електрично поєднується з чотирма іншими отворами. Мікросхеми об'єднуються за допомогою дротів, що вставляють в інші отвори.

Ви можете забезпечити постійне з'єднання мікросхем, використовуючи технологію під назвою *монтаж накруткою.* У цьому випадку кожна мікросхема вставляється у гніздо з довгими



квадратними штирками.

Кожен штирок відповідає виходу мікросхеми. Самі гнізда розташовуються у тонких перфорованих платах. На звороті плати ви використовуєте спеціальний моточний агрегат для того, щоб щільно обмотати штирьок тонким ізольованим проводом. Гострі краї штирка прориваються ізоляцію, завдяки чому між штирком і проводом виникає електричне з'єднання.

Якби ви займалися виробництвом конкретного пристрою на основі ІС, ймовірно, використали б *друковану плату.* У минулі часи її міг виготовити навіть аматор. Плата – це пластина з отворами, вкрита тонким шаром мідної фольги. Ті ділянки фольги, які потрібно зберегти, покриваються кислотостійкою речовиною, після чого решта протравлюється кислотою. Потім ви можете припаяти гнізда ІС (або самі ІС) безпосередньо до мідного покриття. Однак через велику кількість взаємозв'язків між ІС області мідної фольги, що залишилася, зазвичай виявляється недостатньо, тому друковані плати, що виготовляються промисловим способом, мають кілька рівнів міжз'єднань.

На початку 1970-х стало можливим використовувати ІС для збирання комп'ютерного процесора на єдиній платі. А розміщення всього процесора в одному чіпі було лише питанням часу. Незважаючи на те, що компанія Texas Instruments запатентувала однокристальний комп'ютер у 1971 році, честь його створення належить компанії Intel, заснованої в 1968 році колишніми співробітниками Fairchild Semiconductors Робертом Нойсом і Гордоном Муром. Першим важливим продуктом компанії Intel в 1970 став чіп пам'яті з найбільшою на той момент ємністю 1024 біт.

Компанія Intel займалася розробкою мікросхем для програмованого калькулятора, який збиралася виробляти японська Busicom, коли її інженери вирішили використати інший підхід. Як зазначив інженер Intel Тед Хофф, "замість калькулятора з можливістю програмування я хотів створити комп'ютер загального призначення, запрограмований на виконання функцій

калькулятора". Це призвело до розробки Intel 4004, першого комп'ютера в чіпі, або *мікропроцесора.* Продаж мікросхеми 4004 почався в листопаді 1971 року, вона містила 2300 транзисторів. (Згідно з законом Мура, мікропроцесори, створені через 18 років, повинні містити приблизно в 4000 разів більше транзисторів, або близько десяти мільйонів. Це передбачення виявилося досить точним.)

Тепер, коли вам відома кількість транзисторів у мікросхемі 4004, опишу ще три важливі характеристики.

По-перше, схема 4004 – це 4-розрядний мікропроцесор, отже, ширина шини даних процесора становила всього чотири біти. При додаванні чи відніманні чисел він був здатний обробляти лише чотири біти за один такт. Навпаки, комп'ютер, розроблений у темі 17, має 8-розрядні шини даних і є 8-розрядним процесором. Як ми невдовзі побачимо, 8-розрядні мікропроцесори швидко перевершили 4-розрядні. Однак на цьому ніхто не зупинився. Наприкінці 1970-х з'явилися 16-розрядні мікропроцесори. Якщо ви згадаєте комп'ютер з Теми 17 і кілька команд, необхідних для додавання двох 16-розрядних чисел за допомогою 8-розрядного процесора, оцініть перевагу 16-розрядного процесора. У середині 1980-х було розроблено 32-розрядні мікропроцесори, які відтоді залишаються стандартом для домашніх комп'ютерів.

По-друге, максимальна *тактова частота* мікросхеми 4004 становила 108 тисяч циклів за секунду, або 108 кілогерц. Тактова частота – це максимальна швидкість тактового генератора, який можна підключити до мікропроцесора, щоб змусити його працювати. Вища швидкість може призвести до некоректної роботи. До 1999 тактова частота мікропроцесорів, призначених для домашніх комп'ютерів, досягла позначки 500 мегагерц, що приблизно в 5000 разів швидше в порівнянні з аналогічним показником мікросхеми 4004.

По-третє, обсяг *пам'яті, що адресується* мікросхеми 4004 становив 640 байт. Хоча це значення видається смішно маленьким, воно відповідало ємності мікросхем пам'яті того часу. Як ви побачите далі, вже через два роки мікропроцесори могли звертатися до 64 кілобайтів пам'яті, що відповідає можливостям комп'ютера з теми 17. В 1999 мікропроцесори Intel могли звертатися до 64 терабайт пам'яті, проте це занадто багато, враховуючи, що оперативна пам'ять більшості домашніх комп'ютерів не перевищувала 256 мегабайт.

Ці три показники нічого не говорять про *можливості* комп'ютера. Наприклад, 4-розрядний процесор може додавати 32-розрядні числа, просто поділяючи їх на 4-розрядні фрагменти. У деяких відношеннях всі цифрові комп'ютери однакові. Якщо апаратне забезпечення одного процесора може вирішити завдання, яке не під силу іншому, третій здатний впоратися із завданням за допомогою програмного забезпечення; зрештою всі вони роблять те саме. Саме це малося на увазі в статті Алана Тьюринга 1937 про обчислюваності.

Швидкодія – одна з найважливіших причин, через яку ми взагалі використовуємо комп'ютери. Максимальна тактова частота надає очевидний вплив на загальну швидкість роботи процесора, оскільки визначає швидкість виконання кожної команди. Ширина шини даних процесора також впливає на його швидкодію. Незважаючи на те, що 4-розрядний процесор здатний додавати 32-розрядні числа, при вирішенні цього завдання він сильно поступається 32-розрядному. Проте ви можете не відразу усвідомити, який вплив на швидкодію надає максимальний обсяг пам'яті процесора, що адресується. Спочатку може здатися, що пам'ять, що адресується, не має нічого спільного зі швидкістю роботи, а замість цього вказує на обмеження здатності процесора виконувати певні функції, які можуть вимагати великого обсягу пам'яті. Однак процесор завжди може обійти цю межу, використовуючи деякі адреси пам'яті, щоб керувати іншими засобами для збереження та отримання інформації. (Уявіть, що кожен байт, записаний у конкретну комірку пам'яті, – це фактично отвір, пробитий у паперовій стрічці, а кожен байт, зчитаний із цієї комірки, зчитується зі стрічки.) Однак цей процес уповільнює роботу всього комп'ютера. Знову проблема у швидкодії. Звичайно, ці три показники лише приблизно відображають швидкість роботи мікропроцесора. Вони нічого не повідомляють про внутрішню архітектуру мікропроцесора або про ефективність та можливості команд, написаних машинною мовою. У міру ускладнення процесорів багато поширених завдань, які раніше виконували програмне забезпечення, включалися у функціонал самого процесора. У наступних темах наведу приклади, що відображають цю тенденцію.

Всі цифрові комп'ютери мають однакові можливості. Вони не можуть робити нічого, що виходило б за межі потенціалу примітивної обчислювальної машини, розробленої Аланом

Т'юрінгом. При цьому швидкість процесора, *звичайно,* впливає на загальну корисність комп'ютерної системи. Наприклад, будь-який комп'ютер, який рахує повільніше за людський мозок, є марним. І ми навряд чи зможемо подивитися фільм на екрані сучасного комп'ютера, якщо процесору буде потрібна ціла хвилина для того, щоб намалювати один кадр.

Повернемося до середини 1970-х. Незважаючи на свої обмеження, мікросхема 4004 стала початком нової доби. До квітня 1972 року компанія Intel випустила мікросхему 8008 – 8-розрядний мікропроцесор з тактовою частотою 200 кілогерц і пам'яттю, що адресується об'ємом 16 кілобайт. (Бачите, як легко описати процесор за допомогою всього трьох параметрів?) А в 1974 році протягом п'ятимісячного періоду компанії Intel і Motorola випустили мікропроцесори, що перевершують за своїми характеристиками мікросхему 8008. Ці два мікропроцесори змінили світ.

Тема 12. Два класичні мікропроцесори

Мікропроцесор, що є сукупністю всіх компонентів центрального процесорного пристрою (ЦПУ), об'єднаних однією кремнієвою мікросхемою, з'явився 1971 року. Початок був скромним: перший мікропроцесор Intel 4004 містив близько 2300 транзисторів. Сьогодні кількість транзисторів у мікропроцесорах, призначених для домашніх комп'ютерів, наблизилася до позначки десять мільйонів.

Проте принцип його роботи на фундаментальному рівні залишився тим самим. Величезна кількість додаткових транзисторів у сучасних чіпах може виконувати цікаві функції; на перших етапах вивчення воно швидше відволікає, ніж прояснює ситуацію. Для чіткого уявлення роботи мікропроцесора розглянемо перші версії.

Мікропроцесори, про які йтиметься, з'явилися 1974 року. У квітні Intel представила свою мікросхему 8080, а у серпні Motorola, яка з початку 1950-х років займалася виробництвом напівпровідників та транзисторних пристроїв, випустила мікросхему 6800. У той рік на ринку з'явилися не тільки ці мікропроцесори. Тоді ж компанія Texas Instruments представила свою 4-розрядну мікросхему TMS1000, яка встановлювалася в багато калькуляторів, іграшок та інших пристроїв, а National Semiconductor випустила перший 16-розрядний мікропроцесор PACE. Озираючись назад, можна сказати, що мікросхеми 8080 та 6800, безумовно, відіграли найбільш значну роль в історії розвитку обчислювальної техніки.

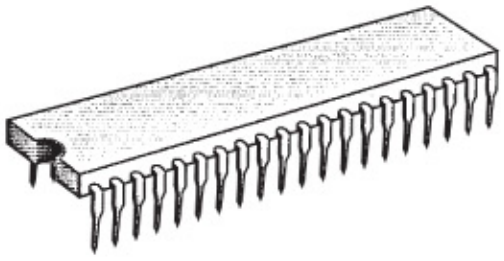
Компанія Intel почала продавати мікросхему 8080 за ціною 360 доларів США, ніби кепкуючи над мейнфреймами System/360 компанії IBM, на які великі корпорації витрачали мільйони доларів. Не можна сказати, що мікросхема 8080 могла тягатися з мейнфреймом System/360, однак у наступні кілька років IBM не могла обійти увагою ці мініатюрні комп'ютери.

Чіп 8080 – це 8-розрядний мікропроцесор, який містить близько 6000 транзисторів, працює з тактовою частотою два мегагерці і може адресувати 64 кілобайти пам'яті. Мікросхема 6800 містить близько 4000 транзисторів і також може адресувати 64 кілобайти пам'яті. Перші чіпи 6800 працювали на частоті один мегагерц, але до 1977 Motorola представила їх удосконалені версії, що працювали з частотою 1,5 і 2 мегагерца.

Мікросхеми 8080 та 6800 називаються *однокристальними* мікропроцесорами, або *однокристальними комп'ютерами.* Процесор – це лише один із компонентів комп'ютера. На додаток до нього потрібні принаймні деякий оперативний пристрій (ОЗУ), якийсь спосіб, що дозволяє користувачеві записати інформацію в комп'ютер (пристрій введення), а також витягти її з нього (пристрій виведення). Крім того, необхідно ще кілька інших мікросхем для поєднання всіх компонентів. Я опишу їх докладніше у розділі 21.

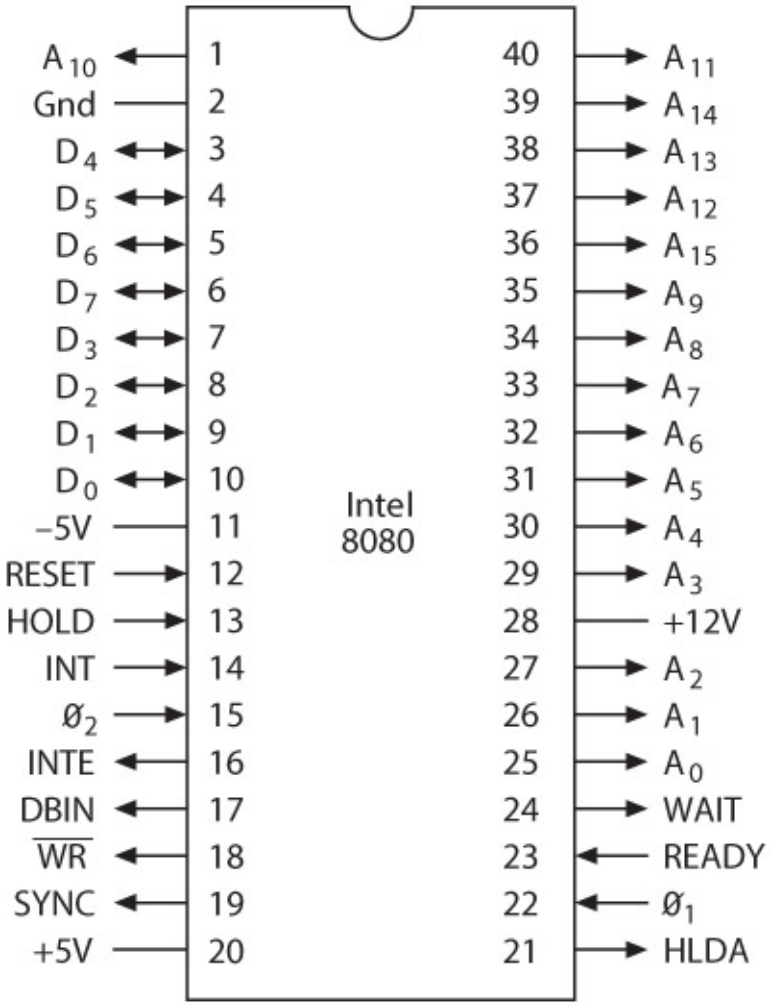
Нині ж розглянемо сам мікропроцесор. Часто його опис супроводжується блок-схемою, яка ілюструє внутрішні компоненти мікропроцесора та те, як вони пов'язані між собою. Однак на це ми вже надивилися у темі 17. Тут ми постараємося зрозуміти, що відбувається всередині процесора, спостерігаючи, як він взаємодіє із зовнішнім світом. Іншими словами, ми можемо уявити мікропроцесор як «чорну скриньку», внутрішню роботу якого нам не обов'язково досконально вивчати, щоб розібратися в його функціях. Для цього достатньо ознайомитись із вхідними та вихідними сигналами, зокрема набором команд.

Чіпи 8080 та 6800 – це інтегральні мікросхеми з 40 виводами. Найчастіше довжина їхнього корпусу становить п'ять сантиметрів, ширина – близько 1,5 сантиметра, а висота – близько трьох



міліметрів.

Зрозуміло, ми говоримо лише про корпус. Розмір кремнієвої пластини всередині набагато менший: у ранніх версіях 8-розрядних мікропроцесорів це квадрат зі стороною близько шести міліметрів. Корпус захищає кремнієвий чіп і забезпечує доступ до всіх його входів та виходів. На



наступній схемі показано функції 40 виводів мікросхеми 8080.

Кожному із створених нами електричних чи електронних пристроїв потрібне джерело живлення. Одна з особливостей мікросхеми 8080 – їй необхідні *три* різні напруги. На контакт 20 має подаватися напруга п'ять вольт, контакт 11 – мінус п'ять вольт, але в контакт 28-12 вольт. Контакт 2 підключається до землі. (У 1976 році Intel випустила мікросхему 8085 з більш простою системою електроживлення.)

Всі інші контакти зображені як стрілки. Стрілка, спрямована *від* мікросхеми, означає *вихідний* сигнал. Це контрольований мікропроцесором сигнал, на який реагують інші чіпи. Стрілка, спрямована *до* мікросхеми, – *вхідний* сигнал, що надходить від іншого чіпа. Деякі контакти використовуються як входи, так і виходи.

Для роботи процесора, описаного в розділі 17 необхідний осцилятор. Мікросхемі 8080 потрібні два різні сигнали тактової синхронізації з частотою два мегагерца, позначені на схемі символами 0_1 і 0_2 поряд з контактами 22 і 15. Ці сигнали зручніше генерувати за допомогою іншого чіпа компанії Intel під назвою генератор тактових імпульсів 8224. До цього чіпу підключається кристал кварцу з частотою 18 мегагерц, а решту робить генератор.

Мікропроцесор завжди передбачає кілька вихідних сигналів адресації пам'яті. Кількість сигналів, використовуваних для цієї мети, безпосередньо пов'язані з обсягом пам'яті, до якої може звертатися мікропроцесор. Чіп 8080 має 16 таких сигналів, позначених символами від A_0 до A_{15} що дозволяє йому адресувати $2^{16,}$ або 65 536 байт.

Мікропроцесор 8080 є 8-розрядним, отже він зчитує дані з пам'яті і записує їх по вісім біт за один раз. Для цього використовуються контакти з D_0 по D_7 які працюють як на вхід, так і на вихід. Коли мікропроцесор зчитує байт із пам'яті, ці контакти функціонують як входи; коли мікропроцесор записує байт на згадку – як виходи.

Ще десять контактів мікропроцесора призначені для *керуючих* сигналів. Наприклад, для скидання мікропроцесора використовується вхідний сигнал RESET. Вихідний сигнал WR змушує мікропроцесор записати байт на згадку. (Сигнал WR відповідає вхідному сигналу запису пам'яті W.) Крім того, іноді при зчитуванні мікропроцесором команд з пам'яті на контакти з D_0 по D_7 подаються інші керуючі сигнали. Для обробки цих додаткових сигналів комп'ютери, побудовані на основі мікропроцесора 8080, зазвичай використовують системний контролер 8228. Система керуючих сигналів мікропроцесора 8080 сумно відома своєю складністю, тому, якщо ви не маєте наміру збирати комп'ютер на основі цього чіпа, краще не мучте себе, намагаючись у ній розібратися.

Припустимо, що мікропроцесор 8080 підключений до пам'яті обсягом 64 кілобайти, з якою ми можемо обмінюватися даними незалежно від мікропроцесора.

Після скидання мікропроцесор 8080 зчитує байт, що зберігається в осередку пам'яті 0000h. Для цього на адресні контакти з A_0 по A_{15} подаються 16 нулів. Байт, який він зчитує, повинен відповідати команді процесора 8080, а процес його зчитування називається *вибіркою команди.*

У комп'ютері, який ми зібрали в темі 17, всі команди, за винятком HLT, займали в пам'яті по три байти, один байт містив код команди, а в інших зберігалася адреса. Команди процесора 8080 можуть займати один, два або три байти. Одні команди змушують процесор 8080 зчитати байт з певної комірки пам'яті, інші записати байт у певну комірку, треті виконати деякі внутрішні операції без використання оперативної пам'яті. Після обробки першої команди процесор 8080 звертається до другої команди, що зберігається в пам'яті, і т. д. Сукупність цих команд комп'ютерна програма, здатна виконувати цікаві функції.

Коли процесор 8080 працює з максимальною частотою два мегагерці, кожен тактовий цикл триває 500 наносекунд (1/2 000 000 циклів за секунду = 0,000000500 секунди). Команди комп'ютера, описаного в темі 17, виконували за чотири тактові цикли. На виконання команд для процесора 8080 потрібно від 4 до 18 тактових циклів, тобто від двох до дев'яти мікросекунд (мільйонних часток секунди).

Ймовірно, щоб зрозуміти, на що здатний конкретний мікропроцесор, потрібно вивчити набір його команд.

Підсумкова модель комп'ютера з теми 17 виконувала лише 12 команд. Набір 8-розрядного мікропроцесора може легко містити до 256 команд, причому код кожної команди відповідає конкретному 8-бітовому значенню. (Насправді команд може бути навіть більше, якщо деяким з них відповідатиме 2-байтний код.) Мікропроцесор 8080 не заходить так далеко, але його набір включає 244 коди. Ця кількість може бути досить великою, проте в цілому процесор 8080 не сильно перевершує комп'ютер з теми 17. Наприклад, якщо вам потрібно зробити операцію множення або поділу за допомогою процесора 8080, все одно доведеться написати для цього невелику програму.

Як ви пам'ятаєте з теми 17, кожен код набору команд процесора зазвичай відповідає певному мнемокоду, і деякі з них супроводжуються аргументами. Проте ці мнемокоди служать виключно зручності при зверненні до кодів команд. Процесор зчитує лише байти; він нічого не знає про відповідний їм текст. (Для ясності буду вільно звертатися з мнемокодами з документації до процесора Intel 8080.)

Набір комп'ютера з теми 17 включав дві важливі команди, які ми назвали «Завантажити» і «Зберегти». Кожна з цих команд займала три байти пам'яті. Перший байт команди «Завантажити» відповідав її коду, а дві наступні – 16-бітовій адресі. Процесор завантажував байт, що зберігається на цій адресі, в акумулятор. Аналогічно команда "Зберегти" консервувала вміст акумулятора за вказаною адресою.

Потім ми виявили, що коди цих двох команд можна скоротити, використовуючи мнемокоди.

```
LOD A,[aaaa]
STO [aaaa], A
```

Тут *A* – акумулятор (місце призначення для команди "Завантажити" і джерело для команди "Зберегти"), а фрагмент *aaaa* – 16-бітна адреса в пам'яті, яка зазвичай записується за допомогою чотирьох шістнадцяткових цифр.

Восьмирозрядний акумулятор у процесорі 8080 позначається літерою *A,* як і акумулятор у темі 17. Крім того, у наборі цього процесора є дві команди, які роблять те саме, що і команди «Завантажити» і «Зберегти» з теми 17. У процесорі 8080 цим командам відповідають коди 32h і

3Ah, за кожним з яких слідує 16-бітна адреса, а мнемокодами для них є STA (Store Accumulator – зберегти з акумулятора) та LDA (Load Accumulator – завантажити в акумулятор).

Код	Команда
32	STA [aaaa], A
3A	LDA A, [aaaa]

На додаток до акумулятора процесор 8080 має шість *регістрів,* які також можуть містити 8-бітові значення. Ці регістри схожі на акумулятор. Акумулятор вважається регістром особливого типу. Подібно до акумулятора, інші шість регістрів є засувками; процесор може переміщати байти з пам'яті регістри і з регістрів назад у пам'ять. На відміну від акумулятора, регістри не такі функціональні. Наприклад, при додаванні двох 8-бітних чисел результат завжди потрапляє в акумулятор, а не в один з регістрів.

Шість додаткових регістрів у процесорі 8080 називаються B, C, D, E, H і L. Насамперед люди запитують: «Що сталося з F і G?» – а потім запитують друге запитання: «А як щодо I, J і K?» Відповідь полягає в тому, що регістри H і L мають деякі особливості, а їх назва походить від слів high і low. Часто 8-бітні значення в регістрах H і L обробляються разом у вигляді 16-бітної *пари регістрів* HL, при цьому в регістрі H міститься старший (high) байт, а в регістрі L – молодший (low). Це 16-розрядне значення використовується для адресації пам'яті. Трохи згодом ми побачимо, як це працює.

Чи потрібні всі ці регістри? Чому ми їх не потребували, збираючи комп'ютер у розділі 17? Теоретично використовувати їх не обов'язково, а практично дуже зручно. Багато комп'ютерних програм здатні одночасно маніпулювати кількома числами. Це найпростіше робити, якщо числа зберігаються не в пам'яті, а в регістрах мікропроцесора. Крім того, програма працює швидше: що рідше вона звертається до пам'яті, то швидше виконується.

Для команди процесора 8080 під назвою MOV (Move – перемістити) передбачено 63 коди. Ці коди займають лише один байт і, як правило, переміщують вміст одного регістра в інший або той самий. Безліч команд MOV – наслідок використання в мікропроцесорі семи регістрів (включаючи акумулятор).

Ось перші 32 команди MOV. Пам'ятайте, що місце призначення відповідає аргументу ліворуч, а джерело аргументу праворуч.

Код	Команда	Код	Команда
40	MOV B, B	50	MOV D, B
41	MOV B, C	51	MOV D, C
42	MOV B, D	52	MOV D, D
43	MOV B, E	53	MOV D, E
44	MOV B, H	54	MOV D, H
45	MOV B, L	55	MOV D, L
46	MOV B,[HL]	56	MOV D,[HL]
47	MOV B, A	57	MOV D, A
48	MOV C, B	58	MOV E, B
49	MOV C, C	59	MOV E, C
4A	MOV C, D	5A	MOV E, D
4B	MOV C, E	5B	MOV E, E
4C	MOV C, H	5C	MOV E, H
4D	MOV C, L	5D	MOV E, L
4E	MOV C,[HL]	5E	MOV E,[HL]
4F	MOV C, A	5F	MOV E, A

Як бачите, дуже зручні команди. За наявності значення в одному з регістрів можна перемістити його до іншого. Зверніть увагу на чотири команди, які використовують пару регістрів HL, наприклад наступну.

```
MOV B,[HL]
```

Згадана вище команда LDA переміщує байт із пам'яті в акумулятор; 16-бітна адреса цього байта слідує безпосередньо за кодом команди LDA. Ця команда MOV переміщує байт із пам'яті в регістр B. Однак адреса байта, який має бути завантажений у регістр, зберігається в парі регістрів HL. Як 16-бітна адреса опинилась у парі регістрів HL? Це могло статися у різний спосіб. Можливо, ця адреса була якимось чином обчислена.

Загалом, обидві команди завантажують байт із пам'яті в мікропроцесор, але використовують два різні методи адресації пам'яті. Перший метод називається *прямою адресацією,* а другий – *індексною адресацією.*

```
LDA A,[aaaa]
MOV B,[HL]
```

Другий набір з 32 команд MOV показує, що осередки пам'яті, адресовані парою регістрів HL, можуть бути не тільки джерелом, а й місцем призначення.

Код	Команда	Код	Команда
40	MOV B, B	50	MOV D, B
60	MOV H, B	70	MOV [HL], B
61	MOV H, C	71	MOV [HL], C
62	MOV H, D	72	MOV [HL], D
63	MOV H, E	73	MOV [HL], E
64	MOV H, H	74	MOV [HL], H
65	MOV H, L	75	MOV [HL], L
66	MOV H, [HL]	76	HLT
67	MOV H, A	77	MOV [HL], A
68	MOV L, B	78	MOV A, B
69	MOV L, C	79	MOV A, C
6A	MOV L, D	7A	MOV A, D
6B	MOV L, E	7B	MOV A, E
6C	MOV L, H	7C	MOV A, H
6D	MOV L, L	7D	MOV A, L
6E	MOV L, [HL]	7E	MOV A, [HL]
6F	MOV L, A	7F	MOV A, A

Деякі з цих команд, наприклад, MOV A, A, не роблять нічого корисного. Команди MOV [HL],[HL] взагалі немає. Код, який міг би відповідати їй, виділено команді HLT (Halt – зупинити).

Більш показовий метод аналізу команд MOV – розгляд бітового шаблону їх коду. Код команди MOV складається з восьми бітів:

01dddsss,

де літери *ddd* відповідають 3-бітному коду місця призначення, а *sss* – 3-бітному коду джерела. Ці 3-бітові коди позначають такі регістри.

000 = регістр B
001 = регістр C
010 = регістр D
011 = регістр E
100 = регістр H
101 = регістр L
110 = осередок пам'яті за адресою HL
111 = акумулятор

Команда MOV L, E відповідає коду 01101011, або 6Bh. Ви можете звіритись з попередньою таблицею, щоб переконатися в цьому.

Ймовірно, десь усередині процесора 8080 три біти *sss* використовуються в селекторі «8 на 1», а три біти *ddd* управляють дешифратором «3 на 8», який визначає регістр, де буде зафіксовано значення.

Регістри B і C також можна використовувати як 16-бітну пару регістрів BC, а регістри D і E – як 16-бітну пару регістрів DE. Якщо будь-яка з цих пар регістрів містить адресу осередку пам'яті, звідки ви хочете рахувати або куди хочете записати байт, можете використовувати наступні

Код	Команда	Код	Команда
02	STAX [BC], A	0A	LDAX A, [BC]
12	STAX [DE], A	1A	LDAX A, [DE]

команди.

Інший тип команди Move називається Move Immediate («Перемістити безпосередньо») і позначається мнемокодом MVI. Ця команда складається із двох байтів. Перший – код команди, другий – байт даних. Цей байт переміщається з пам'яті в один із регістрів або в комірку пам'яті,

Код	Команда
06	MVI B, xx
0E	MVI C, xx
16	MVI D, xx
1E	MVI E, xx
26	MVI H, xx
2E	MVI L, xx
36	MVI [HL], xx
3E	MVI A, xx

адреса якої міститься в парі регістрів HL.

Наприклад, після виконання команди MVI E,37h у регістрі E буде утримуватися байт 37h. Цей третій метод звернення до пам'яті називається *безпосередньою адресацією.*

Набір із 32 кодів команд виконує чотири основні арифметичні операції, з якими ми познайомилися, коли збирали процесор (тема 17). До них відносяться додавання (ADD), додавання з перенесенням (ADC), віднімання (SUB) і віднімання із запозиченням (SBB). В усіх випадках акумулятор є одним із двох операндів, а також місцем призначення для результату.

Код	Команда	Код	Команда
80	ADD A, B	90	SUB A, B
81	ADD A, C	91	SUB A, C
82	ADD A, D	92	SUB A, D
83	ADD A, E	93	SUB A, E
84	ADD A, H	94	SUB A, H
85	ADD A, L	95	SUB A, L
86	ADD A, [HL]	96	SUB A, [HL]
87	ADD A, A	97	SUB A, A
88	ADC A, B	98	SBB A, B
89	ADC A, C	99	SBB A, C
8A	ADC A, D	9A	SBB A, D
8B	ADC A, E	9B	SBB A, E
8C	ADC A, H	9C	SBB A, H
8D	ADC A, L	9D	SBB A, L
8E	ADC A, [HL]	9E	SBB A, [HL]
8F	ADC A, A	9F	SBB A, A

Припустимо, що в акумуляторі A міститься байт 35h, а регістрі B – байт 22h. Після виконання команди SUB A, B в акумуляторі буде утримуватися байт 13h.

Якщо в A міститься байт 35h, в регістрі H – 10h, у регістрі L – 7Ch, в комірці пам'яті 107Ch – 4Ah, то, при виконанні команди ADD A,[HL] байт в акумуляторі (35h) додається до байта в комірці, до якої обертається пара регістрів HL (4Ah), а результат (7Fh) зберігається в акумуляторі.

Команди ADC і SBB дозволяють процесору 8080 додавати та віднімати 16-, 24-, 32-бітові числа, а також числа більшої розрядності. Припустимо, що пари регістрів BC та DE містять 16-бітні числа. Ви хочете додати їх та помістити результат у пару регістрів BC. Це можна зробити так.

```
MOV A, C; молодший байт
```

```
ADD A, E
MOV C, A
MOV A, B; старший байт
ADC A, D
MOV B, A
```

Для додавання використовуються дві команди: ADD – для молодшого байта, ADC – для старшого. Будь-який біт перенесення, що виникає в результаті першого додавання, бере участь у другому додаванні. Оскільки додати можна лише до значення в акумуляторі, в цьому невеликому фрагменті коду команда MOV використовується щонайменше чотири рази. Команди MOV часто зустрічаються в програмному коді для процесора 8080.

Настав час поговорити про прапори мікросхеми 8080. У процесорі з теми 17 використовувалися прапор переносу і прапор нуля. Мікросхема 8080 передбачає ще три прапори: знаку, парності та допоміжного перенесення. Усі прапори зберігаються у 8-бітному регістрі, який називається *словом стану програми* (Program Status Word, PSW). Такі команди як LDA, STA або MOV не впливають на ці прапори. Однак команди ADD, SUB, ADC та SBB змінюють прапори таким чином:

• прапор знаку встановлюється 1, якщо старший біт результату дорівнює 1, тобто якщо результат негативний;

• прапор нуля встановлюється 1, якщо результат дорівнює 0;

• прапор парності встановлюється в 1, якщо результат *парний,* тобто виражений у двійковому форматі результат містить парну кількість 1; прапор парності встановлюється в 0, якщо результат *непарний;* прапор парності іноді використовується для грубої перевірки результату наявності помилок; при написанні програм для 8080 процесора цей прапор використовується рідко;

• прапор переносу встановлюється в 1, якщо в результаті виконання команди ADD або ADC виникає біт переносу або в результаті виконання команд SUB і SBB біт переносу не виникає (така реалізація прапора переносу відрізняється від того, як він був реалізований на комп'ютері з теми 17 ;

• прапор допоміжного перенесення встановлюється в 1, якщо в результаті виконання команди виникає перенесення з молодшої тетради до старшої; цей прапор використовується лише для команди DAA (Decimal Adjust Accumulator – десяткова корекція акумулятора).

На прапор перенесення безпосередньо впливають дві команди.

Opcode	Instruction	Meaning
37	STC	Set Carry flag to 1
3F	CMC	Complement Carry flag

На відміну від комп'ютера з теми 17, який теж виконував команди ADD, ADC, SUB і SBB (хоча і не з таким же ступенем гнучкості), процесор 8080 здатний ще й на булеві операції І, АБО і виключає АБО. За виконання арифметичних та логічних операцій відповідає арифметико-логічний пристрій процесора.

Код	Команда	Код	Команда
A0	AND A, B	B0	OR A, B
A1	AND A, C	B1	OR A, C
A2	AND A, D	B2	OR A, D
A3	AND A, E	B3	OR A, E
A4	AND A, H	B4	OR A, H
A5	AND A, L	B5	OR A, L
A6	AND A, [HL]	B6	OR A, [HL]
A7	AND A, A	B7	OR A, A
A8	XOR A, B	B8	CMP A, B
A9	XOR A, C	B9	CMP A, C
AA	XOR A, D	BA	CMP A, D
AB	XOR A, E	BB	CMP A, E
AC	XOR A, H	BC	CMP A, H
AD	XOR A, L	BD	CMP A, L
AE	XOR A, [HL]	BE	CMP A, [HL]
AF	XOR A, A	BF	CMP A, A

Команди AND, OR і XOR виконуються *побітово,* тобто окремо над кожною парою бітів. Наприклад, в результаті виконання наступних команд значення в акумуляторі дорівнюватиме 05h.

MVI A,0Fh
MVI B,55h
AND A, B

Якби останньою була команда OR, то результат дорівнював би 5Fh; якби останньою була команда XOR – 5Ah.

Команда CMP (Compare – порівняти) аналогічна команді SUB, за винятком того, що результат не зберігається в акумуляторі. Іншими словами, команда CMP виконує віднімання, а потім видаляє результат. У чому сенс? У прапорах! Прапори говорять про те, як два порівнювані байти співвідносяться. Розглянемо, наприклад, такі команди.

MVI B,25h
CMP A, B

Після виконання вміст акумулятора (A) залишається незмінним. Якщо значення A дорівнює 25h, буде встановлено прапор нуля, а якщо значення A менше 25h – прапор переносу.

Для восьми арифметичних та логічних операцій також є версії, які виконуються безпосередньо над байтами.

Код	Команда	Код	Команда
C6	ADI A, xx	E6	ANI A, xx
CE	ACI A, xx	EE	XRI A, xx
D6	SUI A, xx	F6	ORI A, xx
DE	SBI A, xx	FE	CPI A, xx

Наприклад, два наведені вище рядки можна замінити наступним.

CPI A,25h

Ось ще дві команди для процесора 8080.

Код	Команда
27	DAA
2F	CMA

Команда CMA (Complement Accumulator – доповнити акумулятор) виконує доповнення значення в акумуляторі до 1. Кожен 0 звертається до 1, а 1 – до 0. Якщо в акумуляторі міститься значення 01100101, то після виконання команди CMA у ньому буде міститися значення 10011010. Цього ж результату можна досягти за допомогою наступної команди.

```
XRI A, FFh
```

Згадана вище команда DAA (Decimal Adjust Accumulator – десяткова корекція акумулятора), ймовірно, є найскладнішою в наборі команд процесора 8080. Спеціально для неї в мікропроцесорі передбачено невеликий пристрій.

DAA допомагає програмісту виконувати арифметичні операції над десятковими числами в кодуванні BCD (binary-coded decimal – десяткове у двійковому кодуванні), де кожна тетрада може набувати значення лише в діапазоні від 0000 до 1001, тобто від 0 до 9 у десятковому вираженні. У форматі BCD у восьми бітах байта можуть зберігатися дві десяткові цифри.

Припустимо, що акумулятор міститься BCD-значення 27h, яке фактично відповідає десятковому значенню 27, а в регістрі B міститься BCD-значення 94h. (Зазвичай шістнадцяткове значення 27h еквівалентно десятковому значенню 39.) В результаті виконання наступних команд в акумуляторі буде міститися значення BBh, яке, зрозуміло, не є BCD-значенням, оскільки кодування BCD-значення тетради не може перевищувати 9.

```
MVI A,27h
MVI B,94h
ADD A, B
```

Однак при виконанні команди DAA в акумулятор поміщається значення 21h і встановлюється прапор переносу, оскільки сума десяткових чисел 27 і 94 дорівнює 121. Ця команда може стати в нагоді для арифметичних операцій над числами кодування BCD.

Часто виникає необхідність у додаванні 1 до значення або у відніманні 1 від значення. У програмі для виконання множення, описаної в темі 17, потрібно відняти значення 1, і ми робили це, додаючи значення FFh, яке є доповненням до 2 числа -1. Процесор 8080 передбачає спеціальні команди для збільшення на 1 (інкрементування) та зменшення на 1 (декрементування) значення в регістрі або комірці пам'яті.

Код	Команда	Код	Команда
04	INR B	05	DCR B
0C	INR C	0D	DCR C
14	INR D	15	DCR D
1C	INR E	1D	DCR E
24	INR H	25	DCR H
2C	INR L	2D	DCR L
34	INR [HL]	35	DCR [HL]
3C	INR A	3D	DCR A

Однобайтові команди INR та DCR впливають на всі прапори, крім прапора перенесення.

Набір команд процесора 8080 включає чотири команди *циклічного зсуву,* які зрушують вміст акумулятора на один біт вліво або вправо.

Opcode	Instruction	Meaning
07	RLC	Rotate accumulator left
0F	RRC	Rotate accumulator right
17	RAL	Rotate accumulator left through carry
1F	RAR	Rotate accumulator right through carry

Ці команди впливають лише на прапор перенесення.

Припустимо, що акумулятор містить значення A7h, або 10100111 у двійковому форматі. Команда RLC зсуває біти вліво. Старший біт, який виштовхується за ліву межу розрядної сітки, стає молодшим, а також визначає стан прапора перенесення. В результаті виходить значення 01001111, а прапор переносу встановлюється в 1. Команда RRC так само зсуває біти вправо. Після виконання команди RRC значення 10100111 перетворюється на 11010011, а прапор перенесення знову встановлюється на 1.

Команди RAL та RAR працюють дещо інакше. При виконанні команди RAL вміст акумулятора зсувається вліво, старший біт зберігається у прапорі переносу, а в молодший біт записується попереднє значення прапора переносу. Наприклад, якщо акумулятор містить значення 10100111, а прапор переносу дорівнює 0, то після виконання команди RAL вміст акумулятора змінюється на 01001110, а у прапор переносу записується 1. При тих же початкових умовах після виконання команди RAR значення акумулятора змінюється на 01010011, а у прапорі перенесення зберігається значення 1.

Команди зсуву зручні при множенні числа на 2 (зсув ліворуч) та при діленні числа на 2 (зсув вправо).

Пам'ять, до якої звертається мікропроцесор, називається пам'яттю з *довільним доступом* тому, що мікропроцесор може отримати доступ до будь-якої конкретної комірки, просто надавши її адресу. Пам'ять RAM швидше нагадує книгу, яку можна відкрити на будь-якій сторінці, ніж тижневу підшивку газет на мікрофільмі. Щоб знайти потрібну інформацію в суботньому випуску, ми повинні переглянути *більшу* частину газет. Так, для відтворення останньої пісні на касеті ми маємо практично повністю перемотати одну із її сторін. Мікрофільм і магнітна стрічка відносяться до пристроїв не з довільним, а з *послідовним доступом.*

Пам'ять з довільним доступом, безумовно, хороша, особливо для мікропроцесорів, але іноді зручніше використовувати пристрій, доступ до якого здійснюється мимовільно і непослідовно. Допустимо, ви працюєте в офісі, і співробітники підходять до вашого столу, щоб дати завдання. Виконання кожного з них передбачає використання папки з документами. Часто під час роботи над одним завданням ви виявляєте, що ви можете продовжувати, доки виконайте певне завдання, використовуючи іншу папку. Так що поверх першої папки ви кладете другу та працюєте з нею. Потім вам дають ще одне завдання, пріоритетніше, ніж попереднє, і ви кладете нову папку поверх двох інших. Для цього вам потрібна ще одна папка з документами. І ось на вашому столі вже цілий стос із чотирьох папок.

Це впорядкований спосіб зберігання та відстеження всіх завдань. Найвища папка завжди відповідає пріоритетному завданню. Після закінчення роботи з цією папкою ви переходите до наступної. Коли ви нарешті розберетеся з останньою папкою на своєму столі (з тією, з якою почали), зможете вирушити додому.

Технічно така форма зберігання даних називається "*стек".* Будується він знизу нагору, а розбирається зверху донизу. Елементи стека організовані за принципом "останнім увійшов – першим вийшов" (Last In First Out, LIFO). Останній елемент, поміщений у стек, видаляється першим. Перший доданий у стек елемент буде видалено останнім.

Комп'ютери можуть використовувати стек, але не для зберігання завдань, а для зберігання чисел, що зручно. Додавання елемента в стек називається вштовхуванням (push), а видалення – виштовхуванням (pop).

Припустимо, ви пишете програму мовою асемблера, у якій використовуються регістри A, B і C. На якомусь етапі програмі потрібно виконати ще один невеликий розрахунок, що також

передбачає застосування регістрів A, B і C. У результаті потрібно повернутися до того, що ви робили раніше, і продовжити використовувати регістри A, B і C з тими значеннями, які зберігалися.

Безперечно, ви можете просто зберегти значення регістрів A, B і C в інших осередках пам'яті, а потім завантажити їх звідти назад. Однак тоді потрібно буде стежити за вмістом осередків пам'яті. Більш зручний спосіб – переміщення (вштовхування) значень регістрів у стек.

```
PUSH A
PUSH B
PUSH C
```

Я поясню, як працюють ці команди трохи пізніше. Поки що досить зрозуміти, що вони якимось чином зберігають вміст регістрів у пам'яті LIFO. Після виконання цих команд ваша програма може спокійно використовувати ці регістри для інших цілей. Щоб повернути попередні значення, ви просто виштовхуєте елементи зі стека у зворотному порядку.

```
POP C
POP B
POP A
```

Пам'ятайте: останній поміщений у стек елемент видаляється з нього першим. Випадкова зміна порядку команд POP призведе до помилки.

Перевага стека в тому, що його можуть використовувати різні розділи програми без проблем. Наприклад, після поміщення в стек значень регістрів A, B і C іншому розділі програми може знадобитися зробити те саме з регістрами C, D і E.

```
PUSH C
PUSH D
PUSH E
```

Для відновлення значення регістрів використовуються команди POP.

```
POP E
POP D
POP C
```

Після їх виконання зі стека будуть вилучені значення регістрів C, B та A.

Як реалізується стек? Перш за все це просто розділ пам'яті, який не використовується для зберігання будь-яких інших даних. Для звернення до цього розділу пам'яті мікропроцесор 8080 передбачає спеціальний 16-бітовий регістр, який називається *покажчиком стека* (Stack Pointer, SP).

Наведені вище приклади додавання та видалення елементів зі стека не зовсім точно демонструють роботу мікропроцесора 8080. Команда 8080 PUSH фактично зберігає в стеку 16-бітові значення, а команда POP витягує їх. Саме тому замість таких команд, як PUSH C та POP C,

Код	Команда	Код	Команда
C5	PUSH BC	C1	POP BC
D5	PUSH DE	D1	POP DE
E5	PUSH HL	E1	POP HL
F5	PUSH PSW	F1	POP PSW

використовуємо наступні вісім.

Команда PUSH BC зберігає у стеку значення регістрів B і C, а команда POP BC витягує їх. Абревіатура PSW в останньому рядку означає *слова стану програми,* які, як ви пам'ятаєте, є 8-бітовим регістром, що містить прапори. Дві команди в нижньому рядку фактично поміщають та витягують із стека вміст як акумулятора, так і регістра PSW. Якщо ви хочете зберегти вміст *усіх* регістрів та значення *всіх* прапорів, використовуйте такі команди.

```
PUSH PSW
PUSH BC
PUSH DE
PUSH HL
```

Якщо вам потрібно відновити вміст цих регістрів, зверніться до команд POP у зворотному порядку.

```
POP HL
POP DE
POP BC
POP PSW
```

Як працює стек? Припустимо, що покажчик стека дорівнює 8000h. Під час виконання команди PUSH BC відбувається таке:

• значення покажчика стека зменшується на 1 і стає рівним 7FFFh;

• вміст регістру B зберігається за адресою, що відповідає значенню покажчика стека, тобто в комірці 7FFFh;

• значення покажчика стека зменшується на 1 і стає рівним 7FFEh;

• вміст регістру C зберігається за адресою, що відповідає значенню покажчика стека, тобто в комірці 7FFEh.

Команда POP BC, що виконується при значенні покажчика стека, все ще рівному 7FFEh, здійснює зворотні операції:

• вміст регістру C завантажується з комірки, адреса якої відповідає значенню покажчика стека, тобто з комірки 7FFEh;

• значення покажчика стека збільшується на 1 і стає рівним 7FFFh;

• вміст регістру B завантажується з комірки, адреса якої відповідає значенню покажчика стека, тобто з комірки 7FFFh;

• значення вказівника стека збільшується на 1 і дорівнюватиме 8000h.

Кожна команда PUSH збільшує розмір стека на два байти. Існує ймовірність того, що через помилку в програмі розмір стека стане настільки великим, що його вміст почне зберігатися в осередках, зайнятих необхідним програмним кодом або даними. Ця проблема називається *переповненням стека.* Так само занадто багато команд POP може призвести до передчасного *вичерпання стека.*

Якщо до процесора 8080 підключена пам'ять об'ємом 64 кілобайт, є сенс встановити початкове значення вказівника стека рівним 0000h. Перша команда PUSH зменшує це значення на 1 – до FFFFh. Після цього стек займе область пам'яті з найвищими адресами, яка максимально віддалена від ваших програм, що зберігаються, ймовірно, починаючи з 0000h.

Встановити значення вказівника стека можна за допомогою команди LXI (Load Extended Immediate – розширене безпосереднє завантаження). Перелічені далі команди також

Код	Команда
01	LXI BC, xxxx
11	LXI DE, xxxx
21	LXI HL, xxxx
31	LXI SP, xxxx

завантажують у 16-бітові пари регістрів два байти, які йдуть за кодом команди.

Команда LXI BC, 527 Ah еквівалентна наступним командам.

```
MVI B,52
MVI C,7Ah
```

Однак команда LXI дозволяє заощадити один байт. Крім того, остання команда LXI в попередній таблиці використовується для встановлення конкретного значення покажчика стека. Часто ця команда однією з перших виконується процесором після його перезапуску.

```
0000h: LXI SP,0000h
```

Збільшити та зменшити на 1 значення пари регістрів та покажчика стека можна за допомогою наступних команд.

Код	Команда	Код	Команда
03	INX BC	0B	DCX BC
13	INX DE	1B	DCX DE
23	INX HL	2B	DCX HL
33	INX SP	3B	DCX SP

Розглянемо ще кілька 16-бітових команд. Наступні команди додають вміст 16-бітових пар регістрів із вмістом пари регістрів HL.

Код	Команда
09	DAD HL, BC
19	DAD HL, DE
29	DAD HL, HL
39	DAD HL, SP

Ці команди дозволяють заощадити кілька байтів. Наприклад, перша з них, зазвичай, вимагає шість байт.

```
MOV A, L
ADD A, C
MOV L, A
MOV A, H
ADC A, B
MOV H, A
```

Команда DAD зазвичай використовується для обчислення адрес осередків пам'яті і впливає тільки на прапор переносу.

Наступні два коди команд супроводжуються 2-байтовою адресою комірки пам'яті і дозволяють зберегти вміст пари регістрів HL у відповідній комірці, а також завантажити вміст у пару регістрів HL.

Opcode	Instruction	Meaning
2h	SHLD [aaaa],HL	Store HL Direct
2Ah	LHLD HL,[aaaa]	Load HL Direct

Вміст регістру L зберігається за адресою *aaaa,* а вміст регістру H – за адресою *aaaa* + 1.

Ці дві команди завантажують у лічильник команд (PC) або в покажчик стека (SP) значення пари регістрів HL.

Opcode	Instruction	Meaning
E9h	PCHL PC,HL	Load Program Counter from HL
F9h	SPHL SP,HL	Load Stack Pointer from HL

Команда PCHL – своєрідна команда переходу. Після неї процесор 8080 виконує команду, код якої займає комірку за адресою, записаною в парі регістрів HL. Команда SPHL – ще один спосіб встановлення значення покажчика стека.

Наступні дві команди дозволяють поміняти місцями вміст регістрів HL з двома байтами, що є верхніми елементами стека, або з вмістом пари регістрів DE.

Opcode	Instruction	Meaning
E3h	XTHL HL,[SP]	Exchange top of stack with HL
EBh	XCHG HL,DE	Exchange DE and HL

З усіх команд переходу для процесора 8080 поки що я описав лише PCHL. Як ви пам'ятаєте з, процесор передбачає регістр під назвою «лічильник команд», що містить адресу комірки пам'яті, з якої процесор витягує наступну команду, що підлягає виконанню. Як правило, лічильник команд змушує процесор виконувати команди, збережені у пам'яті, послідовно. Проте звані команди *переходу,* чи *розгалуження,* дозволяють процесору відхилитися від цього головного курсу. Ці команди завантажують у лічильник команд інше значення, тому наступна команда витягується процесором з іншої області пам'яті.

Незважаючи на зручність звичайної команди *переходу* команди *умовного переходу* зручніші, оскільки змушують процесор переходити до іншої адреси, ґрунтуючись на значенні певного прапора, наприклад прапора переносу або прапора нуля. Саме реалізація умовного переходу перетворила автоматизований суматор із теми 17 на універсальний цифровий комп'ютер.

У процесорі 8080 є п'ять прапорів, чотири з яких використовуються для реалізації умовних переходів. Набір команд 8080 містить дев'ять команд безумовних та умовних переходів, що залежать від того, чому рівні прапори нуля, перенесення, парності та знаку: 1 або 0.

Перш ніж продемонструвати ці команди, хочу познайомити вас з двома іншими типами команд, які стосуються переходу. Перша – CALL (виклик), вона аналогічна команді переходу, за винятком того, що перед завантаженням нової адреси в лічильник команд процесор зберігає попередню адресу. Де він зберігає цю адресу? Зрозуміло, у стеку!

Ця стратегія передбачає, що команда виклику зберігає інформацію про те місце, *звідки було здійснено перехід.* Збережена адреса дозволяє процесору повернутися до вихідного розташування. Команда для здійснення зворотного переходу називається RET (Return – повернутися). Вона видаляє зі стека 2-байтне значення і завантажує його в лічильник команд.

Команди CALL і RET – надзвичайно важливі функції будь-якого процесора, що дозволяють програмісту реалізовувати підпрограми, які часто використовуються фрагментами коду. (Під словом «*часто*» зазвичай я маю на увазі «більше разу».) Підпрограми – основні організуючі елементи програм мовою асемблера.

Звернемося до приклада. У процесі написання програми мовою асемблера у вас виникає потреба у перемноженні двох байтів. Ви пишете код для виконання цієї операції, а потім продовжуєте роботу з програмою. На якомусь етапі вам знову потрібно перемножити два байти. Оскільки ви вже знаєте, як це зробити, можна просто використовувати ті ж команди знову і знову. Чи збираєтеся ви вдруге ввести ці команди на згадку? Сподіваюся, що ні, оскільки це марна трата часу та пам'яті. Натомість вам слід просто перейти до попереднього фрагмента коду. Щоправда, у такому разі звичайна команда переходу не спрацює, оскільки вона дозволяє повернутися до місця, з якого було здійснено перехід. Саме в цьому випадку знадобляться команди CALL та RET.

Набір команд, що дозволяють перемножити два байти, ідеально підходить на роль підпрограми. Давайте розглянемо одну з них. У темі 17 байти, що підлягають перемноженню, і добуток зберігалися в певних осередках пам'яті. Наведена далі підпрограма 8080 множить байт у регістрі B на байт у регістрі C і поміщає 16-бітний добуток у регістр HL.

```
Multiply:   PUSH PSW        ; Save registers being altered
            PUSH BC

            SUB H,H         ; Set HL (result) to 0000h
            SUB L,L

            MOV A,B         ; The multiplier goes in A
            CPI A,00h       ; If it's 0, we're finished.
            JZ AllDone

            MVI B,00h       ; Set high byte of BC to 0

MultLoop:   DAD HL,BC       ; Add BC to HL
            DEC A           ; Decrement multiplier
            JNZ MultLoop    ; Loop if it's not 0

AllDone:    POP BC          ; Restore saved registers
            POP PSW
            RET             ; Return
```

Зверніть увагу: перший рядок підпрограми починається з мітки Multiply. Ця позначка відповідає адресі комірки пам'яті, в якій розташована підпрограма. Підпрограма починається із двох команд PUSH. Як правило, вона намагається зберегти (а надалі відновити) значення будь-яких регістрів, які можуть знадобитися.

Потім підпрограма записує значення 0 в регістри H і L. Для цього замість команди SUB можна було б використовувати команду MVI (Move Immediate – перемістити безпосередньо), проте в цьому випадку знадобилися чотири команди, а не дві. Після виконання підпрограми в парі регістрів HL утримуватиметься добуток.

Після цього підпрограма переміщує вміст регістру B (множник) в A і перевіряє, чи не дорівнює він 0. Якщо він дорівнює 0, підпрограма завершується, оскільки добуток – 0. Оскільки значення в регістрах H і L вже дорівнюють 0, підпрограма може просто використовувати команду JZ (Jump If Zero – перейти, якщо нуль), щоб перейти до двох команд POP наприкінці програми.

В іншому випадку підпрограма записує в регістр B значення 0. Тепер у парі регістрів BC міститься 16-бітове множине, а в акумуляторі (A) – множник. Команда DAD додає значення BC (множине) до значення HL (добуток). Значення множника A зменшується на 1. Поки він не буде дорівнює 0, виконання команди JNZ (Jump If Not Zero – перейти, якщо не нуль) буде призводити до повторного додавання значення BC зі значенням HL. Цей невеликий цикл буде виконуватися доти, доки кількість операцій додавання BC і HL не стане рівним множнику. (Ефективнішу підпрограму для множення можна написати, використовуючи команди зсуву з набору команд процесора 8080.)

Цю підпрограму для перемноження чисел, наприклад 25h та 12h, можна використовувати у програмі, додавши наступний фрагмент коду.

```
MVI B,25h
MVI C,12h
CALL Multiply
```

Команда CALL зберігає в стеку значення лічильника команд, яке представляє адресу команди, наступної після CALL. Потім CALL викликає перехід до команди, яку вказує мітка Multiply. Це початок підпрограми. Після того, як підпрограма розрахує добуток, вона виконає команду RET, в результаті чого в лічильник команд буде повернуто значення зі стека. Потім буде виконано команду, наступну після CALL.

Набір команд процесора 8080 передбачає умовні команди *виклику* та *повернення,* проте вони використовуються рідше, ніж звичайні команди *переходу.* Усі вони перераховані у таблиці.

Condition	Opcode	Instruction	Opcode	Instruction	Opcode	Instruction
None	C9	RET	C3	JMP aaaa	CD	CALL aaaa
Z not set	C0	RNZ	C2	JNZ aaaa	C4	CNZ aaaa
Z set	C8	RZ	CA	JZ aaaa	CC	CZ aaaa
C not set	D0	RNC	D2	JNC aaaa	D4	CNC aaaa
C set	D8	RC	DA	JC aaaa	DC	CC aaaa
Odd parity	E0	RPO	E2	JPO aaaa	E4	CPO aaaa
Even parity	E8	RPE	EA	JPE aaaa	EC	CPE aaaa
S not set	F0	RP	F2	JP aaaa	F4	CP aaaa
S set	F8	RM	FA	JM aaaa	FC	CM aaaa

Як ви знаєте, до мікропроцесора підключається не лише пам'ять. Комп'ютерна система зазвичай потребує пристрої введення та виведення (I/O), які полегшують користувачам взаємодію з машиною. До цих пристроїв, як правило, належать клавіатура та дисплей.

Як мікропроцесор взаємодіє із цими *периферійними* пристроями? (Всі підключені до мікропроцесора компоненти, крім пам'яті, називаються периферійними.) Конструкція периферійних пристроїв, подібно до пам'яті, передбачає інтерфейс. Мікропроцесор може записувати та зчитувати дані з периферійного пристрою, вказуючи певні адреси, на які він реагує. У деяких мікропроцесорах периферійні пристрої фактично використовують адреси, які зазвичай використовуються для звернення до пам'яті. Така конфігурація називається *введенням-виводом з розподілом пам'яті.* Проте в процесорі 8080, крім звичайних 65536 адрес для пристроїв введення і виведення, спеціально зарезервовані 256 додаткових, які називаються *портами вводу/виводу.* Адресні сигнали пристроїв введення/виводу подаються на входи з A_0 по A_7 а від звернень до пам'яті їх відрізняють сигнали, що фіксуються чіпом системного контролера 8228.

Команда OUT записує вміст акумулятора в порт, що адресується наступним за командою байтом. Команда IN дозволяє зчитати байт з акумулятора.

Код	Команда
D3	OUT pp
DB	IN pp

Периферійним пристроям іноді потрібно привернути увагу мікропроцесора. Наприклад, коли ви натискаєте клавішу на клавіатурі, бажано, щоб мікропроцесор дізнавався про це відразу. Це реалізується завдяки механізму *переривань* – сигналів, що надходять від периферійного пристрою на вхід процесора INT 8080.

Однак після перезапуску мікропроцесор 8080 не реагує на переривання. Для дозволу переривань програма має виконати команду EI (Enable Interrupts – дозволити переривання), а їх заборони – команду DI (Disable Interrupts – заборонити переривання).

Код	Команда
F3	DI
FB	EI

Вихідний сигнал процесора 8080 INTE означає, що переривання було дозволено. Коли периферійному пристрою виникає необхідність перервати роботу мікропроцесора, він подає на вхід INT сигнал, рівний 1. У відповідь на нього процесор 8080 витягає з пам'яті команду, проте керуючі сигнали повідомляють про переривання. У відповідь цей периферійний пристрій передає процесору 8080 одну з наступних команд.

Код	Команда	Код	Команда
C7	RST 0	E7	RST 4
CF	RST 1	EF	RST 5
D7	RST 2	F7	RST 6
DF	RST 3	FF	RST 7

Ці команди RST (Restart – перезавантаження) аналогічні командам CALL у тому, що поточне значення лічильника команд зберігається у стеку. Однак після цього RST здійснює перехід до певних адрес пам'яті: RST 0 переходить до комірки 0000h, RST 1 – до комірки 0008h і так далі аж до RST 7, яка здійснює перехід до комірки 0038h. У цих осередках повинні бути фрагменти коду, передбаченого для обробки переривання. Наприклад, переривання від клавіатури привело до виконання команди RST 4. Це означає, що починаючи з комірки 0020h у пам'яті повинен зберігатися деякий код для зчитування байта, введеного з клавіатури. (У розділі 21 я поясню все це.)

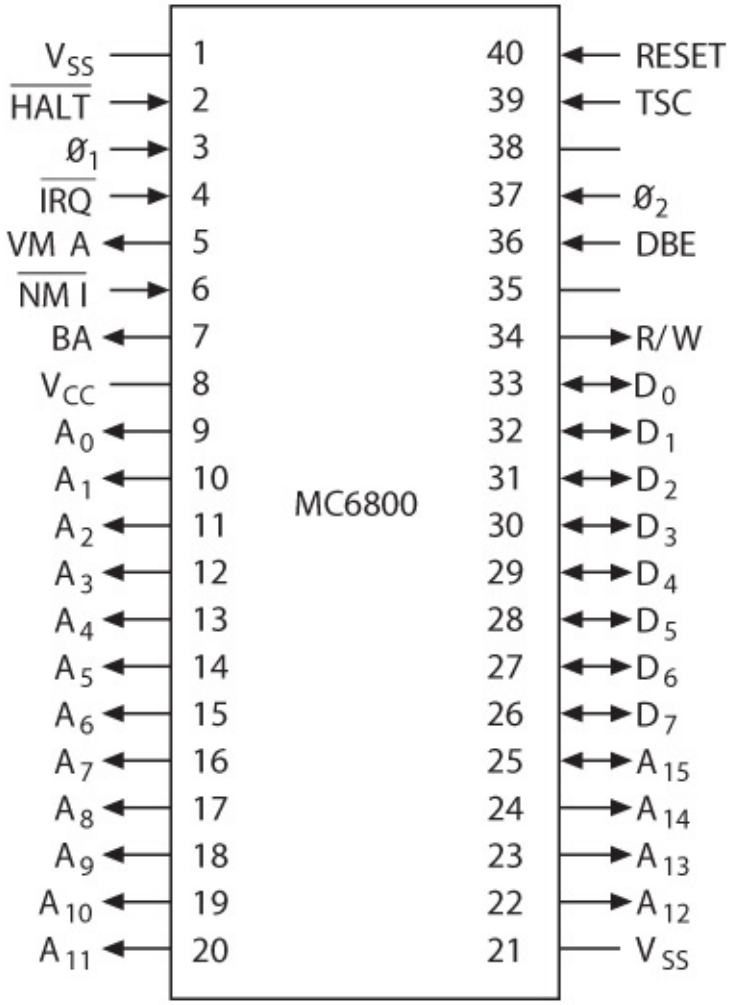
На даний момент мною описано 243 коди команд. Існує 12 байтів, які не відповідають жодним командам: 08h, 10h, 18h, 20h, 28h, 30h, 38h, CBh, D9h, DDh, EDh та FDh. У сумі це дає 255.

Код	Команда
00	NOP

Але є ще один код, про який я маю згадати.

Команда NOP (No Operation – немає операції) змушує процесор не діяти. Навіщо вона потрібна? Для заповнення адресного простору. Як правило, процесор 8080 може виконувати безліч команд NOP без будь-яких неприємних наслідків.

Не буду детально описувати будову мікросхеми 6800 компанії Motorola, оскільки її конструкція та функціонал багато в чому аналогічні відповідним аспектам процесора 8080. На наступній схемі зображено її 40 контактів.



Контакт V_{ss} підключається до землі, V_{cc} – до джерела живлення з напругою п'ять вольт. Як і процесор 8080, мікросхема 6800 передбачає 16 вихідних адресних сигналів та вісім сигналів для даних, що працюють як на введення, так і на виведення. Є сигнали RESET та R/W (читання/запис). На вхід IRQ подається сигнал *запиту на переривання* (interrupt request). Вважається, що система синхронізації в мікросхемі 6800 влаштована набагато простіше, ніж у процесорі 8080. Чого в мікросхемі 6800 немає, то це поняття портів введення/виводу. Адреси всіх пристроїв введення та виведення належать загальному адресному простору 6800.

Мікросхема 6800 передбачає 16-бітний лічильник команд, 16-бітовий покажчик стека, 8-бітовий регістр стану (для прапорів) і два 8-бітові акумулятори, звані A і B.

Як акумулятор, а не простого регістр розглядається B, оскільки його можна використовувати так само, як і A. Однак додаткових 8-бітних регістрів в такій мікросхемі немає.

Натомість 6800 передбачає 16-бітний *індексний регістр,* який може використовуватися для зберігання 16-бітної адреси, подібно до пари регістрів HL в 8080. Для багатьох команд адреса може обчислюватися шляхом підсумовування значення їх індексного регістра і байта, який слідує за кодом команди.

Незважаючи на те, що мікросхема 6800 виконує приблизно ті ж операції, що і процесор 8080 (завантаження, збереження, додавання, віднімання, зсув, перехід, виклик), коди команд і відповідні їм мнемокоди у цих чіпів зовсім різні. У наступній таблиці, наприклад, перераховані команди *переходу* набору мікросхеми 6800.

Opcode	Instruction	Meaning
20h	BRA	Branch
22h	BHI	Branch If Higher
23h	BLS	Branch If Lower or Same
24h	BCC	Branch If Carry Clear
25h	BCS	Branch If Carry Set
26h	BNE	Branch If Not Equal
27h	BEQ	Branch If Equal
28h	BVC	Branch If Overflow Clear
29h	BVS	Branch If Overflow Set
2Ah	BPL	Branch If Plus
2Bh	BMI	Branch If Minus
2Ch	BGE	Branch If Greater than or Equal to Zero
2Dh	BLT	Branch If Less than Zero
2Eh	BGT	Branch If Greater than Zero
2Fh	BLE	Branch If Less than or Equal to Zero

У мікросхемі 6800 не використовується прапор парності, проте, на відміну від 8080, в ній передбачений прапор переповнення. Деякі з перерахованих команд переходу залежить від *комбінацій* прапорів.

Зрозуміло, набори команд 8080 та 6800 різняться. Ці два процесори були спроектовані приблизно в той же час двома різними групами інженерів у двох різних компаніях. Ця несумісність означає, що жоден із цих процесорів не може виконати машинний код, написаний для іншого. Крім того, програма, написана мовою асемблера одного процесора, не може бути перетворена на коди команд іншого. Про написання комп'ютерних програм, які працюють на кількох процесорах, ми поговоримо у розділі 24.

Існує ще одна цікава різниця між 8080 і 6800: в обох мікропроцесорах команда LDA завантажує в акумулятор значення із зазначеної комірки пам'яті. У 8080, наприклад, наступна

послідовність байтів завантажує в акумулятор байт із комірки 347Bh.

Тепер порівняйте цю послідовність із командою LDA для процесора 6800, яка використовує так званий розширений режим адресації.

B6h	Команда 6800 LDA
7Bh	
34h	

Ця послідовність байтів завантажує в акумулятор А байт із комірки 7B34h.

Відмінність не відразу впадає у вічі. Звичайно, ви очікували, що коди команд будуть різними: 3Ah для 8080 і B6h для 6800. Однак ці два мікропроцесори також по-різному обробляють адресу, яка йде за кодом операції. Процесор 8080 передбачає, що першим повинен йти молодший байт, за яким слідує старший, а процесор 6800 – що першим повинен йти старший байт.

Це важливе різницю між мікросхемами Intel і Motorola у частині зберігання багатобайтних значень *не було подолано.* Мікропроцесори Intel досі продовжують зберігати багатобайтні значення, починаючи з молодшого байта, а мікропроцесори Motorola – починаючи зі старшого байта.

Ці два порядки відомі як від молодшого до старшого (little-endian, спосіб Intel) і від старшого до молодшого (big-endian, спосіб Motorola). Перш ніж сперечатися, який з методів краще, майте на увазі, що терміни Big-Endian («тупоконечник») і Little-Endian («гостроконечник») взяті з книги Джонатана Свіфта «Подорожі Гулівера» і пов'язані з війною між Ліліпутією та Блефуску, що розгорілася через розбіжності щодо того, з якого кінця слід розбивати варене яйце: з гострого або тупого. Ця суперечка не має сенсу. (З іншого боку, зізнаюся, що підхід, який я використовував у комп'ютері з теми 17, не здається мені кращим!) Хоча жоден із наведених методів не може вважатися правильним, різниця між ними створює додаткову проблему несумісності при обміні інформацією між системами, заснованими на цих принципах.

Що сталося з двома класичними мікропроцесорами? Процесор 8080 був покладений в основу того, що деякі люди називали першим персональним комп'ютером, хоча його правильніше було б назвати першим *домашнім* комп'ютером. Це був Альтаїр 8800 (Altair 8800), фотографія якого прикрасила обкладинку журналу Popular Electronics в січні 1975 року.

Якщо ви уважно розглянете цей комп'ютер, помітите на передній панелі вже знайомі індикатори та перемикачі. Це той самий примітивний пульт управління, який я запропонував для масиву RAM в темі 17.

Слідом за процесором 8080 були випущені мікросхеми Intel 8085 і Z-80 компанії Zilog, конкурента Intel, заснованого її колишнім співробітником Федеріко Фаджіном, який зробив істотний внесок у розробку мікросхеми 4004. Процесор Z-80 був повністю сумісний з 8080, але передбачав безліч додаткових команд. В 1977 чіп Z-80 був використаний в мікрокомп'ютері компанії Radio Shack TRS-80 Model 1.

Крім того, в 1977 році компанія Apple Computer Company, заснована Стівом Джобсом і Стівом Возняком, представила комп'ютер Apple II, в якому замість 8080 та 6800 використовувався малобюджетний чіп 6502 компанії MOS Technology – удосконалена версія мікросхеми 6800.

У червні 1978 року компанія Intel випустила 16-бітний процесор 8086, здатний адресувати один мегабайт пам'яті. Набір команд 8086 не був сумісний із процесором 8080, проте він передбачав команди для множення та поділу. Через рік Intel представила мікропроцесор 8088, внутрішньо ідентичний чіпу 8086, але з побайтною адресацією зовнішньої пам'яті, що дозволяла мікропроцесору використовувати поширені на той час 8-бітні допоміжні чіпи, розроблені для 8080. Компанія IBM застосовувала мікропроцесор 8089 в своєму персональному комп’ютері під назвою IBM PC.

Вихід IBM на ринок персональних комп'ютерів серйозно на нього вплинув, і багато компаній почали випускати пристрої, сумісні з IBM PC. (Тема сумісності буде розглянута в наступних розділах.) Протягом багатьох років вираз IBM PC-сумісний мав на увазі використання в пристрої мікросхеми Intel, зокрема мікропроцесора Intel сімейства x86. У 1982 році сімейство x86 поповнилося чіпами 186 і 286, у 1985 році – 32-бітним чіпом 386, у 1989-му – чіпом 486, а в 1993-му – мікропроцесорами Intel Pentium, які в даний час встановлюються в комп'ютери, PC. Незважаючи на те, що набори команд мікропроцесорів Intel постійно розширюються, вони продовжують підтримувати коди команд більш ранніх процесорів, починаючи з 8086.

У комп'ютері Apple Macintosh, вперше представленому в 1984 році, використовувався 16-бітний мікропроцесор Motorola 68000, який є прямим нащадком чіпа 6800. Процесор 68000 і його пізніші версії (часто об'єднуються в серію 68K) є одними з найпопулярніших мікропроцесорів.

Починаючи з 1994 року в комп'ютерах Macintosh встановлено мікропроцесор PowerPC, розроблений спільно компаніями Motorola, IBM та Apple. У основі лежить мікропроцесорна архітектура RISC (Reduced Instruction Set Computing – обчислення зі скороченим набором команд), у межах якої реалізується спроба збільшення швидкості процесора з допомогою спрощення деяких його аспектів. На комп'ютері з архітектурою RISC кожна команда, як правило, має однакову довжину (32 біта у випадку PowerPC), доступ до пам'яті обмежений лише командами для завантаження та збереження, а самі команди виконують швидше прості операції, ніж складні. Процесори на основі архітектури RISC зазвичай передбачають безліч регістрів, щоб якомога рідше звертатися до пам'яті.

Мікропроцесор PowerPC не може виконувати код, написаний для чіпів серії 68K, оскільки має зовсім інший набір команд. Проте мікропроцесори PowerPC, які зараз використовуються в комп'ютерах Apple Macintosh, можуть *емулювати* процесор 68K. Програма-емулятор, що працює на PowerPC, послідовно аналізує кожен код команди у програмі, написаній для чіпа серії 68K, та виконує відповідну дію. Це відбувається не так швидко, як виконання "рідного" коду PowerPC, але це працює.

Згідно із законом Мура, кількість транзисторів у мікропроцесорах має подвоюватись кожні 18 місяців. Навіщо потрібні ці численні додаткові транзистори?

Деякі транзистори дозволили збільшити розрядність процесора. Використання інших обумовлено появою нових команд. Велика *частина* сучасних мікропроцесорів передбачає команди для виконання операцій над числами з плаваючою точкою (про що я розповім у розділі 23). Крім того, в набір мікропроцесорів були додані нові команди для виконання деяких обчислень, що часто повторюються, необхідних для відображення на екранах комп'ютерів зображень або фільмів.

Для збільшення швидкодії у сучасних процесорах застосовується кілька методів. Один із них називається *конвеєризацією.* При виконанні однієї команди процесор зчитує наступну, причому

він до певної міри передбачає, як команди переходу можуть змінити програмний потік. Сучасні процесори також включають *кеш* (cache) – масив надшвидкісної оперативної пам'яті всередині процесора, в якому зберігаються нещодавно виконані команди. У комп'ютерних програмах часто використовуються невеликі цикли, а кеш дозволяє обійтися без повторного завантаження команд, що входять до нього. Всі ці функції, що підвищують швидкість процесора, вимагають додаткових логічних схем, отже, додаткових транзисторів.

Як я вже казав, мікропроцесор – це лише одна (нехай і найважливіша) частина комп'ютерної системи. Сконструюємо таку систему у розділі 21, але спочатку ми маємо навчитися записувати на згадку щось, крім кодів команд і чисел. Нам належить повернутися до першого класу і знову навчитися читати і писати текст.

Тема 13. Набір символів ASCII

У цифровій пам'яті комп'ютера зберігаються лише біти, тому все, з чим ми збираємося працювати, має бути у вигляді бітів. Ми вже бачили, як за допомогою бітів можна уявити числа та машинний код. Наступне завдання – представити текст. Зрештою, більшість накопиченої у світі інформації виражена у вигляді тексту, а наші бібліотеки сповнені книг, журналів та газет. Напевно, нам захочеться використовувати комп'ютери для зберігання звуків, зображень і фільмів, а з тексту набагато простіше почати.

Щоб представити текст у цифровій формі, ми маємо розробити деяку систему, де кожна буква відповідає унікальному коду. Для чисел і розділових знаків також потрібно передбачити коди. Коротше нам потрібні коди для всіх *буквено-цифрових* символів. Така система іноді називається *набором кодованих символів,* а окремі коди *кодами символів.*

Спочатку сформулюємо питання: Скільки бітів потрібно для цих кодів? Відповісти на нього непросто!

Коли ми думаємо про подання тексту за допомогою бітів, не забігатимемо далеко вперед. Ми звикли до красиво відформатованого тексту на сторінках книги, журналу чи газети. Абзаци складаються із рядків однакової ширини. Однак таке форматування істотно не впливає на текст. Коли ми читаємо коротку розповідь у журналі, а через роки зустрічаємо її в книзі, вона не здається нам іншою лише тому, що в книзі стовпець тексту ширший.

Іншими словами, забудьте, що текст форматований у вигляді плоских стовпців на друкованій сторінці. Вважайте його одномірним потоком букв, цифр, розділових знаків з додатковим символом, що позначає кінець одного абзацу і початок наступного.

Знову ж таки, якщо ви читаєте розповідь у журналі, а пізніше бачите її в книзі, і при цьому шрифт трохи відрізняється, чи це має якесь значення? Якщо журнальна версія починається так:

Називайте мене Ішмаель…, а книжна – так: Називайте мене Ішмаель…

Чи варто звертати на це увагу? Мабуть, ні. Так, шрифт трохи впливає на сприйняття, проте від його заміни розповідь не втрачає сенсу. Вихідний шрифт завжди можна повернути. Це не завдає шкоди.

Ось ще один спосіб спростити завдання: давайте використовувати простий текст без курсиву, напівжирного накреслення, підкреслення, кольорів, обведення букв, підрядкових та надрядкових індексів, а також без діакритичних значків. Жодних Å, é, ñ або ö. Тільки символи латинського алфавіту, оскільки з них складається 99% англійських слів.

У ході попередніх досліджень кодів Морзе та Брайля ми бачили, як літери алфавіту можуть бути представлені у двійковому форматі. Незважаючи на те, що ці системи чудово справляються з завданнями, що стоять перед ними, вони мають недоліки, коли мова заходить про комп'ютери. В абетці Морзе коди мають *різну довжину.* Наприклад, літери, що часто використовуються, позначаються короткими кодами, більш рідкісні – довгими. Як бачите, такий код підходить для телеграфу, але не для комп'ютерів. Крім того, код Морзе не робить відмінностей між літерами у верхньому та нижньому регістрі.

Коди Брайля мають фіксовану довжину, кожна літера представлена шістьма бітами, що краще для комп'ютерів. Абетка Брайля також розрізняє літеру верхнього та нижнього регістру, хоча й використовує для цього спеціальний *escape-* код, що вказує, що за ним слідує символ у верхньому регістрі. По суті, це означає, що для кожної великої літери потрібні два коди, а не один. Цифри відображаються за допомогою коду *перемикання,* який дає зрозуміти, що наступні коди представляють числа до тих пір, поки не зустрінеться інший код перемикання, що сигналізує повернення до подання букв.

Наша мета – розробка такого набору кодованих символів, щоб наведену нижче пропозицію можна було зашифрувати за допомогою серії кодів, кожен з яких – це певна кількість бітів.

Маю 27 сестер.

Одні коди представлятимуть літери, інші – розділові знаки, треті – числа. Ще нам необхідний код, який відповідає пробілу між словами. Наведене речення складається з 17 символів (включно з пробілами). Послідовність кодів для шифрування подібних речень часто називається текстовим *рядком.*

Те, що нам потрібні коди для чисел у текстовому рядку, таких як «27», може здатися дивним, оскільки ми представляли числа за допомогою бітів у попередніх розділах. Ми можемо припустити, що кодами для цифр 2 і 7 у цьому реченні є просто двійкові числа 10 та 111. Однак це не обов'язково так. У контексті такого речення із символами 2 та 7 можна звертатися як з будь-якими іншими символами у письмовій мові. Їм можуть відповідати коди, які зовсім не пов'язані з фактичними значеннями цих чисел.

Ймовірно, найбільш економічним текстовим кодом є 5-бітний код, створений в 1874 для друкуючого телеграфу Емілем Бодо, співробітником французької телеграфної служби, яка почала використовувати цей код в 1877 році. Надалі код було вдосконалено Дональдом Мюрреєм і стандартизовано у 1931 році Міжнародним консультаційним комітетом з телефонії та телеграфії (Comité Consultatif International Téléphonique et Télégraphique, CCITT; зараз – Міжнародний союз електрозв'язку – International Telecommunication Union, ITU). Офіційно цей код називається міжнародним телеграфним алфавітом №2 (International Telegraph Alphabet №2, ITA-2); у Сполучених Штатах він більш відомий як код Бодо, хоча правильніше було б називати його кодом Мюррея.

У XX столітті код Бодо часто застосовувався у *телетайпних апаратах.* Клавіатура телетайпу Бодо схожа на машинку, але в неї тільки 30 клавіш і пробіл. Клавіші телетайпа – просто перемикачі, використання яких призводить до генерації двійкового коду та його передачі вихідним кабелем апарату, біт за бітом. Крім того, телетайп передбачає друкарський механізм. Коди, що проходять через вхідний кабель телетайпа, активують електромагніти, які друкують символи на папері.

Оскільки код Бодо 5-бітний, він містить лише 32 елементи. Шістнадцяткові значення цих кодів знаходяться в діапазоні від 00h до 1Fh. У наступній таблиці наведено відповідність цих 32 кодів буквам латинського алфавіту.

Hex Code	Baudot Letter	Hex Code	Baudot Letter
00		10	E
01	T	11	Z
02	*Carriage Return*	12	D
03	O	13	B
04	*Space*	14	S
05	H	15	Y
06	N	16	F
07	M	17	X
08	*Line Feed*	18	A
09	L	19	W
0A	R	1A	J
0B	G	1B	*Figure Shift*
0C	I	1C	U
0D	P	1D	Q
0E	C	1E	K
0F	V	1F	*Letter Shift*

Коду 00h не присвоюється нічого. З кодів, що залишилися, двадцять шість призначаються буквам алфавіту, а решта п'ять відповідають допоміжним діям, виділеним у таблиці курсивом.

Код 04h – це пробіл, який створює простір між словами, коди 02h і 08h – повернення каретки та переклад рядка. Ці поняття застосовуються при використанні друкарської машинки. Коли ви досягаєте кінця рядка, друкуючи, ви натискаєте на важіль або кнопку, яка виконує дві дії. По-перше, каретка переміщається вправо, завдяки чому наступний рядок починається з лівого краю листа (повернення каретки). По-друге, друкарська машинка прокручує валик так, щоб наступний рядок знаходився під тим, який ви щойно надрукували (переклад рядка). У системі Бодо ці два коди генеруються окремими кнопками. Телетайпний апарат Бодо реагує на ці два коди під час друку.

Для отримання цифр і розділових знаків у системі Бодо використовується код 1Bh, позначений у таблиці фразою «Переключення на цифри». Усі наступні за ним коди інтерпретуються як цифри або розділові знаки, поки код «Переключення на літери» (1Fh) не просигналізує повернення до літер. У наступній таблиці представлені коди, що відповідають цифрам і розділовим знакам.

Hex Code	Baudot Figure	Hex Code	Baudot Figure
00		10	3
01	5	11	+
02	*Carriage Return*	12	*Who Are You?*
03	9	13	?
04	*Space*	14	'
05	#	15	6
06	,	16	$
07	.	17	/
08	*Line Feed*	18	-
09	)	19	2
0A	4	1A	*Bell*
0B	&	1B	*Figure Shift*
0C	8	1C	7
0D	0	1D	1
0E	:	1E	(
0F	=	1F	*Letter Shift*

У стандарті ITU коди 05h, 0Bh та 16h не визначені: вони зарезервовані «для національного використання». У таблиці показано, як ці коди застосовувалися у Сполучених Штатах. Ці коди зазвичай підходять для букв з діакритичними знаками з деяких європейських мов. Код Сигнал призначений для подачі телетайпом чутного звукового сигналу, «Хто це?» активує механізм, з якого телетайп може ідентифікувати себе.

Як і абетка Морзе, цей 5-бітний код не передбачає відмінностей між великими і малими літерами. Вираз I spent $25 today («Сьогодні я витратив 25 доларів») шифрується наступною послідовністю шістнадцяткових значень.

I SPENT $25 TODAY.
0C 04 14 0D 10 06 01 04 1B 16 19 01 1F 04 01 03 12 18 15 1B 07 02 08

Зверніть увагу на три коди перемикання: 1Bh прямо перед числом, 1Fh після числа та 1Bh перед точкою в кінці речення. Рядок завершується кодами повернення каретки та перекладу рядка.

На жаль, якщо ви двічі відправите цю послідовність значень на пристрій друку телетайпа, отримайте наступний результат.

I SPENT $25 TODAY.
8 '03,5 $25 TODAY.

Що трапилося? Справа в тому, що останній код перемикання, отриманий друкарським апаратом перед другим рядком, представляв код перемикання на цифри, тому коди початку другого рядка були інтерпретовані як цифри.

Подібні проблеми типові під час використання кодів перемикання. Незважаючи на безумовну економічність коду Бодо, краще було б зупинитися на унікальних кодах для цифр і розділових знаків, а також окремих кодах для малих і великих літер.

Якщо ми хочемо з'ясувати, скільки біт нам необхідно для створення кращої порівняно з кодом Бодо системи кодування символів, потрібно просто скласти їх: 52 кодові слова для великих і малих літер, десять кодових слів для цифр від 0 до 9. Це вже 62 кодові слова. Якщо додати кілька розділових знаків, отримаємо 64 кодових слова, а значить, нам потрібно більше шести біт. Однак ми ще не скоро перевищимо значення 128 символів, при якому знадобиться вісім біт.

Як бачите, потрібні сім біт для представлення символів англійського тексту, якщо ми хочемо обійтися без перемикання між літерами нижнього та верхнього регістру.

Що це за коди? Фактично коди можуть бути будь-якими. Якби ми збиралися створити власний комп'ютер, зібрати для цього всі необхідні апаратні засоби, самі його запрограмувати і ніколи не використовувати для підключення до будь-якої іншої машини ми могли б придумати власні коди. Все, що потрібно, це призначити унікальний код кожному символу, який ми збираємося використовувати.

Оскільки комп'ютери рідко створюються і працюють окремо один від одного, розумніше було б домовитися про застосування тих самих кодів. Таким чином, сконструйовані нами комп'ютери можуть бути більш сумісними один з одним і, ймовірно, зможуть навіть обмінюватися текстовою інформацією.

Крім того, не слід надавати коди випадковим чином. Так, коли ми працюємо з текстом на комп'ютері, призначення послідовних кодів букв алфавіту дає деякі переваги. Наприклад, цей метод упорядкування спрощує сортування за абеткою.

На щастя, такий стандарт вже розроблений: *Американський стандартний код обміну інформацією* (American Standard Code for Information Interchange, ASCII). Він був прийнятий у 1967 році і залишається одним із найважливіших у всій комп'ютерній індустрії. Однак він має один виняток (про який розповім пізніше). Працюючи з текстом на комп'ютері, ви можете бути впевнені, що стандарт ASCII якимось чином задіяний у цьому процесі.

ASCII – це 7-бітний код, що використовує двійкові значення в діапазоні від 0000000 до 1111111, які відповідають шістнадцятковим кодам від 00h до 7Fh. Давайте розберемо коди ASCII. Почнемо не з самого початку, оскільки перші 32 коди концептуально трохи складніші за інші, а з другої групи, що складається з 32 кодів, яка включає розділові знаки і десять цифр. У наступній таблиці перелічені шістнадцяткові коди та відповідні символи.

Hex Code	ASCII Character	Hex Code	ASCII Character
20	*Space*	30	0
21	!	31	1
22	"	32	2
23	#	33	3
24	$	34	4
25	%	35	5
26	&	36	6
27	'	37	7
28	(	38	8
29	)	39	9
2A	*	3A	:
2B	+	3B	;
2C	,	3C	<
2D	-	3D	=
2E	.	3E	>
2F	/	3F	?

Зверніть увагу: код 20h відповідає пробілу, що відокремлює слова один від одного.

Наступні 32 коди – великі літери і деякі додаткові розділові знаки. Крім значка @ та символу підкреслення, ці знаки зазвичай відсутні на друкарських машинках, однак є стандартними для комп'ютерних клавіатур.

Hex Code	ASCII Character	Hex Code	ASCII Character
40	@	50	P
41	A	51	Q
42	B	52	R
43	C	53	S
44	D	54	T
45	E	55	U
46	F	56	V
47	G	57	W
48	H	58	X
49	I	59	Y
4A	J	5A	Z
4B	K	5B	[
4C	L	5C	\
4D	M	5D	]
4E	N	5E	^
4F	O	5F	_

Наступна група з 32 символів: всі малі літери і деякі додаткові розділові знаки, які, знову ж таки, рідко зустрічаються на друкарських машинах.

Hex Code	ASCII Character	Hex Code	ASCII Character
60	`	70	p
61	a	71	q
62	b	72	r
63	c	73	s
64	d	74	t
65	e	75	u
66	f	76	v
67	g	77	w
68	h	78	x
69	i	79	y
6A	j	7A	z
6B	k	7B	{
6C	l	7C	|
6D	m	7D	}
6E	n	7E	~
6F	o		

У цій таблиці відсутній останній символ, що відповідає коду 7Fh. Наведені вище три таблиці містять 95 символів. Оскільки код ASCII є 7-бітним, він допускає використання 128 кодів, тому нам має бути доступним ще 33 коди, до яких ми скоро і дістанемося.

Текстовий рядок Hello, you! («Привіт тобі!») може бути представлений у кодуванні ASCII з використанням наступних шістнадцяткових кодів.

Hello, you!

48 65 6C 6C 6F 2C 20 79 6F 75 21

Зверніть увагу на кому (код 2C), пробіл (код 20) і знак оклику (код 21), а також на коди, відповідні буквам. Представлення короткого речення I am 12 years old («Мені 12 років») у кодуванні ASCII таке.

I am 12 years old.

49 20 61 6D 20 31 32 20 79 65 61 72 73 20 6F 6C 64 2E

Число 12 у цьому реченні відображено шістнадцятковими числами 31h і 32h, тобто кодами ASCII для цифр 1 і 2. Коли число 12 – частина тексту, його *не можна* представляти за допомогою шістнадцяткових кодів 01h і 02h, BCD-коду 12h або шістнадцяткового кода 0Ch. Всі ці коди мають у системі ASCII зовсім інші значення.

Конкретна велика літера в системі ASCII відрізняється від своєї малої версії на 20h. Цей факт дозволяє легко написати код, який, наприклад, переводить текстовий рядок у верхній регістр. Припустимо, що певна область пам'яті містить текстовий рядок по одному символу на байт. Наступна підпрограма для процесора 8080 передбачає, що адреса першого символу текстового рядка зберігається у регістрі HL. Регістр C включає довжину цього текстового рядка, тобто кількість символів.

```
Capitalize: MOV A,C     ; C = number of characters left
            CPI A,00h   ; Compare with 0
            JZ  AllDone ; If C is 0, we're finished
```

```
            MOV A,[HL]  ; Get the next character
            CPI A,61h   ; Check if it's less than 'a'
            JC SkipIt   ; If so, ignore it

            CPI A,7Bh   ; Check if it's greater than 'z'
            JNC SkipIt  ; If so, ignore it

            SBI A,20h   ; It's lowercase, so subtract 20h
            MOV [HL],A  ; Store the character

SkipIt:     INX HL          ; Increment the text address
            DCR C           ; Decrement the counter
            JMP Capitalize  ; Go back to the top

AllDone:    RET
```

Оператор, який віднімає 20h з коду малої літери для її перетворення на прописну, можна замінити наступним фрагментом.

ANI A, DFh

Інструкція ANI (AND Immediate) виконує побітову операцію між значенням в акумуляторі і значенням DFh, яке в двійковому форматі дорівнює 11011111. Під словом «побітова» припускаю, що така команда виконує операцію І між кожною парою відповідних бітів, що додають два числа. Ця операція І зберігає всі біти, що містяться в акумуляторі (А), за винятком третього зліва, значення якого задається рівним 0, що, у свою чергу, перетворює малу букву ASCII в прописну.

Наведені вище 95 кодів – це так звані *друковані символи,* оскільки вони мають графічне уявлення. Крім них у кодуванні ASCII також передбачено 33 *керуючі символи,* які не мають графічного представлення, але виконують певні функції. Для повноти наведу ці 33 керуючі символи. Не турбуйтеся, якщо вони будуть здаватися вам незрозумілими. Кодування ASCII спочатку призначалося для використання в телетайпах, тому багато з її кодів в даний час втратили своє значення.

Hex Code	Acronym	Control Character Name
00	NUL	Null (Nothing)
01	SOH	Start of Heading
02	STX	Start of Text
03	ETX	End of Text
04	EOT	End of Transmission
05	ENQ	Enquiry (i.e., Inquiry)
06	ACK	Acknowledge
07	BEL	Bell
08	BS	Backspace
09	HT	Horizontal Tabulation
0A	LF	Line Feed
0B	VT	Vertical Tabulation
0C	FF	Form Feed
0D	CR	Carriage Return
0E	SO	Shift-Out
0F	SI	Shift-In
10	DLE	Data Link Escape
11	DC1	Device Control 1
12	DC2	Device Control 2
13	DC3	Device Control 3
14	DC4	Device Control 4
15	NAK	Negative Acknowledge
16	SYN	Synchronous Idle
17	ETB	End of Transmission Block
18	CAN	Cancel
19	EM	End of Medium
1A	SUB	Substitute Character
1B	ESC	Escape
1C	FS	File Separator or Information Separator 4
1D	GS	Group Separator or Information Separator 3
1E	RS	Record Separator or Information Separator 2
1F	US	Unit Separator or Information Separator 1
7F	DEL	Delete

Керуючі символи можуть використовуватися поряд з друкованими для елементарного форматування тексту. Це найпростіше зрозуміти, якщо ви подумаєте про такий пристрій, як телетайп або простий принтер, який надрукує на сторінці символи під впливом потоку кодів ASCII. Отримавши код символу, друкувальна головка пристрою наносить символ і переміщається на одну позицію праворуч. Найбільш важливі керуючі символи змінюють цю нормальну поведінку.

Розглянемо наступний рядок шістнадцяткових значень.

41 09 42 09 43 09

Символ 09 – код горизонтальної табуляції, або табулятор. Якщо уявити, що всі горизонтальні позиції символу на сторінці принтера нумеруються починаючи з 0, код табуляції дає команду надрукувати наступний символ на наступній горизонтальній позиції, номер якої кратний восьми,

A B C

наприклад.

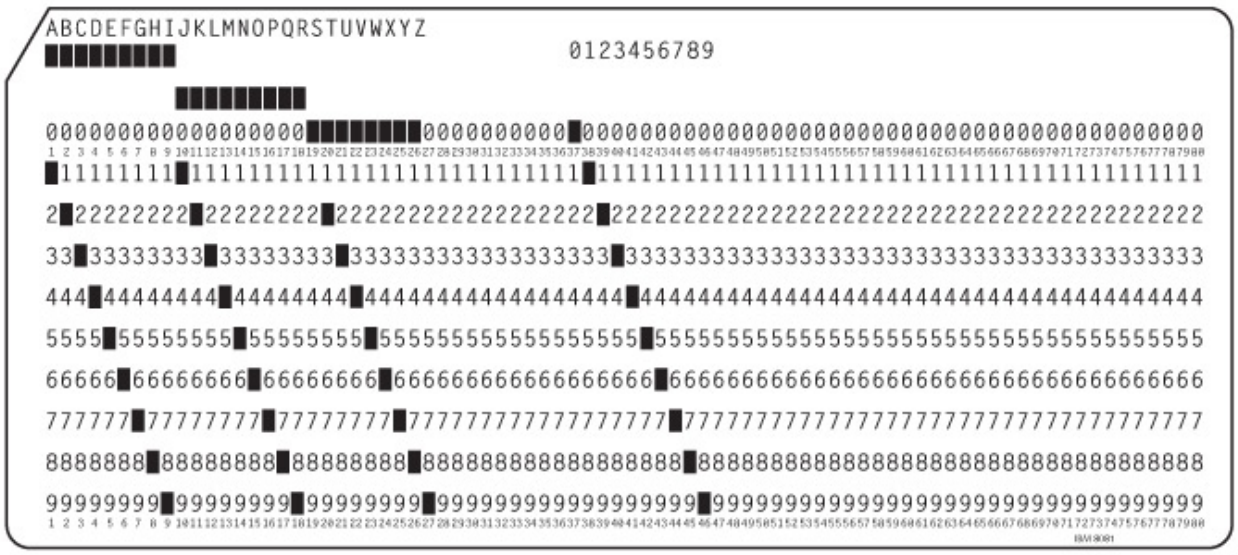
Це зручний спосіб розташування тексту стовпцями.

Навіть сьогодні багато комп'ютерних принтерів реагують на код переходу до нової сторінки (12h), витягуючи поточну сторінку і починаючи друкувати на новій сторінці.

Код для повернення на один символ назад (backspace) можна використовувати для друку складених символів на деяких старих принтерах. Припустимо, вам потрібно, щоб телетайпний апарат відобразив малу букву *е* зі зворотним апострофом: *è.* Це завдання можна вирішити за допомогою шістнадцяткових кодів 65 08 60.

Сьогодні найбільш важливими є керуючі коди, для повернення каретки і перекладу рядка, які мають те ж значення, що і аналогічні коди Бодо. У відповідь на код повернення каретки друкуюча головка принтера переміщається до лівого краю сторінки, а у відповідь на код перекладу рядка на один рядок вниз. Зазвичай для переходу на новий рядок потрібні обидва коди. Код повернення каретки може використовуватися сам по собі для друку поверх існуючого рядка, а код перекладу рядка – для того, щоб перейти до наступного, не переміщаючись до лівого краю сторінки.

Незважаючи на те, що кодування ASCII – домінуючий стандарт у комп'ютерному світі, воно не використовується в багатьох великих комп'ютерних системах IBM. Для мейнфреймів System/360 компанія розробила власний 8-бітний код EBCDIC (Extended BCD Interchange Code – розширений двійково-десятковий код обміну інформацією) – розширений варіант раннього 6-бітного коду BCDIC, отриманого з кодів, що використовуються на перфокартах IBM. Ці перфокарти, що дозволяють зберігати по 80 текстових символів, були створені IBM в 1928 і використовувалися



протягом більше 50 років.

Розглядаючи взаємозв'язок між перфокартами та відповідними ним 8-бітними кодами символів EBCDIC, майте на увазі, що ці коди розроблялися протягом багатьох десятиліть під впливом різних технологій. З цієї причини не варто очікувати від них надмірної логічності чи узгодженості.

На перфокарті символ кодується комбінацією з одного або кількох прямокутних отворів, пробитих в одному стовпчику. Сам символ часто друкується у верхній частині картки. Нижні десять рядків пронумеровано від 0 до 9. Ненумерований рядок над нульовим рядком вважається одинадцятим, а верхній – дванадцятим; десятий рядок відсутній.

Ось ще кілька термінів з області застосування перфокарт IBM: рядки з нульової по дев'яту називаються *цифровими рядками,* або *цифровою пробивкою.* Одинадцятий і дванадцятий рядки –

зонні рядки, або *зонне пробивання.* Іноді нульовий і дев'ятий рядки вважалися не цифровими, а зонними, що викликало плутанину.

Восьмибітний код символу EBCDIC складається зі старшої та молодшої тетради (чотири біта). Молодша тетрада – код BCD, що відповідає цифровому пробиванню символа; старша тетрада – код, який довільно можна поставити у відповідність до зонного пробивання символу. З теми 19 ви пам'ятаєте, що BCD означає *двійково-десятковий код* – 4-бітовий код цифр від 0 до 9.

Для цифр від 0 до 9 немає ніякої зонної пробивки. Відсутність пробивання відповідає старшій тетраді 1111. Молодша тетрада – код BCD цифрового пробивання. У таблиці наведено коди EBCDIC для цифр від 0 до 9.

Hex Code	EBCDIC Character
F0	0
F1	1
F2	2
F3	3
F4	4
F5	5
F6	6
F7	7
F8	8
F9	9

Для великих літер тетрада 1100 відповідає зонному пробиванню тільки дванадцятого рядка, тетрада 1101 – зонному пробиванню тільки одинадцятого рядка, тетрада 1110 – зонному пробиванню тільки нульового рядка. Наведемо коди EBCDIC для великих літер.

Hex Code	EBCDIC Character	Hex Code	EBCDIC Character	Hex Code	EBCDIC Character
C1	A	D1	J		
C2	B	D2	K	E2	S
C3	C	D3	L	E3	T
C4	D	D4	M	E4	U
C5	E	D5	N	E5	V
C6	F	D6	O	E6	W
C7	G	D7	P	E7	X
C8	H	D8	Q	E8	Y
C9	I	D9	R	E9	Z

Зверніть увагу на прогалини у нумерації цих кодів. Якщо ви використовуєте текст EBCDIC для написання програм, ці прогалини можуть заважати.

Рядкові літери відповідають тій же цифровій пробивці, що і великі, але іншій зонній пробивці. Для малих літер від *a* до *i* пробиваються дванадцятий і нульовий рядки, що відповідає коду 1000, для літер від *j* до *r* – дванадцятий і одинадцятий рядки, що відповідає коду 1001, для літер від *s* до *z* – одинадцятий і нульовий рядки, що відповідає коду 1010. Коди EBCDIC для малих літер такі.

Hex Code	EBCDIC Cha		EBCDIC Character	Hex Code	EBCDIC Character
81	a	91	j		
82	b	92	k	A2	s
83	c	93	l	A3	t
84	d	94	m	A4	u
85	e	95	n	A5	v
86	f	96	o	A6	w
87	g	97	p	A7	x
88	h	98	q	A8	y
89	i	99	r	A9	z

Зрозуміло, існують і інші коди EBCDIC – для розділових знаків і керуючих символів, проте ми навряд чи потребуємо проведення повномасштабного дослідження цієї системи.

На перший погляд може здатися, що одного стовпця перфокарти IBM достатньо для кодування 12 біт інформації. Кожен отвір відповідає одному біту, чи не так? За ідеєю, для кодування символу ASCII має бути достатньо семи з 12 позицій у кожному стовпці. Однак на практиці це не дуже добре працює, оскільки при цьому пробивається занадто багато отворів, через що карта стає крихкою.

Багато 8-бітових кодів EBCDIC не визначені. Це говорить про те, що використання 7-бітного кодування ASCII має більше сенсу. За часів розробки системи ASCII пам'ять була дорогою. Деякі люди вважали, що кодування ASCII має бути 6-бітним і передбачати символ перемикання між малими і великими літерами для економії.

Як тільки ця ідея була відкинута, інші стали вважати, що кодування ASCII має бути 8-бітним, оскільки навіть у той час вважалося, що в комп'ютерах буде застосовуватися швидше 8-бітна архітектура, ніж 7-бітна. Звісно, сучасним стандартом є 8-бітові байти. Незважаючи на те, що технічно ASCII – це 7-бітове кодування, майже завжди його коди зберігаються як 8-бітові значення.

Еквівалентність байтів і символів, безумовно, зручна, оскільки ми можемо приблизно уявити, який обсяг пам'яті займає конкретний текстовий документ, просто підрахувавши кількість символів. Деяким людям набагато легше зрозуміти, що таке кілобайт і мегабайт пам'яті, коли цей обсяг ставиться у відповідність до обсягу тексту.

Наприклад, звичайна машинописна сторінка формату A4 з полями 2,5 сантиметра та подвійним міжрядковим інтервалом містить приблизно 27 рядків тексту. На кожному рядку шириною 16 сантиметрів міститься 65 символів. Вміст такої сторінки займає загалом близько 1750 байт. Текст, що міститься на машинописній сторінці з одинарним міжрядковим інтервалом, займає приблизно вдвічі більше – 3,5 кілобайти.

Сторінка в журналі New Yorker включає три стовпці тексту, у кожному з яких міститься 60 рядків по 40 символів. Це 7200 символів (байтів) на сторінку.

Сторінка газети New York Times містить шість стовпців тексту. Якби вся вона була зайнята текстом без заголовків або зображень (що було б незвичайно), то кожен стовпець складався б із 155 рядків по 35 символів. Тоді на всій сторінці було б 32 550 символів, або 32 кілобайти.

На сторінці звичайної книги налічується близько 500 слів. У середньому слово складається приблизно з семи літер, хоча скоріше з восьми, якщо враховувати прогалину. Таким чином, на сторінці книги близько 3000 символів. Припустимо, що середня книга складається із 333 сторінок. Це значення, хоч би яким дивним воно здавалося, дозволяє сказати, що обсяг тексту середньої книги становить близько одного мільйона байт, або один мегабайт.

Зрозуміло, обсяг тексту книг варіюється у великому діапазоні:

- "Великий Гетсбі" Френсіса Скотта Фіцджеральда – близько 300 кілобайт;
- "Над прірвою в житі" Джерома Селінджера – близько 400 кілобайт;
- «Пригоди Гекльберрі Фінна» Марка Твена – близько 540 кілобайт;
- «Грона гніву» Джона Стейнбека – близько одного мегабайта;
- "Мобі Дік, або Білий кит" Германа Мелвілла - 1,3 мегабайта;

• «Історія Тома Джонса» Генрі Філдінга – 2,25 мегабайта;
• «Віднесені вітром» Маргарет Мітчелл – 2,5 мегабайта;
• «Протистояння» Стівена Кінга – 2,7 мегабайта;
• «У пошуках втраченого часу» Марселя Пруста – 7,7 мегабайта.

У Бібліотеці Конгресу Сполучених Штатів налічується близько 20 мільйонів книг, в яких міститься 20 трильйонів символів, що відповідає 20 терабайтам текстових даних. (Крім тексту, там є безліч фотографій та аудіозаписів.)

Незважаючи на те, що ASCII, безумовно, є основним стандартом в комп'ютерній індустрії, він не ідеальний. Проблема в тому, що цей стандарт надто американський! Дійсно, ASCII не цілком підходить навіть для тих країн, де основною мовою є англійська. Кодування ASCII включає символ долара, але де символ британського фунта? А як щодо літер із діакритичними значками, що використовуються у багатьох західноєвропейських мовах? Я вже не кажу про нелатинські алфавіти, такі як грецька, арабська, іврит та кирилиця. А що щодо символів складового листа брахмі, що застосовується в Індії та Південно-Східній Азії, на якому засновані такі види писемності, як деванагарі, бенгалі, тайська та тибетська? Як за допомогою 7-бітного коду в принципі можна уявити *десятки тисяч* ідеограм китайської, японської та корейської мов, а також десять із лишком тисяч хангильських складів?

У період розробки системи ASCII потребам деяких інших країн приділяли увагу, хоча нелатинські алфавіти особливо не враховувалися. Відповідно до опублікованого стандарту ASCII, десять його кодів (40h, 5Bh, 5Ch, 5Dh, 5Eh, 60h, 7Bh, 7Ch, 7Dh та 7Eh) можна перевизначити відповідно до національних потреб. Крім того, символ решітки (#) можна замінити символом британського фунта (£), а символ долара ($) – узагальненим для валюти символом (¤). Очевидно, що заміна символів має сенс лише тоді, коли всі користувачі конкретного текстового документа, що містить ці перевизначені коди, знають про цю зміну.

Оскільки багато комп'ютерних систем зберігають символи у вигляді 8-бітових значень, можна розширити їх набір з 128 до 256. У такому наборі коди з 00h по 7Fh визначаються так само, як і в звичайній системі ASCII, а коди з 80h по FFh можуть представляти що щось зовсім інше. Цей метод використовувався для визначення додаткових кодів для букв з діакритичними значками та нелатинськими алфавітами. Як приклад наведу набір кодів для літер кирилиці. У представленій таблиці старша тетрада шістнадцяткового коду символу вказана у верхньому рядку, а молодша – у лівому стовпці.

	8-	**9-**	**A-**	**E-**
-0	А	Р	а	р
-1	Б	С	б	с
-2	В	Т	в	т
-3	Г	У	г	у
-4	Д	Ф	д	ф
-5	Е	Х	е	х
-6	Ж	Ц	ж	ц
-7	З	Ч	з	ч
-8	И	Ш	и	ш
-9	Й	Щ	й	щ
-A	К	Ъ	к	ъ
-B	Л	Ы	л	ы
-C	М	Ь	м	ь
-D	Н	Э	н	э
-E	О	Ю	о	ю
-F	П	Я	п	я

Символом коду A0h призначено нерозривну прогалину. Зазвичай коли комп'ютерна програма форматує текст у рядки та абзаци, то розрив рядка дорівнює пробілу, код ASCII якого 20h. Код A0h повинен відображатися як пропуск, але не може використовуватися для розриву рядків. Нерозривний пропуск може знадобитися, наприклад, у фразі «WW II». Символ коду ADh – м'яке перенесення. Його використовують для поділу голосних у середині слова. На друкованій сторінці він з'являється лише тоді, коли необхідно перенести слово з одного рядка на інший.

На жаль, за минулі десятиліття було створено багато *різних* розширень кодування ASCII, що призвело до великої плутанини і негативно вплинуло на сумісність. Набір ASCII був розширений радикальнішим чином для кодування ідеограм китайської, японської та корейської мов. В одному популярному кодуванні під назвою Shift-JIS (Japan Industrial Standard – японський промисловий стандарт) коди з 81h по 9Fh фактично становлять перший байт двобайтового коду символу. Таким чином система Shift-JIS дозволяє кодувати близько 6000 додаткових символів. На жаль, Shift-JIS – не єдина система, яка використовує такий підхід. В Азії широко поширені ще три стандартні набори двобайтових символів (Double-byte character sets, DBCS).

Існування кількох несумісних наборів двобайтових символів лише одна з проблем. Сумно, деякі символи, зокрема звичайні символи ASCII, представлені однобайтовими кодами, а тисячі ідеограм – 2-байтовими. Це ускладнює роботу з такими наборами.

Вважаючи за переважне наявність єдиної однозначної системи кодування символів, що підходить для всіх мов світу, в 1988 році кілька великих комп'ютерних компаній об'єдналися для розробки альтернативи ASCII, що отримала назву Unicode. На відміну від 7-бітного коду ASCII, Unicode – 16-бітний. Для кожного символу в кодуванні Unicode потрібно два байти. Отже, Unicode передбачає коди символів від 0000h до FFFFh, тобто може становити 65 536 різних символів. Цього достатньо для охоплення всіх мов світу, які з великою ймовірністю будуть використовуватися в комп'ютерній індустрії, навіть з можливістю розширення.

Кодування Unicode створювалося не з нуля. Перші 128 символів Unicode, коди яких перебувають у діапазоні від 0000h до 007Fh, відповідають тим самим символам у системі ASCII. Крім того, коди Unicode з 00A0h по 00FFh – це коди розширення ASCII для латинського алфавіту Latin Alphabet № 1. У Unicode також включені інші світові стандарти.

Незважаючи на те, що Unicode – очевидне покращення систем кодування, це не гарантує його миттєвого ухвалення. Система ASCII і безліч недосконалих розширень настільки укорінилися у світі комп'ютерних технологій, що витіснити їх буде складно.

Єдина справжня проблема системи Unicode у тому, що вона робить недійсною колишню відповідність між одним текстовим символом та одним байтом пам'яті. Закодований за допомогою стандарту ASCII роман "Грона гніву" займає один мегабайт, а в кодуванні Unicode – два мегабайти. Однак це невелика плата за універсальну та однозначну систему кодування.

Тема 14. Шини

Незважаючи на основну роль процесора, ним будова комп'ютера не обмежується. Крім нього, комп'ютеру потрібна оперативна пам'ять (RAM) для зберігання машинного коду, який виконуватиме процесор. Крім того, комп'ютер повинен передбачати спосіб запису цього коду в пам'ять (пристрій введення) та відображення результатів роботи програми (пристрій виведення). Як ви пам'ятаєте, пам'ять RAM є енергозалежною: її вміст втрачається при відключенні живлення. Так що ще один корисний компонент комп'ютера – довготривалий пристрій, в якому код і дані можуть зберігатися після його вимкнення.

Усі інтегральні схеми (ІВ), із яких складається комп'ютер, монтуються на друкованих платах. У деяких невеликих комп'ютерах всі ІС можуть розміститися на одній платі. Однак набагато частіше різні компоненти розміщуються на двох або більше платах, які обмінюються даними за допомогою *шини.* Шина – це просто набір цифрових сигналів, що передаються по різних середовищах, які подаються на кожну з плат комп'ютера. Ці сигнали поділяються на чотири категорії:

- адресні сигнали – сигнали, що генеруються мікропроцесором і використовуються в основному для адресації оперативної пам'яті; вони також застосовуються для звернення до інших підключених до комп'ютера пристроїв;
- сигнали виведення даних – ці сигнали теж генеруються мікропроцесором і використовуються для запису даних в оперативну пам'ять або їх передачі до інших пристроїв; будьте обережні з термінами "введення" та "виведення": сигнал виведення даних мікропроцесора – сигнал введення даних для оперативної пам'яті та інших пристроїв;
- сигнали введення даних – сигнали, які генеруються іншими компонентами комп'ютера та зчитуються мікропроцесором; сигнали введення даних найчастіше виникають на виходах RAM, завдяки чому мікропроцесор зчитує вміст пам'яті, однак інші компоненти також генерують сигнали введення даних для мікропроцесора;
- керуючі сигнали – різноманітні сигнали, які зазвичай відповідають керуючим сигналам конкретного мікропроцесора, на базі якого побудовано комп'ютер; керуючі сигнали можуть генеруватися в мікропроцесорі або інших пристроях, яким потрібно передати дані в мікропроцесор; приклад керуючого сигналу – сигнал, за допомогою якого мікропроцесор вказує на необхідність запису деяких даних у конкретну комірку пам'яті.

На додаток до цього шина подає живлення на різні плати, які входять до складу комп'ютера.

Однією з перших популярних шин для ПК була модель S-100, представлена в 1975 як компонент першого персонального комп'ютера «Альтаїр» компанії MITS. Незважаючи на те, що ця шина розроблялася для мікропроцесора 8080, пізніше вона була адаптована під інші процесори, такі як 6800. Розмір плати S-100 становить 13,4 × 25,4 сантиметра. Одна зі сторін друкованої плати вставляється в роз'єм, який має 100 контактів (звідси назва).

Комп'ютер із шиною S-100 включає велику плату, звану *материнською,* яка містить кілька (близько дванадцяти) пов'язаних один з одним гнізд для плат S-100. Ці гнізда іноді називаються *слотами розширення,* у яких вставляються плати S-100, чи *плати розширення.* Одну плату S-100 займає мікропроцесор 8080 та допоміжні чіпи (про деякі я згадав у темі 19). Оперативна пам'ять займає одну або кілька плат.

Оскільки шина S-100 розроблялася для мікросхеми 8080, вона має 16 адресних ліній, вісім ліній для введення та вісім ліній для виведення даних. Як ви пам'ятаєте, у самому процесорі 8080 лінії для введення та виведення даних об'єднані. Сигнали поділяються на вхідні та вихідні за допомогою інших мікросхем, встановлених на тій самій платі, що і процесор 8080. Шина також передбачає вісім ліній для *переривань* сигналів, що генеруються іншими пристроями, коли їм потрібно привернути увагу центрального процесора. Як побачимо далі, клавіатура може генерувати сигнал переривання при натисканні кнопки. У відповідь на це процесор 8080 виконує коротку програму, щоб визначити, яка клавіша була натиснута, і зробити відповідну дію. Для обробки переривань до плати з процесором 8080 зазвичай підключається чіп Intel 8214 (пристрій для управління пріоритетними перериваннями). Коли виникає переривання, цей чіп генерує для процесора 8080 відповідний сигнал. Коли останній підтверджує отримання запиту на переривання, чіп посилає команду RST (Restart – перезапуск), яка змушує мікропроцесор зберегти

поточне значення лічильника команд і залежно від отриманого переривання перейти до команди в осередку 0000h, 0008h, 0010h, 002 або 0038h.

Якби ви розробили нову комп'ютерну систему із шиною нового типу, потрібно було б вирішити, чи опублікувати технічні характеристики шини чи зберегти їх у таємниці.

Якщо їх опублікувати, інші, так звані *сторонні,* виробники зможуть проектувати і продавати плати розширення, сумісні з цією шиною. Доступність додаткових плат розширення робить комп'ютер більш функціональним, отже, зростає попит. Зростання продажів комп'ютерів веде до збільшення ринку плат розширення. Це спонукає розробників більшості невеликих комп'ютерних систем дотримуватися принципу *відкритої архітектури,* що дозволяє іншим виробникам створювати периферійні пристрої. Згодом шина може перетворитися на галузевий *стандарт,* а стандарти мають велике значення для індустрії персональних комп'ютерів.

Найвідомішим ПК із відкритою архітектурою був перший IBM PC, випущений восени 1981 року. IBM опублікувала технічний довідник, що містить повні електричні схеми цього комп'ютера і всіх плат розширення. Цей довідник став важливим інструментом, що дозволило багатьом виробникам створити не тільки свої плати розширення для IBM PC, але й *клони,* які були практично ідентичні цьому комп'ютеру і використовували програмне забезпечення.

На численних нащадків першого комп'ютера IBM PC нині припадає 90% ринку. Незважаючи на те, що IBM належить лише невелика частка цього ринку, вона могла б бути ще меншою, якби архітектура її першого комп'ютера була *закритою.* Архітектура комп'ютера Apple Macintosh спочатку була закритою. Незважаючи на рідкісні експерименти з відкритою архітектурою, це прийняте на початку рішення, ймовірно, пояснює, чому на частку Macintosh припадає менше 10% ринку настільних ПК. При цьому закрита архітектура комп'ютерної системи не заважає стороннім компаніям писати для неї *програмне забезпечення.* Тільки виробники деяких відеоігрових консолей забороняють іншим компаніям створювати програми для своїх систем.

У першому комп'ютері IBM PC використовувався процесор Intel 8088, що дозволяв адресувати один мегабайт пам'яті. Незважаючи на те, що мікропроцесор 8088 – 16-розрядний, обмін даними з пам'яттю він здійснює фрагментами по вісім біт. Шина, яку компанія IBM розробила для свого першого комп'ютера, тепер має назву ISA (Industry Standard Architecture – архітектура промислового стандарту). Така шина передбачає 62 лінії, з яких 20 адресних, вісім використовуються для введення та виведення даних, шість – для запитів на переривання, три – для запитів на прямий доступ до пам'яті (Direct Memory *Access,* DMA). Режим DMA дозволяє прискорити роботу пристроїв для зберігання даних. Зазвичай читання та запис даних на пам‘ять здійснює мікропроцесор. Завдяки режиму DMA інший пристрій може перехопити керування шиною і здійснити обмін даними безпосередньо з пам'яттю, минаючи мікропроцесор.

У системі S-100 усі компоненти розміщені на платах розширення. У комп'ютері IBM PC мікропроцесор, деякі допоміжні чіпи та частина оперативної пам'яті містилися на платі, яку компанія назвала *системною,* хоча ця плата також часто називається материнською, чи головною.

В 1984 IBM представила персональний комп'ютер PC/AT, в якому використовувався 16-розрядний мікропроцесор Intel 80286, що дозволяв адресувати 16 мегабайт пам'яті. IBM встановила ту ж шину, але додала ще один 36-контактний роз'єм, який включав сім адресних ліній (хоча потрібно всього чотири), вісім ліній для введення та виведення даних, п'ять ліній для запитів на переривання та чотири лінії для запитів на прямий доступ до пам'яті.

Шини необхідно модернізувати або замінювати у міру того, як мікропроцесори переростають їх за розрядністю даних (наприклад, зі збільшенням з 8 до 32 біт), за кількістю адресних ліній або швидкодією. Перші шини створювалися для мікропроцесорів, які працювали з тактовою частотою в кілька мегагерц, але не в кілька сотень. Коли шина працює на швидкостях, для яких не призначена, вона може стати джерелом високочастотних перешкод, що викликають статичний або інший шум у радіоприймачах і телевізорах, що працюють поблизу.

В 1987 IBM випустила шину MCA (Micro Channel Architecture – мікроканальна архітектура). Деякі аспекти цієї шини були запатентовані, що дозволило компанії отримати ліцензійні платежі. Ймовірно, саме з цієї причини MCA шина не *стала* галузевим стандартом. А 1988 року консорціум із дев'яти компаній (до якого IBM не увійшла) виготовив альтернативну 32-розрядну шину EISA (Extended Industry Standard Architecture – розширена архітектура промислового стандарту). Наприкінці 1990-х років у IBM-сумісних комп'ютерах широко використовувалася розроблена

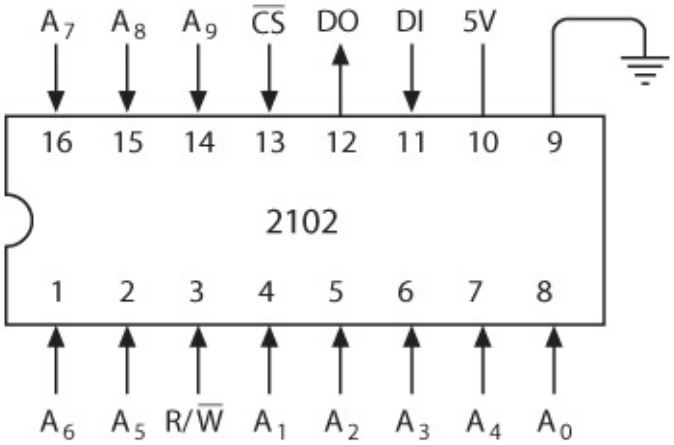
компанією Intel шина PCI (Peripheral Component Interconnect – взаємозв'язок периферійних компонентів).

Щоб зрозуміти, як працюють різні компоненти комп'ютера, знову треба повернутися в середину 1970-х, коли все було дуже просто. Представимо розробку плат для комп'ютера "Альтаїр" або для нашого власного комп'ютера на базі процесора 8080 або 6800. Для нього нам, ймовірно, потрібно зібрати масив RAM, клавіатуру для введення даних, екран для їх виведення, а також деякий пристрій, що дозволяє зберігати інформацію після вимикання. Розглянемо різні *інтерфейси,* які можемо створити для підключення перелічених компонентів до комп'ютера.

Як ви пам'ятаєте з теми 16 масив RAM передбачає адресні входи, входи для введення і виведення даних, а також сигнал, що використовується для запису даних в пам'ять. Від кількості адресних входів залежить кількість окремих значень, які можна зберегти у масиві RAM:

$$\text{Кількість значень у масиві RAM} = 2^{\text{кількість адресних входів}}$$

Кількість входів для введення і виведення даних визначає розрядність значень, що зберігаються.



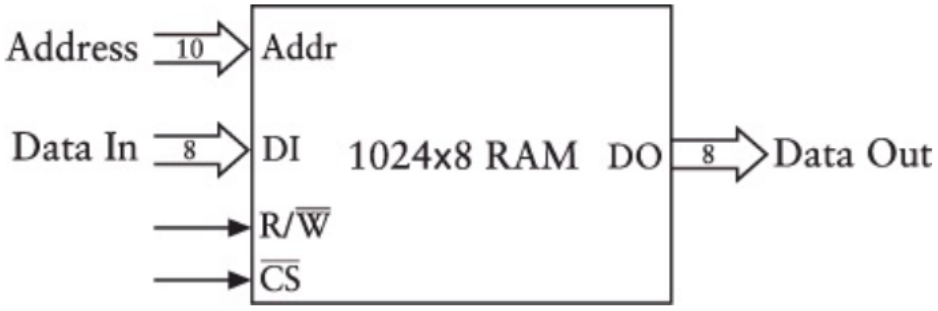
У 1970-х років у домашніх комп'ютерах часто застосовувалася мікросхема пам'яті 2102.

Мікросхема 2102 була створена за технологією МОП (метал – оксид – напівпровідник), або MOS (metal-oxide-semiconductor), як і самі мікропроцесори 8080 та 6800. Мікросхеми МОП можна легко підключити до мікросхем ТТЛ; від останніх вони відрізняються вищою щільністю транзисторів. До того ж, вони працюють не так швидко.

Якщо підрахувати кількість адресних входів (A_0-A_9) і звернути увагу, що тут лише один вхід для введення (DI) і один вхід для виведення (DO) даних, ви зрозумієте, що ємність мікросхеми обмежена 1024 біт. Залежно від типу використовуваної мікросхеми 2102 *час доступу,* тобто час, що пройшов між подачею адреси на адресні входи та виведення відповідних даних на вихід DO, становить від 350 до 1000 наносекунд. При зчитуванні інформації з пам'яті сигнал R/$\overline{W}$ (читання/запис) зазвичай дорівнює 1. Коли вам потрібно записати дані в мікросхему, цей сигнал повинен дорівнювати 0 протягом від 170 до 550 наносекунд, знову ж таки залежно від типу використовуваної мікросхеми 2102.

Особливий інтерес представляє сигнал $\overline{CS}$ (Chip Select – вибір мікросхеми). Коли цей сигнал дорівнює 1, мікросхема *не обрана,* отже, вона не реагує на сигнал R/$\overline{W}$. Сигнал $\overline{CS}$ здійснює на мікросхему також інший важливий вплив, про який розповім трохи згодом.

Очевидно, якщо ви збираєте масив пам'яті для 8-розрядного мікропроцесора, потрібно організувати цю пам'ять так, щоб у ній зберігалися 8-бітні, а не однобітні значення. Для збереження цілих байтів доведеться поєднати принаймні вісім таких мікросхем, підключивши їх адресні входи, сигнали R/$\overline{W}$ та $\overline{CS}$ один до одного. Результат можна зобразити так.

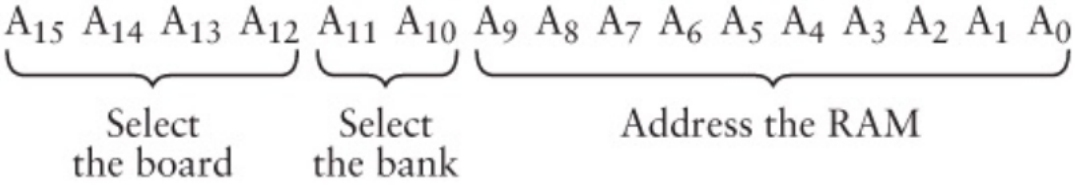


Це масив RAM 1024×8 ємністю один кілобайт.

З практичної погляду ці мікросхеми пам'яті необхідно монтувати на друковану плату. Скільки може поміститися на одній платі? Якщо розміщувати їх максимально близько, то одну плату S-100 можна встановити 64 такі мікросхеми, які забезпечать вісім кілобайт пам'яті.

Спробуємо обійтися пам'яттю в чотири кілобайти, використовуючи лише 32 чіпи. Кожен набір мікросхем, з'єднаних між собою для зберігання цілого кілобайта (як показано вище), називається *банком.* Плата пам'яті ємністю чотири кілобайти містить чотири банки, кожен із яких складається з восьми мікросхем.

У таких 8-розрядних мікропроцесорах, як 8080 та 6800, використовуються 16-розрядні адреси, за допомогою яких можна адресувати 64 кілобайти пам'яті. Коли ви підключаєте плату пам'яті ємністю чотири кілобайти, що містить чотири банки мікросхем, 16 адресних сигналів плати пам'яті виконують такі функції.



Десять адресних сигналів, з A_0 по A_9 безпосередньо підключені до мікросхем RAM. Адресні сигнали A_{10} і A_{11} дозволяють вибрати до якого з чотирьох банків здійснюється звернення. Адресні сигнали з A_{12} по A_{15} визначають, які адреси відносяться до конкретної плати, тобто на які адреси ця плата реагує. Наша плата пам'яті ємністю чотири кілобайти може займати один із шістнадцяти 4-кілобайтних діапазонів у всьому адресному просторі процесора ємністю 64 кілобайти:

- від 0000 до 0FFFh;
- від 1000 до 1FFFh;
- від 2000 до 2FFFh;

…

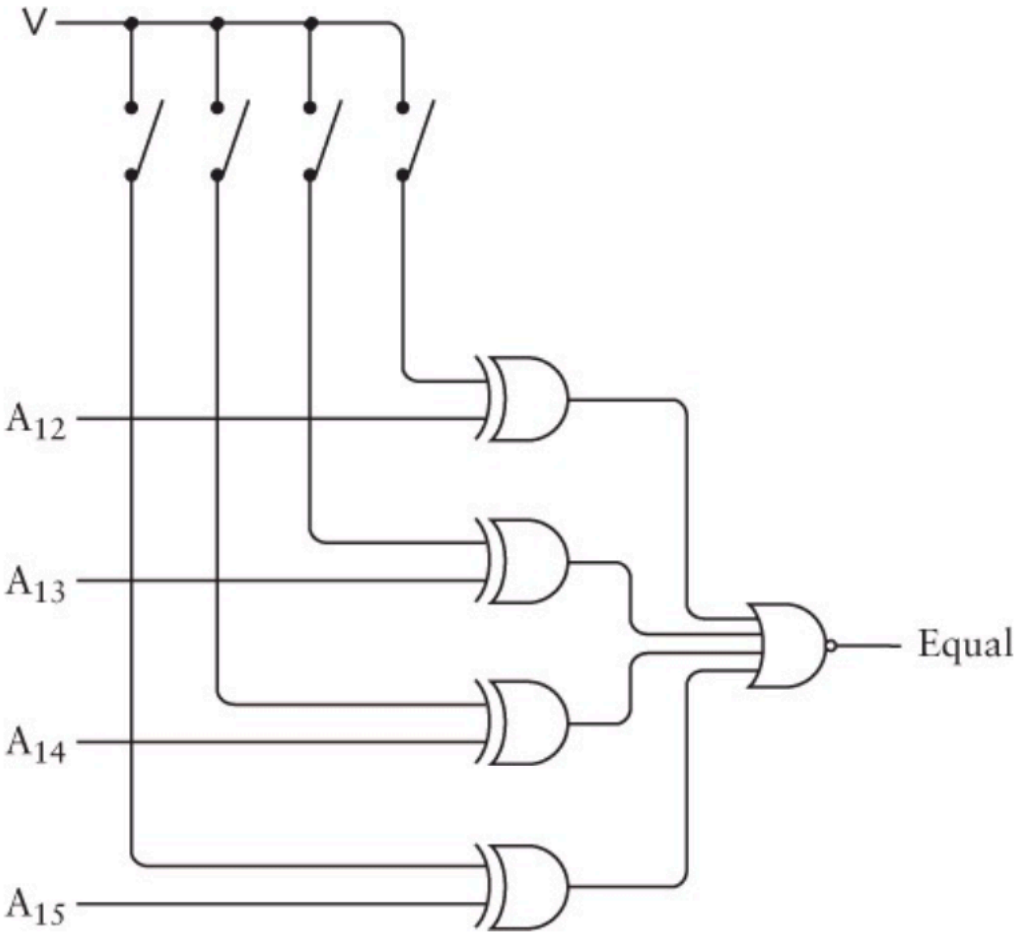
- від F000h до FFFFh.

Припустимо, ми вирішили, що до цієї плати пам'яті ємністю чотири кілобайти будуть відноситися адреси в діапазоні від A000h до AFFFh. Значить, адреси з A000h по A3FFh будуть зайняті першим банком однокілобайтних мікросхем, з A400h по A7FFh – другим, з A800h по ABFFh – третім, з AC00h по AFFFh – четвертим.

Зазвичай 4-кілобайтна плата пам'яті передбачає можливість зміни діапазону адрес, на які вона реагує. Для цього використовується так званий *DIP-перемикач* (Dual Inline Package), що представляє собою набір крихітних перемикачів (від двох до дванадцяти) у корпусі з дворядним

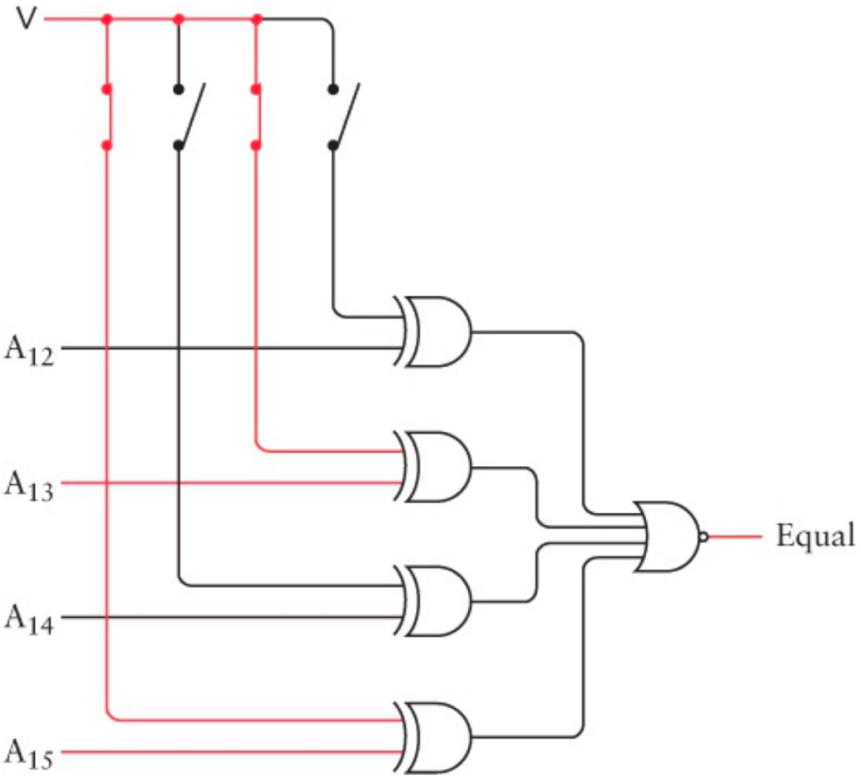
розташуванням виводів, який вставляється у гніздо для інтегральної мікросхеми.

Можна підключити його до чотирьох старших адресних розрядів шини, використовуючи схему *компаратор.*

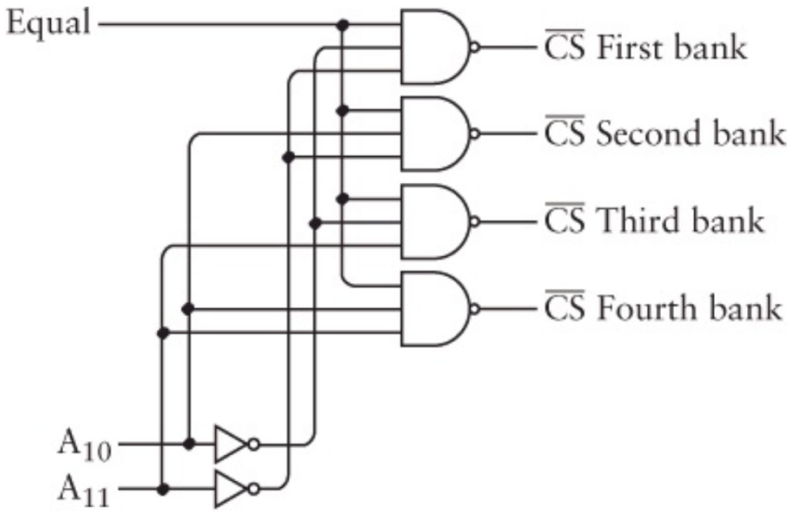


Як ви пам'ятаєте, вихід вентиля Викл-АБО дорівнює 1 тільки тоді, коли на його входи подаються різні значення. Вихід вентиля Викл-АБО – 0, якщо обидва вхідних значення однакові.

Наприклад, замикання перемикачів, що відповідають лініям A_{13} та A_{15}, призведе до того, що плата пам'яті буде реагувати на адреси з A000h по AFFFh. Коли значення адресних сигналів шини A_{12}, A_{13}, A_{14} і A_{15} рівні значенням, встановленим за допомогою перемикачів, виходи всіх чотирьох вентилів Викл-АБО рівні 0, значить, вихід вентиля АБО-НЕ дорівнює 1.



Потім ви можете об'єднати цей сигнал "Рівне" з дешифратором "2 на 4", щоб генерувати сигнали $\overline{CS}$ для кожного з чотирьох банків пам'яті.



Якщо сигнал A_{10} дорівнює 0, а A_{11} – 1, значить, обраний третій банк.

Якщо ви ще пам'ятаєте складний процес збирання масивів RAM з теми 16, можете припустити, що нам потрібно використовувати вісім селекторів "4 на 1" для вибору правильних вихідних сигналів від чотирьох банків пам'яті. Однак у цьому випадку вони не будуть потрібні, і ось чому.

Як правило, вихідні сигнали інтегральних схем, сумісних з ТТЛ-чіпами, набувають значення або більше 2,2 вольта (логічна одиниця), або менше 0,4 вольта (логічний нуль). Що станеться, якщо спробувати з'єднати ці вихідні сигнали? Наприклад, до чого призведе з'єднання вихідного сигналу, що дорівнює 1, однієї схеми і вихідного сигналу, що дорівнює 0, іншої? Виразно відповісти на це питання не можна, тому виходи інтегральних схем зазвичай не поєднуються один з одним.

Вихідний сигнал мікросхеми 2102 відомий як сигнал *з трьома станами.* Крім логічних 0 і 1, для цього вихідного сигналу передбачено третій стан, що відповідає відсутності будь-якого сигналу, ніби це виведення мікросхеми взагалі ні до чого не підключено. Вихідний сигнал мікросхеми 2102 переходить у цей третій стан, коли вхід CS дорівнює 1. Це означає, що ми *можемо* з'єднати відповідні вихідні сигнали всіх чотирьох банків і використовувати ці вісім комбінованих виходів як вісім ліній шини для введення даних.

Загострюю вашу увагу на вихідному сигналі із трьома станами, тому що він відіграє важливу роль у роботі шини. Практично всі плати, підключені до шини, використовують лінії введення даних. У будь-який момент лише одна підключена до шини плата може задіяти ці лінії. При цьому вихідні сигнали інших плат повинні бути у третьому стані.

Мікросхема 2102 – це *статична* пам'ять з довільним доступом, або SRAM (Static Random Access Memory), яка відрізняється від *динамічної* пам'яті з довільним доступом, або DRAM (Dynamic Random Access Memory). Пам'яті SRAM зазвичай потрібно чотири транзистори для зберігання одного біта (це не так багато, як у тригерах з теми 16). Пам'яті DRAM для цього потрібний лише один транзистор. Проте нестача пам'яті DRAM – необхідність використання складніших допоміжних схем.

Вміст пам'яті SRAM, наприклад мікросхеми 2102, зберігається лише за наявності живлення. Якщо живлення відключається, вміст зникає. Це стосується і пам'яті DRAM, проте мікросхема DRAM також потребує періодичного зчитування даних, навіть якщо в них не має необхідності. Такий цикл *оновлення* повинен повторюватися кілька сотень разів на секунду. Це все одно що періодично штурхати людину, щоб вона не заснула.

Незважаючи на складності, пов'язані з використанням пам'яті DRAM, ємність цих мікросхем, що постійно збільшується, зробила їх стандартом. У 1975 році компанія Intel представила мікросхему DRAM ємністю 16384 біт. Відповідно до закону Мура ємність мікросхем DRAM збільшується вчетверо кожні три роки. Сучасні комп'ютери зазвичай передбачають гнізда для пам'яті прямо на системній платі, куди вставляються невеликі плати, які називаються *модулями*

пам'яті SIMM (Single Inline Memory Module) або DIMM (Dual Inline Memory Module) з кількома мікросхемами DRAM.

Тепер ви знаєте, як створювати плати пам'яті, проте не варто заповнювати пам'яттю весь адресний простір мікропроцесора. Потрібно виділити деяку його частину пристрою виведення.

Електронно-променева трубка (ЕПТ) була найбільш поширеним пристроєм виведення для комп'ютерів. ЕПТ, підключена до комп'ютера, зазвичай називається *дисплеєм,* або *монітором;* Електронний компонент, що подає дисплею сигнал, називається *відеоадаптером.* Часто відеоадаптер займає у комп'ютері окрему *відеокарту.*

Незважаючи на те, що двовимірне зображення дисплея або телевізора може здатися складним, насправді воно створюється одним безперервним променем світла, який швидко пробігає екраном. Промінь починає свій рух у верхньому лівому кутку і переміщається по екрану вправо, після чого повертається до лівого краю, щоб розпочати наступну горизонтальну лінію, відому як *рядок розгортки.* Переміщення променя справа наліво до початку чергового рядка – *зворотний хід по горизонталі.* Коли промінь закінчує останній рядок, він повертається з нижнього правого до верхнього лівого кута екрана (*зворотний хід по вертикалі)*, і весь процес починається знову. За американськими стандартами телевізійного мовлення це має відбуватися 60 разів на секунду. Така *частота розгортки* досить висока, щоб зображення не мерехтіло.

ТБ влаштований складніше через використання *черезрядкової розгортки.* Один *кадр* – окреме нерухоме зображення – розбивається на два поля з парних рядків (перше) та непарних (друге). *Частота малої розгортки,* або частота проходження рядків, становить 15 750 герц. Якщо поділити це число на 60 герц, отримаємо 262,5 рядка в одному полі кадру. Цілий кадр містить вдвічі більше, тобто 525 рядків розгортки.

Незалежно від виду розгортки дисплея безперервний промінь світла, що породжує відео, управляється одним безперервним сигналом. Хоча аудіо- та відеосигнали телевізійної програми об'єднуються під час трансляції або передачі через систему кабельного телебачення, у результаті вони поділяються. *Відеосигнал,* який я опишу, ідентичний сигналу, що подається на відповідні входи та виходи відеомагнітофонів, камер та деяких телевізорів.

У разі чорно-білого телебачення цей відеосигнал є досить простим і зрозумілим. (Колір все ускладнює.) Шістдесят разів на секунду в цей сигнал записується *імпульс вертикальної синхронізації,* який вказує на початок поля. Цей імпульс – подача нульової напруги (земля) протягом 400 мікросекунд. *Імпульс горизонтальної синхронізації* вказує на початок кожного рядка розгортки: 15750 разів на секунду протягом п'яти мікросекунд подається нульова напруга. Між імпульсами горизонтальної синхронізації значення сигналу варіюється від 0,5 вольта (чорний) до двох вольт (білий), відтінки сірого відповідають значенням від 0,5 до 2 вольт.

Таким чином, зображення телевізора є частково цифровим та частково аналоговим. Це зображення розділене по вертикалі на 525 рядків, проте всередині кожного з рядків розгортки напруга постійно змінюється, що обумовлює варіації яскравості. Проте напруга неспроможна змінюватися довільно. Існує гранична швидкість, з якою телеекран може реагувати зміну сигналу. Цей показник називається "пропускна здатність".

Пропускна спроможність – надзвичайно важлива концепція у сфері зв'язку, оскільки визначає, який обсяг інформації можна передати конкретним каналом. У разі телевізора пропускна здатність – гранична швидкість, з якою відеосигнал може змінитися від рівня чорного до рівня білого, а потім знову до рівня чорного. За американськими стандартами телевізійного мовлення смуга пропускання приблизно дорівнює 4,2 мегагерца.

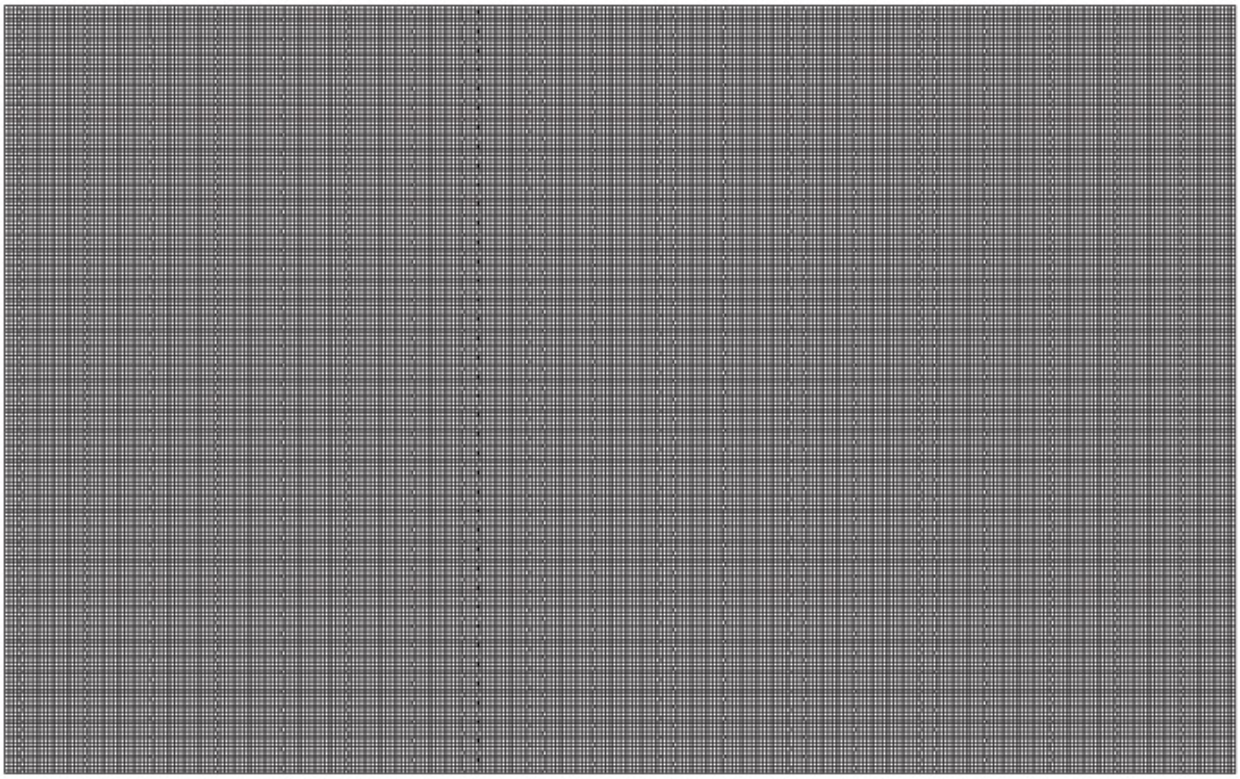
Коли дисплей необхідно підключити до комп'ютера, його незручно подавати у вигляді гібрида аналогового та цифрового пристрою. Його легко розглядати як повністю цифровий пристрій. З погляду комп'ютера відеозображення зручніше відобразити як прямокутну сітку з дискретними крапками, які називаються *пікселями* (pixel від picture element – «елемент зображень»).

Смуга пропускання відеодисплея обмежує кількість пікселів у горизонтальному рядку розгортки. Я визначив смугу пропускання як швидкість, з якою відеосигнал може змінитись від рівня чорного до рівня білого, а потім знову до рівня чорного. Смуга пропускання телевізорів, що дорівнює 4,2 мегагерца, допускає створення двох пікселів 4,2 мільйона разів на секунду. Якщо розділити добуток 2 × 4 200 000 на 15 750 (частота малої розгортки), отримаємо 533 пікселі в кожному рядку. Однак приблизно третина цих пікселів виявляється невидимою через їхнє

знаходження на далеких кінцях зображення або через зворотний хід променя по горизонталі. Таким чином, кожен рядок – це приблизно 320 пікселів.

Так само по вертикалі не налічується 525 пікселів. Деякі з них губляться у верхній та нижній частині екрану і під час зворотного ходу променя по вертикалі. Крім того, *не слід* покладатися на черезрядкову розгортку під час використання телевізора як дисплея. Таким чином, реальна кількість пікселів по вертикалі дорівнює приблизно двомстам.

Отже, можна сказати, що *роздільна здатність* примітивного відеоадаптера, підключеного до звичайного телевізора, становить 320 пікселів по горизонталі на 200 пікселів по вертикалі, або



320 × 200.

Щоб визначити загальну кількість пікселів у цій сітці, можете підрахувати їх або просто перемножити числа 320 і 200, отримавши 64 тисячі пікселів. Залежно від того, як ви налаштували відеоадаптер (про що розповім трохи пізніше), кожен піксель може бути або чорним, або білим (чорно-біле зображення), або мати певний колір (кольорове зображення).

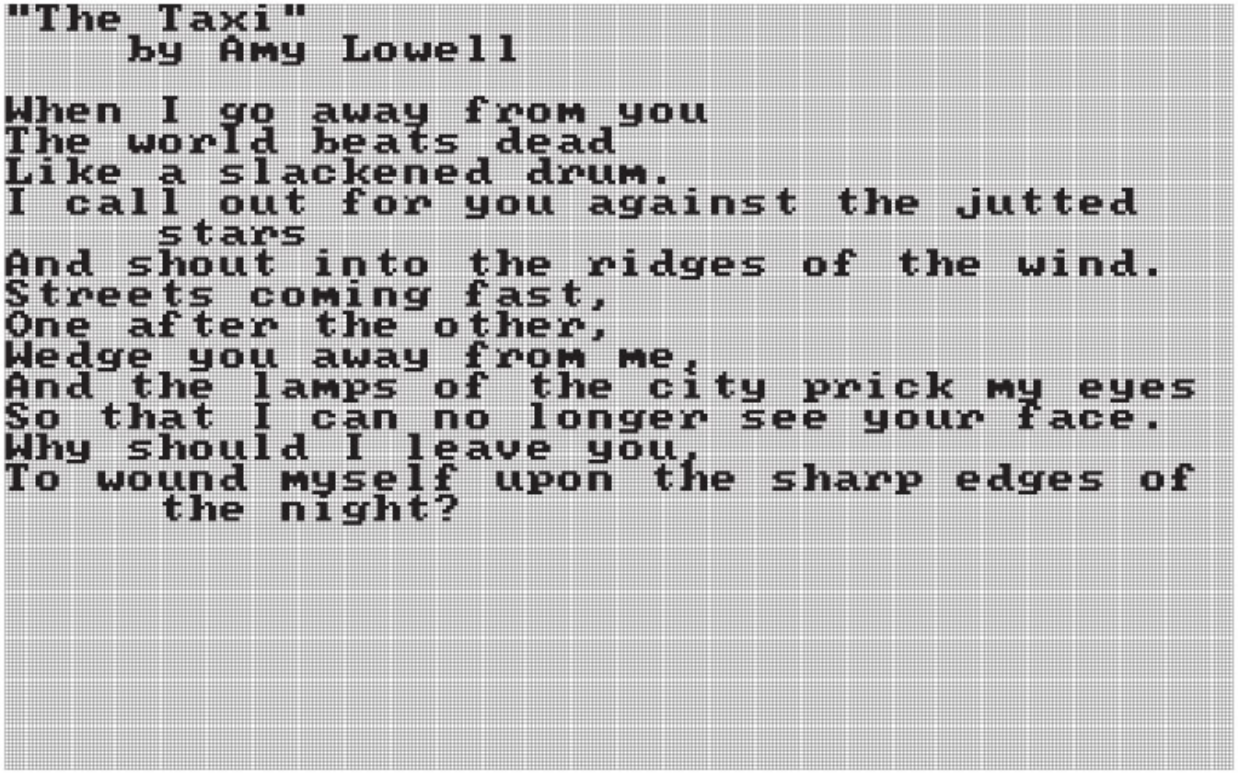
Припустимо, що нам потрібно відобразити на цьому дисплеї деякий текст. Скільки тексту на ньому може вміститися?

Очевидно, все залежить від того, скільки пікселів використовується для відображення кожного текстового символу. Далі представлений один із можливих підходів, при якому для кожного символу використовується сітка 8×8 (64 пікселі).

Ці символи відповідають кодам ASCII з 20h до 7Fh. Коди ASCII з 00h до 1Fh відведені під символи, що не відображаються.

У наведеному прикладі кожен символ відповідає не тільки 7-бітному ASCII-коду, але і 64 бітам на екрані, що визначає його зовнішній вигляд. Ці 64 біти теж можна розглядати як своєрідний код.

При такому способі представлення символів можна помістити на відеодисплеї з роздільною здатністю 320 × 200 пікселів 25 рядків по 40 символів. Цього достатньо, наприклад, для



короткого вірша Емі Лоуелл.

Відеоадаптер використовує деяку кількість оперативної пам'яті для зберігання вмісту дисплея, а мікропроцесор повинен мати можливість записувати дані в цю пам'ять для зміни

зображення на цьому дисплеї. Найзручніше, коли пам'ять RAM є частиною загального адресного простору процесора. Скільки ж оперативної пам'яті знадобиться відеоадаптеру, який я описую?

Це непросте питання! Значення може змінюватись від 1 до 192 кілобайт!

Спочатку оцінимо нижню межу. Один із способів зменшення вимог до пам'яті полягає в тому, щоб обмежити можливості адаптера лише відображенням тексту. Ми вже з'ясували, що можемо відобразити 25 рядків по 40 символів або 1000 символів. У пам'яті RAM на відеоплаті повинні зберігатися лише 7-бітові коди ASCII відповідних символів. Тисяча 7-бітових значень – приблизно 1024 байт, або один кілобайт.

Така плата відеоадаптера також повинна оснащуватися *генератором символів,* що містить точкові шаблони всіх символів ASCII – на зразок тих, які були показані на одному з попередніх зображень. Як правило, генератор символів – це *постійний запам'ятовуючий пристрій,* або ПЗП (Read-Only Memory, ROM), інтегральна схема, виготовлена таким чином, що у відповідь на звернення до конкретної адреси завжди видаються ті самі дані. На відміну від пам'яті RAM, ПЗП не передбачає жодних сигналів для введення даних.

Пам'ять ПЗУ можна вважати схемою, що перетворює один код на інший. ПЗП, в якому зберігаються точкові шаблони (8 × 8 пікселів) для 128 символів ASCII, може передбачати сім адресних входів (для ASCII-кодів) та 64 виходи для даних. Таким чином, ПЗУ перетворює 7-бітний ASCII-код на 64-бітний, що визначає зовнішній вигляд символу. Однак наявність 64 виходів зробили б чіп надто громіздким! Набагато зручніше використовувати десять адресних входів та вісім виходів. Сім адресних сигналів вказують на конкретний символ ASCII. (Ці сім біт адреси подаються з виходів RAM на відеоплаті.) Інші три адресні сигнали визначають рядок. Наприклад, біти адреси 000 відповідають верхньому рядку точкового шаблону, а біти 111 – нижньому рядку, вісім вихідних бітів – вісім пікселів кожного рядка.

Припустимо, що ASCII-код дорівнює 41h. Цей код можна порівняти з великою літерою *A.* Її точковий шаблон складається з восьми рядків на вісім біт. У наступній таблиці наведено 10-бітові адреси (пробіл відокремлює ASCII-код від коду рядка) та сигнали на виходах для даних, що відповідають великій літері *A.*

Address	Data Output
1000001 000	00110000
1000001 001	01111000
1000001 010	11001100
1000001 011	11001100
1000001 100	11111100
1000001 101	11001100
1000001 110	11001100
1000001 111	00000000

Ви бачите букву *A,* намальовану одиницями на тлі нулів?

Відеоадаптер, який відтворює лише текст, також повинен передбачати можливість відображення *курсору* – рисочки в тому місці екрана, де з'явиться наступний введений з клавіатури символ. Номери рядка та стовпця, що збігаються з позицією курсору, зазвичай зберігаються у двох 8-бітових регістрах на відеоплаті, в які мікропроцесор може записувати значення.

Якщо плата відеоадаптера не обмежується відтворенням тексту, вона називається *графічною.* Записуючи дані в оперативну пам'ять графічної плати, процесор може виводити на екран зображення, а також відображати текст різних розмірів і стилів. Графічним платам потрібно

більше пам'яті, ніж текстовим. При роздільній здатності 320 × 200 зображення складається з 64 тисяч пікселів. Якщо кожен піксель відповідає одному біту пам'яті, такій платі потрібно 64 тисячі біт, або 8000 байт оперативної пам'яті. Це, звісно, абсолютний мінімум. Те, що один біт пам'яті дорівнює одному пікселю, дозволяє використовувати тільки два кольори, наприклад, чорний і білий. Значення нуль біт може співвідноситися із чорним пікселем, а один біт – із білим.

Зрозуміло, чорно-білі телевізори відображають не лише чорні та білі пікселі. Вони також здатні відтворювати безліч відтінків сірого. Щоб графічна плата могла передати відтінки сірого, кожному пікселю зазвичай відводиться цілий *байт* пам'яті RAM, причому значення 00h відповідає чорному кольору, FFh – білому, проте проміжні значення – різноманітним відтінкам сірого. Відеоплаті, що відображає 256 відтінків сірого на дисплеї з роздільною здатністю 320 × 200, потрібно 64 тисяч *байт* пам'яті. Це майже весь адресний простір одного із 8-бітних мікропроцесорів, про які я говорив раніше!

Використання повнокольорового режиму передбачає виділення трьох байтів на кожен піксель. Якщо ви розглянете екран телевізора або комп'ютерного дисплея через збільшувальне скло, виявите, що кожен колір створюється різними комбінаціями основних кольорів: червоного, зеленого та синього. Щоб відобразити весь діапазон, потрібно по одному байту для вказівки інтенсивності кожного з трьох основних кольорів, тобто 192 тисяч байт пам'яті RAM. (В останній темі я докладно зупинюся на графіці.)

Кількість різних кольорів, які може відобразити відеоадаптер, визначається кількістю бітів, виділених для кожного пікселя. Це співвідношення може здатися знайомим, оскільки в ньому використовується ступінь двійки:

$$\text{Кількість кольорів} = 2^{\text{кількість біт на піксель.}}$$

Роздільна здатність 320 × 200 пікселів максимальна для типового телевізора. Саме тому монітори, створені спеціально для комп'ютерів, мають ширшу смугу пропускання, ніж телевізійні екрани. Перші монітори, які продавалися з комп'ютером IBM PC у 1981 році, могли відображати 25 рядків по 80 символів. Саме така кількість символів відтворювалася CRT дисплеями, які використовувалися з великими і дорогими мейнфреймами IBM. Для IBM число 80 має особливе значення. Чому? *Саме стільки символів містилося на перфокарті IBM!* Справді, спочатку дисплеї CRT, підключені до мейнфреймів, часто використовувалися для перегляду вмісту перфокарт. Навіть сьогодні дехто продовжує називати рядки дисплея, що відображає тільки текст *картами.*

Протягом багатьох років відеоадаптери вдосконалювалися в плані роздільної здатності та кольору. Важлива віха була досягнута в 1987 році, коли в персональних комп'ютерах IBM серії Personal System/2 та Apple Macintosh II почали застосовуватися відеоадаптери, здатні відображати 640 пікселів по горизонталі та 480 пікселів по вертикалі. З того часу цей показник – мінімальна стандартна роздільна здатність для відео.

Це може здатися неймовірним, але причина важливості роздільної здатності 640 × 480 пов'язана з роботою Томаса Едісона! Приблизно в 1889 році, коли Едісон і його інженер Вільям Діксон працювали над кінокамерою «Кінетограф» і проектором «Кінетоскоп», вони вирішили зробити так, щоб ширина зображення, що рухається, на третину перевищувала його висоту. Співвідношення ширини та висоти зображення називається *характеристичним ставленням.* Співвідношення, яке вибрали Едісон і Діксон, зазвичай виражається як 1,33:1, або 4:3. Протягом понад 60 років це співвідношення використовувалося при виробництві кінофільмів та конструюванні телевізорів. Тільки на початку 1950-х голлівудські студії почали знімати фільми у широкоекранному форматі, що й склало конкуренцію телебаченню завдяки виходу за рамки співвідношення 4:3.

Більшість комп'ютерних моніторів (і телевізорів) мають характерні відношення 4:3, в чому ви можете переконатися за допомогою лінійки. Роздільна здатність 640 × 480 також відповідає відношенню 4:3. Це означає, що горизонтальна лінія, що складається, наприклад, зі 100 пікселів, має ту ж фізичну довжину, що і вертикальна лінія зі 100 пікселів. Таким чином, пікселі є *квадратними,* що вважається кращим для комп'ютерної графіки.

Відеоадаптери та монітори практично завжди мають роздільну здатність 640 × 480, проте вони також здатні працювати у відеорежимах з роздільною здатністю 800 × 600, 1024 × 768, 1280 × 960 та 1600 × 1200.

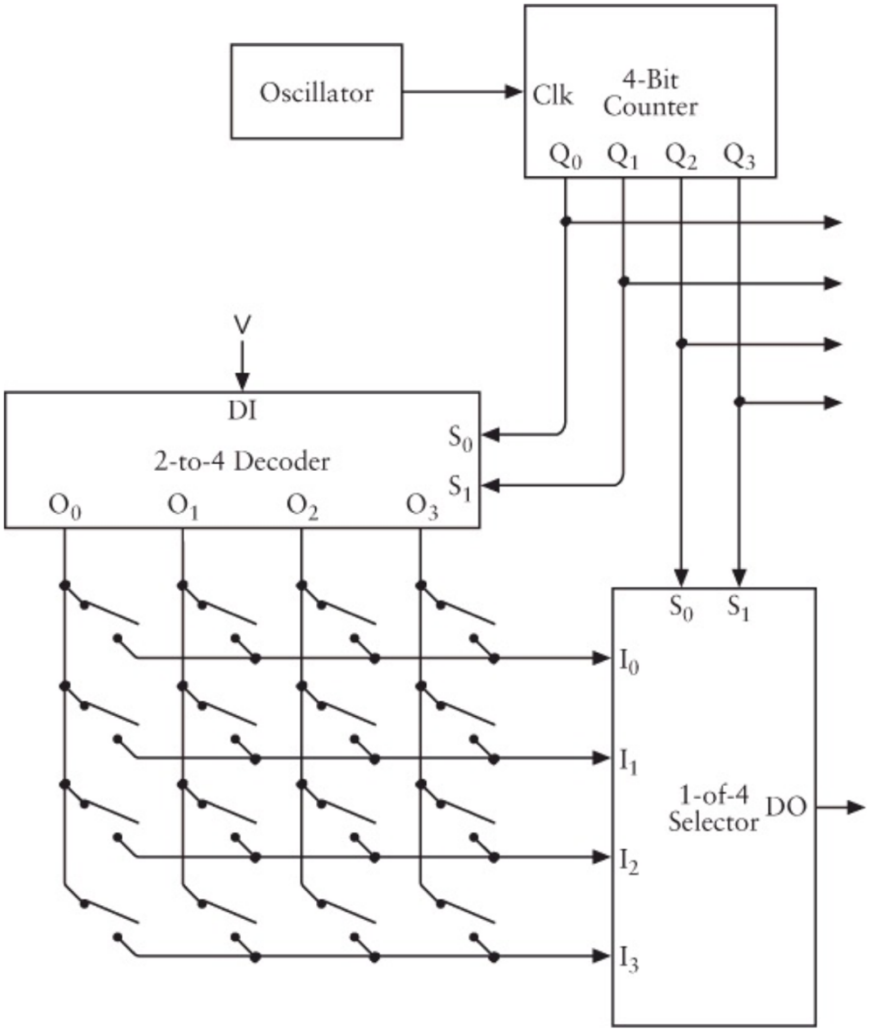
Зазвичай комп'ютерний дисплей та клавіатура здаються нам пов'язаними, оскільки символи, введені з клавіатури, відображаються на екрані. Однак насправді вони зазвичай не залежать один від одного.

Кожна клавіша на клавіатурі, насправді, простий перемикач. Якщо натиснути клавішу, перемикач замикається. Перші клавіатури нагадували друкарську машинку і складалися лише з 48 клавіш; клавіатура сучасних персональних комп'ютерів часто налічує понад 100 клавіш.

Клавіатура, підключена до комп'ютера, повинна передбачати деяке обладнання, що надає унікальний код для кожної клавіші. Ви можете припустити, що цей код є кодом ASCII, який відповідає символу на клавіші. Однак розробляти апаратні засоби, що визначають ASCII-код, непрактично та небажано. Наприклад, клавіша A може відповідати коду ASCII 41h або 61h залежно від того, чи натискається разом з нею клавіша Shift. Крім того, на сучасних комп'ютерних клавіатурах є багато клавіш, які не відповідають символам ASCII. Код, який надає апаратне забезпечення клавіатури, називається *скан-кодом.* Для визначення відповідності натиснутої клавіші ASCII-коду використовується невелика комп'ютерна програма.

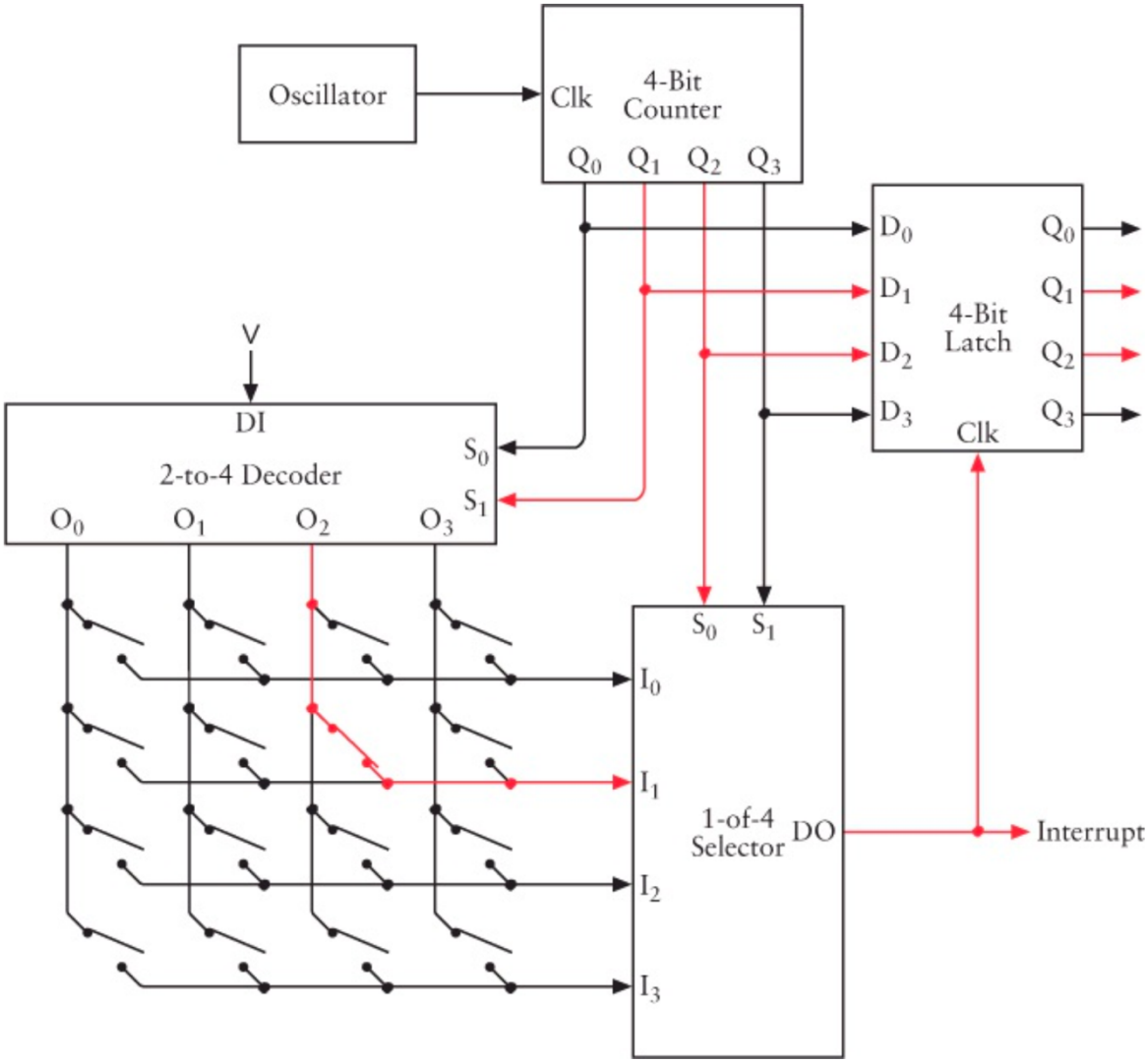
Щоб схема апаратного забезпечення клавіатури не стала занадто заплутаною, вважатимемо, що вона складається всього з 16 клавіш. При натисканні клавіші апаратне забезпечення повинне згенерувати 4-бітовий код, який приймає двійкові значення в діапазоні від 0000 до 1111.

Апаратне забезпечення клавіатури містить знайомі нам компоненти.



Шістнадцять клавіш представлені у вигляді простих перемикачів у нижній лівій частині схеми. Чотирьохбітний лічильник багаторазово і швидко перебирає 16 кодів, що відповідають клавішам. Він повинен робити це швидше, ніж людина натискає та відпускає клавішу.

Вихідні сигнали 4-бітного лічильника подаються на входи Sel дешифратора "2 на 4" та селектора "4 на 1". Якщо не була натиснута жодна клавіша, жоден із входів селектора не дорівнюватиме 1. Отже, і вихід не дорівнюватиме 1. Однак якщо натиснута конкретна клавіша, то при відповідному їй значенні вихідного сигналу 4-бітного лічильника вихід селектора дорівнюватиме 1. Наприклад, при натисканні другої зверху та праворуч клавіші та вихідному сигналі лічильника 0110 вихід селектора дорівнюватиме 1.



Це код, який відповідає цій клавіші. Коли клавіша натиснута, жодне інше значення вихідного сигналу лічильника не призведе до того, що вихід селектора дорівнюватиме 1. Для кожної клавіші передбачено власний код.

Якщо клавіатура складається з 64 клавіш, вам знадобиться 6-бітний скан-код та 6-бітний лічильник. Ви можете організувати клавіші в масив 8 × 8, використовуючи дешифратор «3 на 8» і селектор «1 на 8». Якщо клавіатура має від 65 до 128 клавіш, знадобиться 7-бітовий код. Клавіші можна організувати в масив 8 × 16 і використовувати дешифратор 4 на 16 і селектор 8 на 1 (або дешифратор 3 на 8 і селектор 16 на 1).

Те, що відбувається далі в цій схемі залежить від складності інтерфейсу клавіатури. Апаратне забезпечення може передбачати для кожної кнопки один біт оперативної пам'яті. Пам'ять RAM буде адресуватися лічильником, а вмістом цієї пам'яті може стати 0, якщо кнопка не натиснута, і 1 – якщо натиснута. Цю пам'ять RAM також може зчитувати мікропроцесор визначення стану кожної з клавіш.

Однією з корисних функцій інтерфейсу клавіатури є сигнал переривання. Як ви пам'ятаєте, мікропроцесор 8080 передбачає вхідний сигнал, який дозволяє зовнішньому пристрої переривати роботу мікропроцесора. У відповідь цей сигнал процесор зчитує команду з пам'яті. Зазвичай це команда RST, що змушує процесор перейти до певної комірки, де зберігається програма обробки переривання.

Останній периферійний пристрій, який опишу в цій темі, – пристрій довгострокового зберігання даних. Як ви пам'ятаєте, оперативна пам'ять незалежно від того, з чого вона зібрана (реле, вакуумні лампи або транзистори), втрачає свій вміст при відключенні живлення. Тому комп'ютер потребує тривалого зберігання даних. Один перевірений часом спосіб – пробивання отворів у паперових або картонних картах, на кшталт перфокарт IBM. На зорі ери невеликих комп'ютерів для збереження та подальшого завантаження програм та даних отвори пробивалися в рулонах паперової стрічки.

Недоліки перфокарт та паперової стрічки в тому, що їх не можна використовувати повторно. Пробитий отвір непросто заклеїти. Ще один дефект – невисока ефективність. Зараз, якщо ви можете *побачити* біт неозброєним оком, можете впевнено сказати: «Він займає занадто багато місця!»

З цих причин набагато частіше зустрічаються *магнітні накопичувачі.* Принцип їхньої роботи було описано ще 1878 року американським інженером Оберліном Смітом (1840–1926). Однак перший працюючий пристрій було створено лише через 20 років, у 1898 році, датським винахідником Вальдемаром Поульсеном (1869-1942). Спочатку телеграфон Поульсена призначався для запису телефонних повідомлень, якщо ніхто не міг взяти трубку. Для запису звуку на сталевому дроті застосовувався електромагніт – всюдисущий пристрій, з яким ми познайомилися, коли розглядали телеграф. Електромагніт намагнічував дріт відповідно до змін форми звукової хвилі, а для відтворення звуку дріт з тією ж швидкістю простягався вздовж обмоток електромагніту, індукуючи в них струм. Електромагніт, створений для зберігання та зчитування інформації, називається *головкою* незалежно від типу магнітного накопичувача.

У 1928 році австрійський винахідник Фріц Пфлеумер (1881-1945) запатентував магнітний записуючий пристрій, в якому використовувалася паперова стрічка з металевим напиленням, зроблена за технологією, розробленою для металізованих смужок на сигаретах. Незабаром папір замінила міцніша ацетилцелюлозна основа, завдяки чому народився один із найнадійніших і добре відомих носіїв інформації. Магнітна стрічка, тепер упакована в пластикові касети, знайшла застосування у записі та відтворенні музики та відео.

Першу комерційну систему для запису цифрових комп'ютерних даних на магнітну стрічку було представлено компанією Remington Rand 1950 року. У той час на котушці стрічки завширшки пів дюйма (1,27 сантиметра) могло поміститися кілька мегабайт. На зорі ери домашніх комп'ютерів люди перетворювали звичайні касетні магнітофони на пристрої для запису. За допомогою невеликих програм вміст блоку пам'яті записувався на стрічку, а пізніше зчитувався з неї. Перші комп'ютери IBM PC передбачали роз'єм для касетного накопичувача. Магнітна стрічка використовується для довгострокового архівування даних. Тим не менш, цей носій не ідеальний через неможливість швидкого переходу до потрібного місця на стрічці. Зазвичай для цього потрібно перемотати її вперед чи назад, а це триває деякий час.

Носієм, який забезпечує швидший доступ до даних, є диск. Сам диск обертається навколо своєї осі, поки над ним переміщається штанга з однією або декількома головками, завдяки чому доступ до будь-якої області диска здійснюється дуже швидко.

Магнітні диски фактично використовувалися для звукозапису ще до магнітної стрічки. Однак перший диск для зберігання комп'ютерних даних був винайдений у компанії IBM у 1956 році (одним із його розробників був українець Любомир Романків). Дискова система пам'яті RAMAC (Random Access Method for Accounting and Control) містила 50 металевих дисків діаметром 60 сантиметрів і могла зберігати п'ять мегабайт.

З того часу розмір дисків значно зменшився, а ємність збільшилася. Диски зазвичай поділяються на *гнучкі* (floppy, *дискети)* і *жорсткі* (hard, *незнімні диски)*. Дискета – це пластиковий диск, укладений у корпус (корпус спочатку робили картонним, потім – пластиковим). Пластиковий корпус не дає дискеті гнутися, тому вона вже не така *гнучка,* як раніше, хоча і продовжує називатися гнучким диском. Для запису та читання даних з дискети її необхідно помістити у спеціальний пристрій під назвою *флоппі-дисковод.* Діаметр перших гнучких дисків складав близько 20 сантиметрів. У перших комп'ютерах IBM PC встановлювалися гнучкі диски діаметром близько 13 сантиметрів; потім дискети діаметром близько дев'яти сантиметрів. Можливість витягувати ці гнучкі диски з дисководу дозволяє з їх допомогою переносити дані з одного комп'ютера в інший. Окрім того, на дискетах поширювалося комерційне програмне забезпечення.

Жорсткий диск зазвичай складається з кількох металевих дисків, вбудованих у дисковод. Як правило, жорсткі диски працюють швидше та вміщують більше даних, ніж дискети, проте їх неможливо витягти.

Поверхня диска розділена на концентричні кільця, які називають *доріжками.* Кожна доріжка розділена на *сектори,* які зберігають певну кількість даних, зазвичай 512 байт. Флоппі-дисковод першого комп'ютера IBM PC використовував лише одну сторону 13-сантиметрової дискети і поділяв її на 40 доріжок по вісім секторів, кожен із яких зберігав 512 байт. Таким чином, на

кожній дискеті знаходилися 163840 байт, або 160 кілобайт. Дискети 3,5 дюйми, що використовувалися в PC-сумісних комп'ютерах, мали дві сторони по 80 доріжок і 18 секторів на доріжку. Кожен сектор такої дискети зберігав 512 байт, що забезпечувало загальну ємність 1474560 байт, або 1440 кілобайт.

Місткість першого жорсткого диска, представленого 1983 року IBM у комп'ютері PC/XT, становила десять мегабайт. В 1999 менш ніж за 400 доларів можна було придбати жорсткий диск ємністю 20 гігабайт (20 *мільярдів* байт).

Як правило, дискета або жорсткий диск передбачає власний електронний інтерфейс, проте для обміну даними з процесором потрібен ще один. Найбільш популярні стандарти інтерфейсів для жорстких дисків – SCSI (Small Computer System Interface), ESDI (Enhanced Small Device Interface) та IDE (Integrated Device Electronics). Всі ці інтерфейси використовують прямий доступ до пам'яті (DMA) для того, щоб перехопити керування шиною та здійснювати обмін даними безпосередньо між оперативною пам'яттю та диском, минаючи мікропроцесор. При цьому обмін інформацією відбувається фрагментами, що відповідають розміру дискового сектора, який зазвичай дорівнює 512 байт.

Багато користувачів початківців домашніх комп'ютерів, наслухавшись розмов про мегабайти і гігабайти, починають плутати напівпровідникову оперативну пам'ять з жорстким диском. В останні роки з'явилося правило, що дозволяє уникнути плутанини у термінології. Відповідно до нього слово «пам'ять» слід використовувати тільки для позначення напівпровідникової оперативної пам'яті, а терміни «накопичувач» і «пристрій, що запам'ятовує» – для позначення всього іншого, тобто дискет, жорстких дисків і магнітної стрічки. У цій книзі я намагався дотримуватись цього принципу.

Найбільш очевидна різниця між пам'яттю та накопичувачем у тому, що пам'ять є енергозалежною: вона втрачає свій вміст при відключенні живлення. Накопичувач не залежить від живлення. Дані зберігаються на дискеті або жорсткому диску, доки користувач не зітре їх або не перезапише. Тим не менш, існує ще одна значна відмінність, яку можна помітити, тільки зрозумівши принцип роботи мікропроцесора. При подачі адресного сигналу мікропроцесор завжди звертається до пам'яті, а не до пристрою.

Переміщення даних з накопичувача у пам‘ять для їх подальшого використання мікропроцесором вимагає додаткових дій. Для цього мікропроцесор має виконати невелику програму, яка здійснює звернення до диска.

Щоб зрозуміти різницю між пам'яттю та накопичувачем, можна використати таку аналогію: пам'ять схожа на робочий стіл. Ви можете працювати з усім, що на столі. Накопичувач подібний до шафи з папками. Якщо потрібна якась папка, ви повинні встати, підійти до шафи, дістати потрібну і покласти її на стіл. Коли на вашому столі виявляється занадто багато папок, потрібно прибрати деякі з них у шафу.

Дані на диску зберігаються у вигляді так званих *файлів.* За збереження та вилучення файлів відповідає надзвичайно важлива програма – *операційна система.*

Тема 15. Фіксована крапка, плаваюча крапка

У повсякденному житті ми легко оперуємо цілими числами, дробами та відсотками одночасно. Ми купуємо півдесятка яєць, заплативши податок у розмірі 8¼ відсотка з грошей, отриманих за 2¾ години понаднормової роботи, сплаченої за тарифом, що в півтора рази перевищує звичайний. Більшість людей не мають труднощів при використанні таких величин. Почувши від статистиків про те, що «середнє американське домогосподарство складається з 2,6 людини», ми не жахаємося від думки про пов'язані з цим повсюдні каліцтва.

Проте коли справа стосується комп'ютерної пам'яті, перемикання між цілими та дробовими числами виявляється складнішим. Так, усі дані зберігаються у комп'ютерах у вигляді бітів, тобто у вигляді двійкових чисел. Проте одні види чисел виразити у бітах набагато легше, ніж інші.

Спочатку ми використовували біти для уявлення *позитивних цілих* чи *позитивних натуральних* чисел. Ми також дізналися, як за допомогою доповнення до двох можна відобразити *негативні цілі* числа, щоб спростити операцію додавання. У наведеній нижче таблиці показано, які діапазони позитивних і негативних цілих чисел (негативні числа виражені за допомогою доповнення до двох) можна зберігати в комірках пам'яті ємністю 8, 16 і 32 біт.

Number of Bits	Range of Positive Integers	Range of Two's-Complement Integers
8	0 through 255	–128 through 127
16	0 through 65,535	–32,768 through 32,767
32	0 through 4,294,967,295	–2,147,483,648 through 2,147,483,647

Однак на цьому ми зупинилися. Крім цілих чисел математики також розрізняють *раціональні* числа, які можуть бути представлені як *відношення* двох цілих чисел. Це відношення також називається *дробом.* Наприклад, дріб ¾ – раціональне число, відношення чисел 3 і 4. Це число також можна записати у вигляді *десяткового дробу:* 0,75. Десятковий дріб залишається відношенням двох чисел, у даному разі 75/100.

У Темі 7 розповідалося, що в десятковій системі числення цифри зліва від десяткового роздільника є множниками цілих *позитивних* ступенів числа 10, а цифри праворуч – множниками цілих *негативних* ступенів числа 10. В одному з прикладів я показав, що число 42 705,684 дорівнює:

4 × 10 000 +
2 × 1000+
7 × 100 +
0 × 10 +
5 × 1 +
6 ÷ 10 +
8 ÷ 100 +
4 ÷ 1000.

Зверніть увагу на знаки ділення. Потім я представив цю послідовність операцій без ділення:

4 × 10 000 +
2 × 1000+
7 × 100 +
0 × 10 +
5 × 1 +
6 × 0,1 +
8 × 0,01 +
4×0,001.

І, нарешті, відобразив це число, використовуючи степені числа 10:

$4 \times 10^{4} +$
$2 \times 10^{3} +$
$7 \times 10^{2} +$

$0 \times 10^{1} +$
$5 \times 10^{0} +$
$6 \times 10^{-1} +$
$8 \times 10^{-2} +$
$4 \times 10^{-3}.$

Існують раціональні числа, які не так легко уявити у вигляді десяткового дробу, наприклад ⅓. Якщо ви поділите 1 на 3, то отримаєте таке.

0,33333333333333333333333333333333333333333333333333333333…

І так до безкінечності. Подібний дріб називається періодичним і записується 0,(3). Незважаючи на те, що запис числа ⅓ у вигляді десяткового дробу виглядає незграбно, це число, як і раніше, є раціональним, оскільки це відношення двох цілих чисел. Ось ще один приклад: відношення 1/7 можна записати у вигляді десяткового дробу.

0,142857142857142857142857142857142857142857142857142857…

Або $0,\overline{142857.}$

Ірраціональні числа – це числа, які не можна виразити у вигляді відношення двох цілих чисел, отже, десятковий дріб триває безкінечно без повторюваних послідовностей цифр. До чисел такого типу відноситься квадратний корінь із 2.

$\sqrt{2} \approx 1,41421356237309504880168872420969807856967187537695…$

Квадратний корінь з 2 – рішення наступного рівняння алгебри:

$x^2 - 2 = 0.$

Якщо число *не є* рішенням будь-якого рівня алгебри з цілочисельними коефіцієнтами, воно називається *трансцендентним.* (Всі трансцендентні числа ірраціональні, проте не всі ірраціональні числа трансцендентні.) До трансцендентних чисел відноситься число π, що представляє відношення довжини кола до його діаметра і приблизно дорівнює наступному.

3,14159265358979328462643383279502884197169399375 11…

$$(1 + \frac{1}{n})^n$$

Ще одним трансцендентним числом є *e,* до якого прагне вираз:

при *n,* що прагне до нескінченності. Це число приблизно дорівнює наступному:

2,71828182845904523536028747135266249775724709369996…

Усі числа, про які ми говорили, тобто раціональні та ірраціональні, називаються *дійсними,* або *речовими.* Це позначення відрізняє їх від *уявних* – квадратних коренів із негативних чисел. *Комплексні* числа – це комбінації уявних і речових чисел. Незважаючи на свою назву, уявні числа *існують* і використовуються, наприклад, при вирішенні деяких складних завдань з електроніки.

Ми звикли вважати, що числовий ряд *безперервний.* Якщо ви дасте два раціональні числа, я визначу, яке число між ними. Для цього достатньо знайти їхнє середнє арифметичне. Однак, цифрові комп'ютери не можуть працювати з континуумами. Біти можуть дорівнювати або 0, або 1, між якими немає більше ніяких значень. Отже, цифрові комп'ютери можуть мати справу тільки з *дискретними* значеннями. Кількість дискретних значень, які можна представити, безпосередньо пов'язані з кількістю доступних бітів. Наприклад, в комірках ємністю 32 біти можна зберігати

позитивні цілі числа в діапазоні від 0 до 4294967295. При необхідності зберегти значення 4,5 доведеться переглянути цей підхід і діяти інакше.

Чи можна уявити дробові значення у двійковому форматі? Так можна. Ймовірно, найпростіший підхід – використання двійково-десяткового коду (BCD). Як говорилося у темі 19, кодування BCD дозволяє записати десяткові числа у двійковому форматі. Для кодування кожної десяткової цифри (0, 1, 2, 3, 4, 5, 6, 7, 8 та 9) потрібно чотири біти.

Decimal Digit	Binary Value
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Формат BCD особливо корисний у комп'ютерних програмах, які працюють із грошовими сумами. Найбільш очевидні приклади – програми для банків та страхових компаній; багато дробових чисел в них передбачають не більше двох знаків після десяткового роздільника.

Як правило, для зберігання двох BCD-цифр достатньо одного байта. Така система запису іноді називається *упакованим кодом BCD.* У такому кодуванні не використовується додаток до двох. З цієї причини у разі упакованого коду BCD для вказівки того, чи є число позитивним або негативним, зазвичай потрібно додатковий біт, що називається *знаковим бітом.* Оскільки для зберігання BCD-значення зручно виділяти цілу кількість байтів, під біт знаку зазвичай відводиться чотири або вісім біт пам'яті.

Припустимо, що сума грошей, якою має оперувати ваша комп'ютерна програма, ніколи не перевищить ±10 мільйонів доларів. Іншими словами, вам потрібні значення від -9999999,99 до 9999999,99. Можна виділити по п'ять байт пам'яті для кожної суми, що зберігається в доларах. Наприклад, число -4325120,25 можна представити за допомогою п'яти байт.

00010100 00110010 01010001 00100000 00100101

У шістнадцятковому форматі це еквівалентно наступному запису.

14h 32h 51h 20h 25h

Зверніть увагу: крайня ліва тетрада дорівнює 1, тобто число є негативним. Це знаковий біт. Якби число було позитивним, то крайня ліва тетрада дорівнювала б 0. Для представлення кожної цифри в числі потрібно по чотири біти, а прочитати їх можна безпосередньо за шістнадцятковими значеннями, оскільки вони збігаються з десятковими.

Для представлення значень в діапазоні від -9999999,99 до 9999999,99 вам знадобиться шість байт: п'ять байт для десяти цифр і ще цілий байт для знакового біта.

Такий формат запису дробових чисел називається записом *з фіксованою крапкою,* оскільки після десяткового роздільника завжди слідує певна кількість цифр, у нашому прикладі дві. Важливо: дані про положення цього роздільника не зберігаються разом із числом. Програмам, які працюють з числами у такому форматі, необхідно повідомити, де знаходиться цей роздільник. Ви можете створювати числа з будь-якою кількістю десяткових знаків, а також використовувати їх в одній комп'ютерній програмі. Однак будь-яка частина програми, яка виконує над числами арифметичні операції, повинна знати, де знаходиться десятковий роздільник.

Формат із фіксованою крапкою добре працює тільки в тому випадку, якщо ви впевнені, що числа не перевищать розміри виділених під них осередків пам'яті, що вам не потрібно збільшувати кількість десяткових знаків. Використання цього формату є зовсім недоречним у ситуаціях, коли числа можуть стати занадто великими або маленькими. Припустимо, потрібно зарезервувати область пам'яті для зберігання відстаней. Проблема в тому, що ці відстані можуть значно змінюватись. Відстань від Землі до Сонця становить 150 000 000 000 метрів, а радіус атома водню – 0,00000000005 метра. Для зберігання значень у форматі з фіксованою крапкою, що належать цьому діапазону, доведеться виділити 12 байт пам'яті.

Можливо, ми зможемо придумати більш зручний спосіб зберігання таких чисел, якщо пригадаємо, що вчені та інженери виражають числа за допомогою системи, яка називається *науковою нотацією* (*експоненційний запис)*.

Наукова нотація особливо корисна для представлення дуже великих і дуже маленьких чисел, оскільки передбачає використання ступеня числа 10, отже дозволяє обійтися без довгих рядків

490 000 000 000	$4{,}9 \times 10^{11}$
0,00000000026	$2{,}6 \times 10^{-10}$

нулів. У науковій нотації такі числа записуються так.

У цих двох прикладах числа 4,9 та 2,6 називаються *дробовою частиною,* або *мантисою* (хоча цей термін більш доречний для логарифмів). Однак я дотримуватимусь комп'ютерної термінології, називаючи цей фрагмент наукової нотації *значущою частиною числа.*

Порядок – це ступінь, до якої підноситься число 10. У першому прикладі порядок дорівнює 11, у другому -10. Порядок показує, на скільки місць було зсунуто десятковий роздільник у значній частині числа.

Існує угода, за якою значна частина числа повинна належати інтервалу від 1 (включно) до 10. Незважаючи на те, що такі числа рівні, перший варіант подання є кращим:

$$4{,}9 \times 10^{11} = 49 \times 10^{10} = 490 \times 10^{9} = 0{,}49 \times 10^{12} = 0{,}049 \times 10^{13}.$$

Така форма наукової нотації іноді називається *нормалізованою.*

Зверніть увагу: знак показника ступеня говорить лише про порядок числа, але не про те, чи воно є негативним чи позитивним. Ось як виражаються негативні числа у науковій нотації:

$-5{,}8125 \times 10^{7}$ відповідає -58 125 000;

$-5{,}8125 \times 10^{-7}$ відповідає -0,00000058125.

У комп'ютерах замість формату з фіксованою крапкою використовується *формат з плаваючою крапкою,* який ідеально підходить для зберігання малих і великих значень, оскільки ґрунтується на науковій нотації. Однак формат, що використовується в комп'ютерах, з плаваючою крапкою передбачає запис у науковій нотації *двійкових* чисел, тому нам необхідно з'ясувати, як виглядають дробові числа в двійковому форматі.

Все набагато простіше, ніж може здатися. У десятковому записі числа цифри праворуч від десяткового роздільника відповідають негативним ступеням числа 10. У двійковому записі цифри праворуч від *двійкового роздільника* (який виглядає так само, як і десятковий) відповідають негативним ступеням числа 2. Давайте перетворимо двійкове число на десяткове.

101,1101

1 × 4 +

0 × 2 +
1 × 1 +
1 ÷ 2 +
1 ÷ 4 +
0 ÷ 8 +
1 ÷ 16

Операції ділення можна замінити множенням на негативні ступені числа 2:

$1 \times 2^{2} +$
$0 \times 2^{1} +$
$1 \times 2^{0} +$
$1 \times 2^{-1} +$
$1 \times 2^{-2} +$
$0 \times 2^{-3} +$
$1 \times 2^{-4.}$

Негативні ступені числа 2 можна також розрахувати шляхом послідовного ділення 1 на 2:

1 × 4 +
0 × 2 +
1 × 1 +
1 × 0,5 +
1 × 0,25 +
0 × 0,125 +
1×0,0625.

У результаті обчислення знаходимо, що десятковий еквівалент двійкового числа 101,1101 дорівнює 5,8125.

У десятковій науковій нотації нормалізована значна частина числа має бути більшою або дорівнювати 1, але менше 10. Таким же чином у двійковій науковій нотації нормалізована значна частина числа повинна бути більшою або дорівнювати 1, але менше числа 10, яке відповідає 2 у

$$101{,}1101 \qquad 1{,}011101 \times 2^{2}$$

десятковій системі числення. Виразимо число у двійковій науковій нотації.

Цікавий наслідок цього правила: ліворуч від двійкового роздільника у нормалізованому двійковому числі з плаваючою крапкою може стояти лише 1.

Більшість сучасних комп'ютерів і програм, що використовують числа з плаваючою крапкою, застосовується стандарт, запроваджений 1985 року Інститутом інженерів електротехніки та електроніки (IEEE) і визнаний Американським національним інститутом стандартів (American National Standards Institute, AN – ANSI/IEEE Std 754-1985, *стандарт IEEE для двійкової арифметики з плаваючою крапкою.* У стислому описі цього стандарту, що займає всього 18 сторінок, добре викладено основи кодування двійкових чисел з плаваючою крапкою.

Стандарт IEEE передбачає два основних формати: число *одинарної точності,* що займає в пам'яті чотири байти, і число *подвійної точності,* що займає вісім байт.

Спочатку розглянемо число одинарної точності. Воно складається з трьох частин: один біт відводиться для знака (0 використовується для позитивного числа, а 1 – для негативного), вісім біт – для порядку, а 23 біта – для дробової значущої частини числа, в якій наймолодший біт – крайній справа.

s = 1-Bit Sign	e = 8-Bit Exponent	f = 23-Bit Significand Fraction

Разом 32 біти, або чотири байти. Оскільки у значущій частині нормалізованого двійкового числа з плаваючою крапкою ліворуч від двійкового роздільника завжди стоїть 1, відповідний біт не включається при збереженні числа у форматі IEEE. Зберігається лише 23-бітова *дробова частина.* Незважаючи на те, що для зберігання значущої частини числа використовується тільки 23 біти, вважається, що *точність* дорівнює 24 бітам. Трохи згодом ми розберемося в тому, що це означає.

Значення 8-бітного порядку знаходиться в діапазоні від 0 до 255. Такий порядок називається *зміщеним.* Для знаходження справжнього значення порядку з урахуванням знаку необхідно відняти від нього число, зване *зсувом.* Для чисел одинарної точності з плаваючою крапкою зсув порядку дорівнює 127.

Значення порядку 0 і 255 використовуються в особливих випадках, про які я розповім трохи пізніше. Якщо значення порядку належить діапазону від 1 до 254, число, представлене конкретними значеннями *s* (біт знаку), *e* (порядок) і *f* (дробова частина), дорівнює:

$$(-1)^{s} \times 1, f \times 2^{e-127.}$$

Вираз $(-1)^s$ використовується для визначення знака числа. Якщо *s* дорівнює 0, то число позитивне (оскільки будь-яке число в ступені 0 дорівнює 1), якщо *s* дорівнює 1, число негативне (оскільки -1 у степені 1 дорівнює -1).

Наступна частина виразу 1, *f* представляє 1, за якою слідує двійковий роздільник і 23-бітова дробова частина. Все це множиться на 2, піднесене в степінь, значенням якої є різниця 8-бітного зміщеного порядку, що зберігається в пам'яті, і числа 127.

Я не згадав про спосіб вираження такого поширеного числа, як 0. Схоже, про нього ми забули. Для цього передбачено декілька особливих випадків:

• якщо *e* дорівнює 0, *f* дорівнює 0, число дорівнює 0; як правило, для представлення числа 0 у всі 32 біти записуються нулі, проте біт знака може дорівнювати 1, і в цьому випадку число інтерпретується як *негативний нуль;* біт може позначати дуже мале негативне число, для представлення якого з одинарної точністю доступних цифр і порядків недостатньо;

• якщо *e* дорівнює 0, а *f* не дорівнює 0, число є дійсним, але не нормалізованим:

$$(-1)^{s} \times 0, f \times 2^{-127;}$$

зверніть увагу на 0 зліва від двійкового роздільника частини;

• якщо *e* дорівнює 255, а *f* дорівнює 0, число символізує позитивну або негативну нескінченність – залежно від знака *s;*

• якщо *e* дорівнює 255, а *f* не дорівнює 0, значення вважається «не числом» і позначається абревіатурою *NaN* (Not a Number – «не число»); NaN може вказувати на невідоме число або результат неприпустимої математичної операції.

Найменше нормалізоване позитивне або негативне двійкове число, яке можна представити з одинарною точністю у форматі з плаваючою крапкою, таке:

$$1{,}00000000000000000000000_{\text{ДВА}} \times 2^{-126.}$$

У цьому числі після двійкового роздільника слідують 23 двійкові нулі. Найбільше нормалізоване позитивне чи негативне двійкове число, яке можна представити з одинарною точністю у форматі з плаваючою крапкою, таке:

$$1{,}11111111111111111111111_{\text{ДВА}} \times 2^{127.}$$

У десятковій системі числення ці два числа приблизно дорівнюють $1{,}175494351 \times 10^{-38}$ та $3{,}402823466 \times 10^{38.}$ Саме цими числами обмежується діапазон чисел із плаваючою крапкою одинарної точності.

Ймовірно, ви пам'ятаєте, що десять двійкових цифр приблизно еквівалентні трьом десятковим цифрам. Під цим маю на увазі, що двійкове число, що складається з десяти одиниць, яке відповідає числу 3FFh у шістнадцятковому форматі і 1023 у десятковому, приблизно дорівнює числу з трьох дев'яток, тобто 999.

$$2^{10} \approx 10^{3.}$$

З цього співвідношення випливає, що 24-бітове двійкове число одинарної точності у форматі з плаваючою крапкою, приблизно еквівалентно десятковому числу, що складається з семи цифр. З цієї причини вважається, що число одинарної точності у форматі з плаваючою крапкою, має *точність* до 24 бітів, або близько семи десяткових знаків. Що це означає?

Точність числа з фіксованою крапкою очевидна. Наприклад, грошова сума, виражена у вигляді числа з фіксованою крапкою, що має два десяткові знаки, визначена з точністю до центу.

Однак про числа з плаваючою крапкою ми не можемо сказати нічого подібного. Залежно від значення порядку число з плаваючою крапкою може мати точність до часток цента або до десятків доларів.

Правильніше було б сказати, що число одинарної точності з плаваючою крапкою має точність до однієї частини з 224 (однієї частини з 16777216, приблизно до шести частин зі 100 мільйонів). Що це означає *насправді* ?

По-перше, якщо ви спробуєте виразити значення 16777216 і 16777217 у вигляді чисел одинарної точності з плаваючою крапкою, вони виявляться однаковими. Більш того, будь-яке число в проміжку між цими двома значеннями (наприклад, 16777216,5) теж буде збігатися з ними. Всі три десяткові числа зберігаються в пам'яті у вигляді 32-бітного числа одинарної точності з плаваючою крапкою, яке, будучи розділеним на біти знака, порядку та значущої частини, виглядає наступним чином.

4B800000h
0 10010111 00000000000000000000000

І воно еквівалентне

$$1{,}00000000000000000000000_{\text{ДВА}} \times 2^{24.}$$

Наступне значення, виражене двійковим числом з плаваючою крапкою, еквівалентне числу 16777218:

$$1{,}00000000000000000000001_{\text{ДВА}} \times 2^{24.}$$

Зберігання двох різних десяткових значень як однакових чисел з плаваючою крапкою не завжди створює проблеми.

Правда, якщо при написанні банківської програми ви використовуєте числа одинарної точності з плаваючою крапкою для зберігання грошових сум у доларах і центах, вас, ймовірно, турбуватиме те, що 262 144,00 долара дорівнюють 262 144,01 долара. Обидві ці суми виражаються числом:

$$1{,}00000000000000000000000_{\text{ДВА}} \times 2^{18.}$$

Це одна з причин, чому при роботі з доларами та центами краще застосовувати формат із фіксованою крапкою. При використанні чисел з плаваючою крапкою ви можете виявити інші дратівливі нюанси. Наприклад, програма, яка виконує обчислення, в результаті якого має вийти число 3,50, видає значення 3,4999999999999. Так часто буває при використанні чисел з плаваючою крапкою і з цим нічого не можна вдіяти.

Якщо ви твердо вирішили зупинитися на числах з плаваючою крапкою, але вам недостатньо одинарної точності, спробуйте застосувати числа з плаваючою крапкою *подвійної точності.* Числа у цьому форматі займають вісім байт пам'яті, розподілених в такий спосіб.

s = 1-Bit Sign	e = 11-Bit Exponent	f = 52-Bit Significand Fraction

Зсув порядку дорівнює 1023, або 3FFh, тому число в цьому форматі записується так:

$$(-1)^{s} \times 1, f \times 2^{e-1023.}$$

До нуля, нескінченності та значень NaN застосовуються правила, аналогічні тим, які ми розглядали, коли говорили про числа одинарної точності.

Найменше позитивне чи негативне число подвійної точності з плаваючою крапкою таке:

$$1{,}0000000000000000000000000000000000000000000000000000_{\text{ДВА}} \times 2^{-1022.}$$

У цьому числі після двійкового роздільника йдуть 52 нулі. Найбільше:

$$1{,}11111111111111111111111111111111111111111111111111_{\text{ДВА}} \times 2^{1023}.$$

Відповідні десяткові числа формують діапазон приблизно від $2{,}2250738585072014 \times 10^{-308}$ до $1{,}7976931348623158 \times 10^{308}$. Число 10^{308} дуже велике, воно представляє одиницю з 308 нулями.

П'ятдесятитрибітна значна частина числа (включаючи перший біт, що не враховується) приблизно еквівалентна 16 десятковим знакам. Це вже набагато краще формату з одинарною точністю, проте ймовірність того, що якесь число одного разу стане рівне іншому, як і раніше, існує. Візьмемо, наприклад, числа 140737488355328,00 і 140737488355328,01. Вони обидва зберігатимуться у вигляді 64-бітного числа подвійної точності з плаваючою крапкою:

42E0000000000000h.

У двійковому форматі це число виглядає так:

$$1{,}0000000000000000000000000000000000000000000000000000_{\text{ДВА}} \times 2^{47}.$$

Зрозуміло, розробка формату для зберігання у пам'яті чисел з плаваючою крапкою є лише невеликим етапом у процесі фактичного використання цих чисел у програмах, написаних мовою асемблера. Якби ви конструювали комп'ютер з нуля, зараз виникла б проблема створення набору функцій для додавання, віднімання, множення та ділення чисел з плаваючою крапкою. На щастя, ці завдання можна розбити на дрібніші підзавдання, пов'язані із додаванням, відніманням, множенням і діленням *цілих* чисел, розв‘язувати які ви вже навчилися.

Наприклад, додавання чисел з плаваючою крапкою зводиться до додавання їх значущих частин; складність полягає лише у порядках. Припустимо, вам необхідно виконати таку операцію додавання:

$$(1{,}1101 \times 2^{5}) + (1{,}0010 \times 2^{2}).$$

У такому разі потрібно додати числа 11101 і 10010, проте друге число необхідно перетворити з урахуванням різниці значеннях порядків. Фактично потрібно додати цілі числа 11101000 та 10010. Підсумкова сума становитиме:

$$1{,}1111010 \times 2^{5}.$$

Іноді різниця в порядках може бути такою великою, що одне із двох чисел навіть не вплине на суму. Це може статися, наприклад, при додаванні відстані від Землі до Сонця та радіусу атома водню.

Перемноження двох чисел з плаваючою крапкою зводиться до перемноження двох значущих частин як звичайних цілих чисел і додавання двох цілих значень порядків. Нормалізація значущої частини може призвести до зменшення нового порядку.

Наступний рівень складності при використанні чисел з плаваючою крапкою пов'язаний з обчисленням квадратних коренів, ступенів, логарифмів та тригонометричних функцій. Однак усі ці завдання можна вирішити за допомогою чотирьох основних операцій: додавання, віднімання, множення та ділення.

Наприклад, така тригонометрична функція, як синус, обчислюється за допомогою

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$

розкладання до ряду.

Аргумент *x* повинен виражатися в *радіанах,* де 360 ° – це 2π радіан, знак оклику – *факторіал* числа, або добуток всіх цілих чисел від 1 і до цього числа. Наприклад, 5! = 1×2×3×4×5. У цьому випадку все зводиться до простого множення. Піднесення чисельників дробів у степінь також передбачає множення. Іншими операціями є ділення, додавання та віднімання. Єдина по-

справжньому складна частина – крапка в кінці виразу, що означає, що це обчислення може продовжуватися *нескінченно.* Насправді якщо ви обмежитеся діапазоном від 0 до π/2 (з якого можна вивести всі інші значення синуса), то зможете уникнути зайвих обчислень. Вам достатньо десятка доданків у цьому розкладанні для отримання 53-бітових значень подвійної точності.

Якщо врахувати, що комп'ютери призначені для того, щоб полегшувати людям життя, може здатися, що написання безлічі підпрограм для виконання арифметичних операцій з плаваючою крапкою, суперечить меті їх створення. Однак у цьому вся принадність програмного забезпечення. Написані підпрограми для конкретного комп'ютера можуть використовуватися іншими людьми. Арифметика з плаваючою крапкою настільки важлива для наукових та інженерних додатків, що їй традиційно надається величезного значення. На зорі комп'ютерної ери при створенні програмного забезпечення для нового типу комп'ютерів написання підпрограм для виконання розрахунків з плаваючою крапкою було одним із першочергових завдань.

Доцільно розробити машинні інструкції спеціально для виконання обчислень з плаваючою крапкою! Очевидно, це легше сказати, ніж зробити, проте важливість такого завдання важко переоцінити. Якщо ви зможете реалізувати арифметику з плаваючою крапкою на рівні апаратного забезпечення, подібно до команд множення та ділення в 16-розрядних мікропроцесорах, то всі обчислення з плаваючою крапкою будуть виконуватися комп'ютером набагато швидше.

Першим комерційним комп'ютером, у якому обчислення з плаваючою крапкою могли здійснюватися на апаратному рівні, був IBM 704, випущений 1954 року. Усі числа у ньому зберігалися у вигляді 36-бітових значень. Числа з плаваючою крапкою розбивалися на 27-бітну частину, 8-бітний порядок і однобітний знак. Спеціальні апаратні компоненти для розрахунків з плаваючою крапкою могли виконувати операції додавання, віднімання, множення та ділення. Інші функції реалізовувалися у програмному забезпеченні.

У настільному комп'ютері апаратне забезпечення для обчислень з плаваючою крапкою з'явилося в 1980 році, коли компанія Intel випустила чіп 8087. Інтегральні мікросхеми такого типу в наші дні називаються *математичними співпроцесорами* або *блоками обчислень з плаваючою крапкою.* Мікросхема 8087 була названа *спів*процесором, оскільки не могла працювати сама собою. Її можна було використовувати лише у поєднанні з першими 16-розрядними мікропроцесорами Intel 8086 та 8088.

Співпроцесор 8087 – мікросхема з 40 виводами, яка використовує багато з тих же сигналів, що і мікропроцесори 8086 і 8088. За допомогою цих сигналів мікропроцесор та математичний співпроцесор взаємодіють. Коли ЦПУ зчитує спеціальну команду ESC (Escape), співпроцесор перехоплює керування та виконує наступний машинний код, який відповідає одній з 68 команд для обчислення тригонометричних функцій, ступенів, логарифмів тощо. Типи даних засновані на стандарті IEEE. Свого часу співпроцесор 8087 вважався найскладнішою інтегральною схемою.

Співпроцесор можна назвати невеликим автономним комп'ютером. У відповідь на отримання конкретної машинної інструкції для виконання операції з плаваючою крапкою (наприклад, команди FSQRT для обчислення квадратного кореня) співпроцесор виконує власну послідовність команд, збережених у ПЗП. Ці внутрішні команди називаються *мікрокодом.* Як правило, ці обчислення здійснюються за допомогою циклу, тому їхній результат надається не відразу. Проте математичний співпроцесор зазвичай вирішує завдання щонайменше вдесятеро швидше, ніж еквівалентні процедури, реалізовані як ПЗ.

Материнська плата першого комп'ютера IBM PC передбачала 40-контактне гніздо для мікросхеми 8087 поруч із процесором 8088. На жаль, це гніздо було порожнім. Користувачам, яким потрібно прискорити операції з плаваючою крапкою, доводилося придбати мікросхему 8087 окремо і самостійно встановлювати її. Однак навіть після встановлення математичного співпроцесора не всі програми починали працювати швидше. Деякі програми, наприклад текстові редактори, практично не потребують обчислень з плаваючою крапкою. Інші програми, на зразок електронних таблиць, можуть виконувати подібні обчислення набагато частіше, тому вони повинні працювати швидше, проте від використання цієї мікросхеми швидкість роботи збільшувалася далеко не у всіх програм.

Справа в тому, що програмістам потрібно було писати спеціальний код для використання машинних інструкцій співпроцесора. Оскільки математичний співпроцесор не був стандартним

компонентом апаратного забезпечення, багато хто себе цим не турбував. Зрештою їм все одно доводилося писати власні підпрограми для обчислень з плаваючою крапкою (оскільки більшість користувачів не мала математичного співпроцесора), тому мікросхема 8087 не звільнила їх від зайвої роботи, а навпаки, виявила додаткові завдання. Згодом програмісти навчилися писати програми для використання математичного співпроцесора, якщо він входив до складу комп'ютера, і емулювати його роботу, якщо не входив.

Надалі компанія Intel також випустила математичні співпроцесори 287 та 387 для мікропроцесорів 286 та 386 відповідно. У 1989 році з'явився процесор Intel 486DX, у який співпроцесор був вбудований. З того часу він перестав бути додатковим компонентом. На жаль, у 1991 році Intel сконструювала більш дешевий мікропроцесор 486SX без вбудованого співпроцесора, а також окремий математичний співпроцесор 487SX. Однак із виготовленням процесора Pentium в 1993 році вбудований співпроцесор знову став стандартом, мабуть назавжди. Компанія Motorola інтегрувала співпроцесор у мікросхему 68040 у 1990 році. До цього Motorola продавала математичні співпроцесори 68881 і 68882 для раніше виготовлених мікропроцесорів сімейства 68000. Мікросхеми PowerPC також передбачають вбудований співпроцесор для виконання обчислень з плаваючою крапкою.

Незважаючи на те, що апаратний компонент для операцій з плаваючою крапкою є підмогою для асемблерного програміста, цей пристрій здається малозначущою віхою в історії розвитку обчислювальної техніки, особливо якщо порівнювати з іншими розробками, розпочатими в 1950-х роках. Далі ми поговоримо про мови програмування.

Список використаної літератури

1. Computer Architecture / Charles Fox., No Starch Press, 2024
2. Code: The Hidden Language of Computer Hardware and Software / Charles Petzold., Microsoft Press, 2023
3. Make: Electronics, Charles Platt, Make Community, LLC, 2021. – 326 p.
4. P. Savaryn, V. Strekha, M. Brych, L. Brych, V. Kabak and M. Polishchuk, «The Original Method of Controlling a Computer Using Distance Sensors», 2022 IEEE 16th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET), 2022, pp. 683-688, doi: 10.1109/TCSET55632.2022.9767011.
5. Savaryn, P. (2023). TESLA SWITCH OF 4 BATTERIES BASED ON THE ARDUINO UNO BOARD / Polishchuk, M., Grinyuk, S., Kostiuchko, S., Tkachuk, A. // Informatyka, Automatyka, Pomiary W Gospodarce I Ochronie Środowiska, 13(3), 111-116. https://doi.org/10.35784/iapgos.4051.
6. Саварин, П., Редько, О., Редько, Р., & Великий, О. (2024). ЛЮДИНО-КОМП'ЮТЕРНА ВЗАЄМОДІЯ НА ОСНОВІ ARDUINO. Automation of Technological and Business Processes, 15(4), 98-105. https://doi.org/10.15673/atbp.v15i4.2724
7. https://udemy.com/course/crash-course-digital-electronics/
8. https://www.codehiddenlanguage.com

Кодування інформації та архітектура комп'ютера : Конспект лекцій для здобувачів першого (бакалаврського) рівня вищої освіти освітньої програми «Професійна освіта (комп'ютерні технології)» галузі знань А Освіта спеціальності А5.39 Професійна освіта (Цифрові технології) денної та заочної форм навчання / уклад. П. САВАРИН. Луцьк: ЛНТУ, 2026. – 192 с.

Комп'ютерний набір П. САВАРИН
Редактор П. САВАРИН

Підп. до друку «__»_____2026 р. Формат 60х84/16. Папір офс.
Гарн. Таймс. Ум. друк. арк. 12.
Тираж 50 прим.

Луцький національний технічний університет
43018, м. Луцьк, вул. Львівська, 75