Міністерство освіти і науки, молоді та спорту України
Чернівецький національний університет
імені Юрія Федьковича

Підлягає поверненню на кафедру

Архітектура комп'ютерів

Лабораторний практикум

Чернівці
Чернівецький національний університет
2010

ББК 32.973.26-02
А–87
УДК 004.2

Друкується за ухвалою редакційно-видавничої ради
Чернівецького національного університету
імені Юрія Федьковича

Рецензенти: Остапов С.Е., доктор фіз.-мат. наук, професор,
Ленюк М.П., доктор фіз.-мат. наук, професор.

А–87 **Архітектура** комп'ютерів : лабораторний практикум / укл. Жихаревич В.В. – Чернівці : ЧНУ, 2010. – 126 с.

У запропонованому виданні розглядаються основи цифрової електроніки як елементної бази комп'ютерних систем, особливості побудови операційних та керуючих мікропрограмних автоматів у складі мікропроцесорів, а також етапи проектування елементарних мікропроцесорних систем нейманівської архітектури. Достатня увага приділяється архітектурі мікропроцесорів родини 80x86 фірми Intel, в тому числі – особливостям їх організації та функціонуванню на рівні команд асемблера.

Для студентів вищих навчальних закладів, які навчаються за напрямком підготовки „Програмна інженерія".

ББК 32.973.26-02

УДК 004.2

ЗМІСТ

ВСТУП

Дисципліна „Архітектура комп'ютерів" є однією з головних у напрямках програмної інженерії та комп'ютерних наук. Мета викладання даної дисципліни – набуття студентами базових знань з основ побудови та функціонування комп'ютерних систем, які необхідні у подальшому навчанні, а також у практичній діяльності на виробництві. Успішне виконання лабораторних робіт та засвоєння дисципліни в цілому передбачає знання шкільного курсу інформатики.

Запропонований лабораторний практикум ілюструє визначальні положення лекційного курсу "Архітектура комп'ютерів" на конкретних прикладах проектування комбінаційних і послідовних вузлів комп'ютерної техніки, операційних та керуючих мікропрограмних автоматів, елементарних мікропроцесорних систем нейманівської архітектури. Досліджуються питання організації та функціонування мікропроцесорів, а також особливості їх взаємодії із зовнішніми пристроями на прикладі моделі 16-розрядного мікропроцесора 8086 фірми Intel.

Книга складається з 12 лабораторних робіт, має традиційну побудову та відображає такі теми, як основи цифрової електроніки, побудова запам'ятовуючих пристроїв, проектування операційних пристроїв на основі мікропрограмного керування, проектування елементарних мікропроцесорних систем нейманівської архітектури та дослідження моделей актуальних на сьогоднішній день мікропроцесорів родини 80x86 фірми Intel. Методичні вказівки містять багато ілюстрацій та прикладів. Суть лабораторних занять полягає у складанні схем цифрової електроніки у системі схемотехнічного моделювання Proteus із можливістю використання моделі мікропроцесора 8086 і розробці для нього невеликих програм мовою асемблера, які ілюструють окремі аспекти організації та функціонування комп'ютерних систем на низькому рівні.

Методичні вказівки призначені для студентів спеціальності „Програмне забезпечення автоматизованих систем" напрямку підготовки „Програмна інженерія".

Лабораторна робота № 1
Дослідження роботи базових логічних елементів та основних комбінаційних вузлів комп'ютерної техніки

Теоретична частина

Побудова будь-яких цифрових пристроїв заснована на принципі багатократного повторення відносно простих базових логічних схем. Зв'язки між цими схемами будуються на основі чисто формальних методів. Інструментом такої побудови служить булева алгебра, яка стосовно цифрової техніки називається також алгеброю логіки.

На відміну від змінної у звичайній алгебрі, логічна змінна має тільки два значення, які зазвичай називаються логічним нулем та логічною одиницею. Як позначення використовуються „0" та „1" або просто 0 та 1.

Існують три основні операції між логічними змінними: кон'юнкція (логічне множення), диз'юнкція (логічне додавання) та інверсія (логічне заперечення). За аналогією з алгеброю чисел в алгебрі логіки використовуються такі позначення операцій.

Кон'юнкція: $y = x_1 \wedge x_2 = x_1 \cdot x_2 = x_1 x_2$.

Диз'юнкція: $y = x_1 \vee x_2 = x_1 + x_2$.

Інверсія: $y = \overline{x}$.

В таблиці 1.1 подано функції для кон'юнкції та диз'юнкції.

Таблиця 1.1

Таблиця істинності для логічного множення (кон'юнкції) $y = x_1 \cdot x_2$

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

Таблиця істинності для логічного додавання (диз'юнкції) $y = x_1 + x_2$

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

З таблиці 1.1 видно, що у випадку логічного множення y тільки тоді дорівнює 1, коли x_1 *і* x_2 дорівнюють 1. З цієї

причини операція кон'юнкції називається також функцією І (англ. AND). При диз'юнкції двох змінних y дорівнює 1 тоді, коли x_1 *або* x_2 дорівнюють 1. Тому операцію диз'юнкції називають також функцією АБО (англ. OR). Обидві ці функції можна розповсюдити на будь-яку кількість змінних. Якщо вихідні значення y зазначених функцій інвертувати, то отримуємо, відповідно, функції І-НЕ (англ. NOT AND або NAND) та АБО-НЕ (англ. NOT OR або NOR). Якщо у функції АБО інвертувати лише значення при x_1=1 та x_2=1, то отримаємо так звану функцію ВИКЛЮЧНЕ АБО, тобто функцію нерівнозначності (англ. eXclusive OR або XOR), при інверсії якої отримаємо функцію ВИКЛЮЧНЕ АБО-НЕ, тобто функцію рівнозначності (англ. NOT eXclusive OR або NXOR).

У цифрових електронних схемах можливістю представлення логічних змінних є електрична напруга, яка може набувати два різних рівня: високий та низький. Цим рівням можна поставити у відповідність логічні стани 1 та 0.

Основні логічні функції можуть бути реалізовані за допомогою відповідних електронних схем. Ці схеми мають один або декілька входів і один вихід, причому здебільшого логічні схеми аналізуються без врахування їх внутрішньої структури. Справедливість такого підходу випливає з того, що в сучасній цифровій техніці майже завжди використовуються логічні елементи, які, крім виводів живлення, містять тільки розглянуті логічні входи і виходи.

Для реалізації основних логічних функцій існує ряд різних схем, які відрізняються споживаною потужністю, напругою живлення, значенням високого та низького рівнів вихідної напруги, часом затримки розповсюдження сигналу та навантажувальною здатністю. Серед прикладів схемної реалізації основних логічних функцій можна виділити елементи [13]: Резистор-Діодної Логіки (РДЛ), Резистор-Транзисторної Логіки (РТЛ), Діодно-Транзисторної Логіки (ДТЛ), Транзистор-Транзисторної Логіки (ТТЛ), Транзистор-Транзисторної Логіки з діодами Шоткі (ТТЛШ), Емітерно-Зв'язаної Логіки (ЕЗЛ),

Інтегрально-Інжекційної Логіки (ІІЛ або $І^2Л$), МДН-транзисторної Логіки (МДНЛ), Логіки на Комплементарних МДН-транзисторах (КМДНЛ).

Суматори двійкових чисел

Суматором називається функціональний вузол комп'ютера, призначений для додавання двох *n*-розрядних слів (чисел). Операція віднімання замінюється додаванням слів в оберненому або доповняльному коді. Операції множення та ділення зводяться до реалізації багаторазового додавання та зсуву. Тому суматор є важливою частиною арифметико-логічного пристрою.

Суматор складається з окремих схем, які називаються однорозрядними суматорами. Вони виконують усі дії з додавання значень однойменних розрядів двох чисел (операндів). Суматори класифікуються за такими ознаками:

- способом додавання – паралельні, послідовні та паралельно-послідовні;
- кількістю входів – напівсуматори, однорозрядні та багаторозрядні суматори;
- організацією зберігання результату додавання – комбінаційні, накопичувальні, комбіновані;
- організацією перенесення між розрядами – з послідовним, наскрізним, паралельним або комбінованим перенесеннями (з груповою структурою);
- системою числення – позиційні (двійкові, двійково-десяткові) та непозиційні, наприклад, у системі залишкових класів;
- розрядністю (довжиною) операндів – 8-, 16-, 32-, 64-розрядні;
- способом подання від'ємних чисел – в оберненому або доповняльному кодах, а також в їхніх модифікаціях;
- часом додавання – синхронні, асинхронні.

У паралельних n-розрядних суматорах значення всіх розрядів операндів надходять одночасно на відповідні входи однорозрядних підсумовуючих схем. У послідовних суматорах значення розрядів операндів та перенесення, що запам'ятовувалися в минулому такті, надходять послідовно в напрямку від молодших розрядів до старших на входи одного однорозрядного суматора. В паралельно-послідовних суматорах

числа розбиваються на частини, наприклад, байти, розряди байтів надходять на входи восьмирозрядного суматора паралельно (одночасно), а самі байти – послідовно, в напрямку від молодших до старших байтів з урахуванням запам'ятованого перенесення.

У комбінаційних суматорах результат операції додавання запам'ятовується в регістрі результату. В накопичувальних суматорах процес додавання поєднується зі зберіганням результату. Це пояснюється використанням Т-тригерів як однорозрядних схем додавання.

Організація перенесення практично визначає час виконання операції додавання. Послідовні перенесення схемно створюються просто, але є повільнодіючими. Паралельні перенесення схемно організуються значно складніше, але дають високу швидкодію.

Розрядність суматорів знаходиться в широких межах: 4-16 – для мікро- та міні-комп'ютерів та 32-64 і більше – для універсальних машин.

Суматори з постійним інтервалом часу для додавання називаються синхронними. Суматори, в яких інтервал часу для додавання визначається моментом фактичного закінчення операції, називаються асинхронними. В асинхронних суматорах є спеціальні схеми, які визначають фактичний момент закінчення додавання і повідомляють про це в пристрій керування. На практиці переважно використовуються синхронні суматори.

Мультиплексори і демультиплексори

У багатьох випадках виникає необхідність послідовного опитування логічних станів великої кількості змінних і передачі їх на один вихід. Для цієї мети служить мультиплексор (колектор). Мультиплексором називається функціональний вузол комп'ютера, призначений для почергової комутації (перемикання) інформації від одного з n входів на загальний вихід. Номер конкретної вхідної лінії, що підключається до виходу в кожний такт машинного часу, визначається адресним кодом a_0, a_1, ..., a_{m-1}. Зв'язок між кількістю інформаційних n і адресних m входів визначається співвідношенням $n = 2^m$. Таким

чином, мультиплексор реалізує керовану передачу даних від кількох вхідних ліній в одну вихідну.

Іноді виникає необхідність у використанні схеми, яка б виконувала зворотну задачу – задачу розподілення одного вхідного сигналу по декількох різних виходах. Подібна схема називається демультиплексором (селектором). Логіка вибору конкретної вихідної лінії, що підключається до входу в демультиплексорі, аналогічна вибору в мультиплексорі, причому, як і у попередньому випадку, справедливе співвідношення $n = 2^m$, але тут n – це кількість інформаційних виходів.

Шифратори

Шифратором називається функціональний вузол комп'ютера, призначений для перетворення вхідного n-розрядного унітарного коду у вихідний m-розрядний двійковий позиційний код. Двійкові шифратори виконують функцію, обернену функції дешифратора. При активізації однієї з вхідних ліній дешифратора на його виходах формується код, який відображає номер активного входу. Повний двійковий шифратор має $n = 2^m$ входів і m виходів.

У цифрових пристроях шифратори використовуються для таких операцій: перетворення унітарного вхідного коду у вихідний двійковий позиційний код; введення десяткових даних із клавіатури; вказування на старшу одиницю в слові; передачі інформації між різними пристроями при обмеженій кількості ліній зв'язку тощо.

Одне з основних застосувань шифратора – введення даних із клавіатури, наприклад десяткових цифр. Натискання клавіші з десятковою цифрою 0, 1, ..., 9 мають приводити до передачі в цифровий пристрій двійково-деяткового коду цієї цифри. Для цього використовується неповний шифратор „з 10 в 4".

Шифратори, які при одночасному натисканні кількох клавіш виробляють код тільки старшої цифри, називаються пріоритетними. Пріоритетні шифратори, які призначені для пошуку старшої (лівої) одиниці в слові та формування на виході двійкового номера шуканого розряду, називаються покажчиками

старшої одиниці. Їх застосовують у пристроях нормалізації чисел із плаваючою комою, в системах із пріоритетним обслуговуванням запитів на переривання роботи комп'ютера, у паралельних аналогово-цифрових перетворювачах тощо.

Схеми порівняння (компаратори)

Схемою порівняння (компаратором) називається функціональний вузол комп'ютера, призначений для вироблення ознак відношень між двійковими словами (числами). Функція F_i, що задає результат відношення, набуває одиничне значення, якщо відношення виконується, тобто істинне, і нульове, якщо відношення не виконується, тобто помилкове. Основними відношеннями вважаються: „дорівнює" $F_{A=B}$, „більше" $F_{A>B}$ і „менше" $F_{A<B}$. Маючи у своєму розпорядженні основні ознаки відношень, можна на їхній основі отримати ряд додаткових ознак, наприклад: $F_{A\neq B}=\overline{F}_{A=B}$; $F_{A\leq B}=\overline{F}_{A>B}$; $F_{A\leq B}=F_{A=B}\vee F_{A<B}$ тощо.

Ознаки відношення використовуються як логічні умови (повідомляючи сигнали) в мікропрограмах, командах передачі керування, а також у пристроях контролю і діагностики. Після виконання кожної команди в комп'ютері автоматично формуються ознаки результатів операції. Ці ознаки, які називають прапорцями, записуються у спеціальний регістр прапорців. До прапорців зазвичай відносять ознаки нульового результату, переповнення розрядної сітки, знак результату, наявність перенесень із старшого розряду суматора, парну або непарну кількість одиниць у результаті тощо.

Практична частина

Складіть схему, яка містить готові моделі основних логічних елементів NOT, AND, OR, NAND, NOR та XOR, зображену на рис. 1.1, та дослідіть їх роботу за допомогою цифрового аналізатора. При подачі на входи логічних елементів за допомогою генераторів імпульсів усіх можливих комбінацій логічних сигналів, отримаєте часові діаграми вихідних сигналів, які відповідають таблицям істинності відповідних елементів.

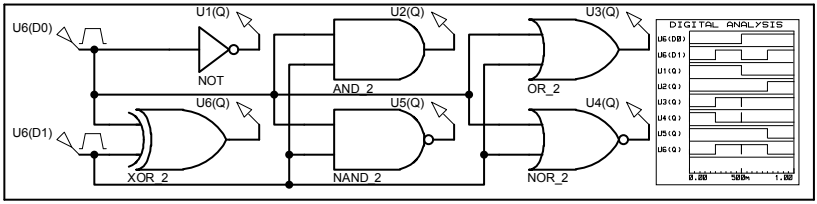


Рис. 1.1. Схема дослідження базових логічних елементів

Як доводиться алгеброю логіки, використовуючи розглянуті базові логічні елементи І, АБО та НЕ, можна побудувати будь-яку іншу логічну функцію. Зокрема, функція XOR може бути зображена у вигляді $x_1 \oplus x_2 = \overline{x_1 \cdot x_2 + \overline{x_1} \cdot \overline{x_2}}$. Складіть схему, яка реалізує дану логічну функцію, як зображено на рис. 1.2, та переконайтеся, шляхом аналізу часової діаграми роботи, в її повній еквівалентності функції XOR.

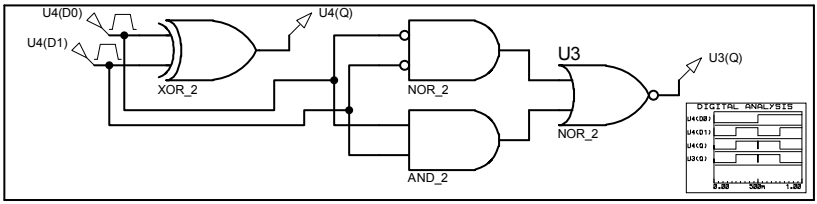


Рис. 1.2. Зображення функції XOR за допомогою інших функцій

Складіть схему однорозрядного повного суматора, зображеного на рис. 1.3. Дана логічна схема виконує додавання значень *i*-х розрядів двійкових чисел a_i та b_i з урахуванням перенесення з молодшого сусіднього розряду c_i та виробляє на виходах результат додавання одного розряду s_i і перенесення в старший сусідній розряд c_{i+1}. На основі однорозрядних схем додавання з трьома входами та двома виходами (однорозрядний повний суматор) будуються багаторозрядні суматори будь-якого типу. Алгоритм роботи однорозрядного суматора відображається таблицею істинності 1.2.

Таблиця 1.2

Таблиця істинності схеми повного суматора

Вхід			Вихід	
a_i	b_i	c_i	s_i	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Якщо подати дану таблицю істинності у вигляді булевих функцій із мінімальною кількістю логічних операцій, то їх вигляд буде таким:

$$s_i = a_i \oplus b_i \oplus c_i; \qquad c_{i+1} = a_i b_i + c_i (a_i \oplus b_i). \qquad (1.1)$$

Отже, при побудові схеми однорозрядного повного суматора можна використати по два логічних елементи І та ВИКЛЮЧНЕ АБО, які утворюють дві схеми напівсуматорів, а також один елемент АБО, як показано на рис. 1.3.

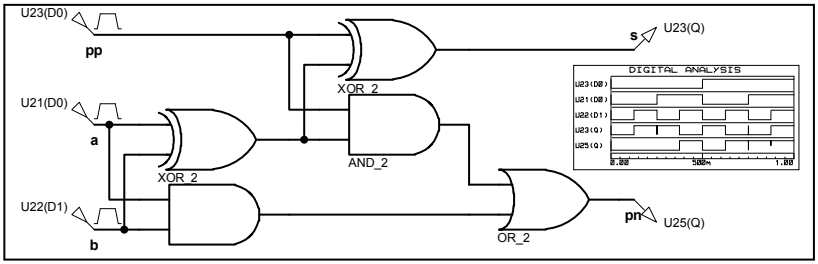


Рис. 1.3. Схема однорозрядного повного суматора

На основі схеми однорозрядного повного суматора складіть схему чотирьохрозрядного паралельного суматора з послідовним перенесенням, яка зображена на рис. 1.4. Оскільки для додавання двох багаторозрядних двійкових чисел на кожний розряд необхідний повний суматор, то слід використати відповідно чотири однорозрядних суматори. На вхід перенесення у

нульовому розряді числа необхідно подати сигнал логічного нуля. На рис. 1.4 наведено результат роботи схеми – додавання двох чисел $F_{(16)} + F_{(16)} = 1E_{(16)}$ (або у двійковій та десятковій системі числення відповідно: $1111_{(2)} + 1111_{(2)} = 11110_{(2)}$; $15_{(10)} + 15_{(10)} = 30_{(10)}$), що висвітлюється набором семисегментних світлодіодних індикаторів. Для запобігання втраті старшого розряду при додаванні вводиться додатковий індикатор, з'єднаний із розрядом перенесення. Таким чином, переповнення розрядної сітки, а отже, й виведення неправильного результату в даній схемі неможливе.

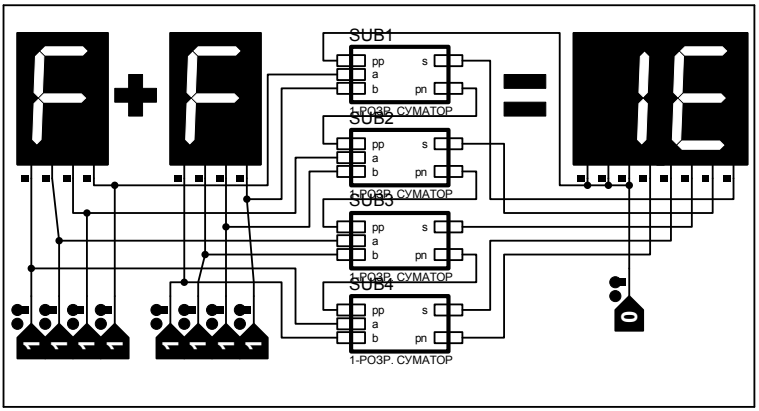


Рис. 1.4. Схема чотирьохрозрядного паралельного суматора

Складіть схему мультиплексора з чотирма інформаційними входами, як показано на рис. 1.5. В залежності від стану входів адреси a_0 і a_1 вихід мультиплексора y з'єднується з одним з його інформаційних входів $d_0 - d_3$. Схема побудована в такий спосіб, що з виходом з'єднується саме той вхід, індекс якого дорівнює двійковому числу, яке визначається змінними a_0 і a_1. Безпосередньо з рис. 1.5 випливає, що:

$$y = \overline{a}_1\overline{a}_0 d_0 + \overline{a}_1 a_0 d_1 + a_1\overline{a}_0 d_2 + a_1 a_0 d_3 . \qquad (1.2)$$

Відмітимо, що логічне множення адресних сигналів дорівнює одиниці тільки для тої вхідної змінної, індекс якої збігається з

необхідною адресою. Наприклад, якщо $a_1 = 1$ і $a_0 = 0$, то: $y = 0 \cdot 1 \cdot d_0 + 0 \cdot 0 \cdot d_1 + 1 \cdot 1 \cdot d_2 + 1 \cdot 0 \cdot d_3 = d_2$.

За вказаним принципом ця схема може бути розповсюджена на будь-яку кількість вхідних змінних. Як уже зазначалося, за допомогою m адресних входів можна вибирати один з 2^m інформаційних сигналів.

Оскільки кожній адресі відповідає тільки один інформаційний вхід, то за допомогою мультиплексора можна реалізувати будь-які логічні функції адресних сигналів. Для цього на інформаційні входи подаються постійні сигнали, які відповідають необхідним значенням логічної функції. При цьому схема працює точно так, як однорозрядний постійний запам'ятовуючий пристрій.

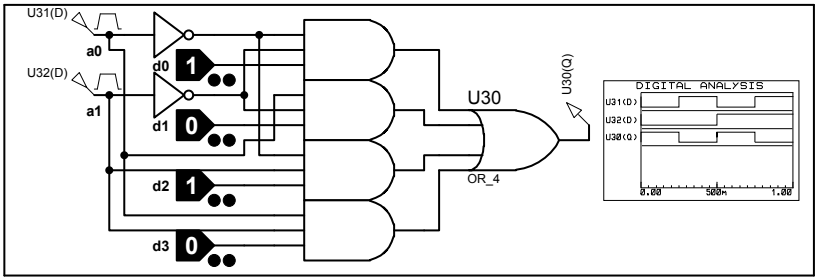


Рис. 1.5. Схема мультиплексора

Складіть схему демультиплексора з чотирма інформаційними виходами, як показано на рис. 1.6. Сигнал d подається на інформаційний вхід. Схема підключає його саме до того виходу, номер якого заданий адресними сигналами a_0 і a_1. Логіка вибору адреси тут така ж, як і у мультиплексора, зображеного на рис. 1.5. При $d = 1 = const$ демультиплексор працює як звичайний дешифратор „один з n".

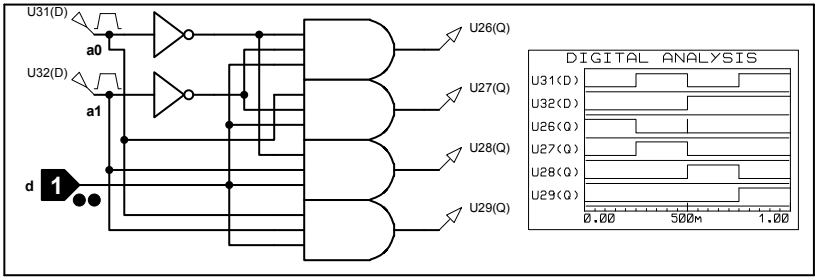


Рис. 1.6. Схема демультиплексора

Складіть схему пріоритетного шифратора „з 4 в 2”, зображену на рис. 1.7, та дослідіть особливості її роботи за допомогою цифрового аналізатора. Структура схеми випливає з таблиці істинності її роботи 1.3.

Таблиця 1.3

Таблиця істинності схеми пріоритетного шифратора „з 4 в 2”

Вхід				Вихід	
d_3	d_2	d_1	d_0	Q_1	Q_0
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	x	0	1
0	1	x	x	1	0
1	x	x	x	1	1

В даній таблиці позначкою „x” відмічено випадки, коли значення відповідних входів можуть набувати як одиничні, так і нульові значення. З таблиці видно також, що від входу d_0 значення на виході шифратора не залежать. Зобразимо таблицю істинності у вигляді булевих функцій:

$$Q_1 = d_2 + d_3;\ Q_0 = d_1\overline{d_2} + d_3. \qquad (1.3)$$

З формул (1.3) також видно, що булеві функції, які описують функціонування пріоритетного шифратора, не залежать від d_0.

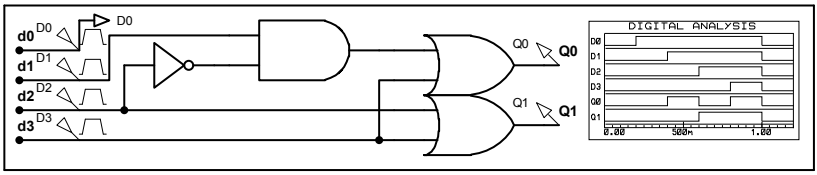


Рис. 1.7. Схема пріоритетного шифратора

Якщо в таблиці істинності схеми пріоритетного шифратора „з 4 в 2” замість „х” взяти „0”, тобто якщо забезпечена наявність лише однієї логічної одиниці на входах під час функціонування шифратора, то відповідні булеві функції та схема шифратора дещо спрощуються, що видно з рис. 1.8.

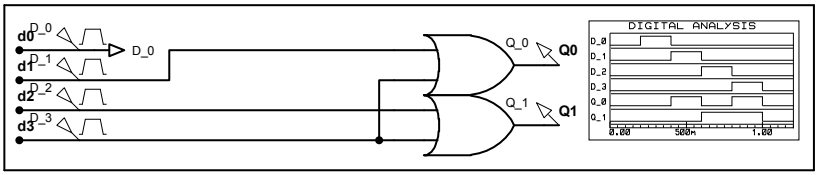


Рис. 1.8. Схема шифратора

Складіть схему чотирирозрядного компаратора для порівняння двох слів A і B, як це показано на рис. 1.9. Очевидно, що двійкові слова рівні, коли одночасно попарно рівні всі їхні розряди, тобто $A(n) = B(n)$, якщо $a_i = b_i$, $i = 1, 2, ..., n$. Порозрядний аналіз можна проводити, використовуючи логічну функцію ВИКЛЮЧНЕ АБО (функцію нерівнозначності). Таким чином, ознака рівності двох n-розрядних слів може бути визначена інвертованим значенням логічного додавання порозрядних умов нерівності, що і реалізовано на рис. 1.9.

Окрім такого підходу, порівнювати можна шляхом віднімання двох слів за допомогою суматора. Отримання нульового результату вказує на відношення „дорівнює”, а аналізуючи знак отриманого результату, можна встановити відношення „більше” або „менше”.

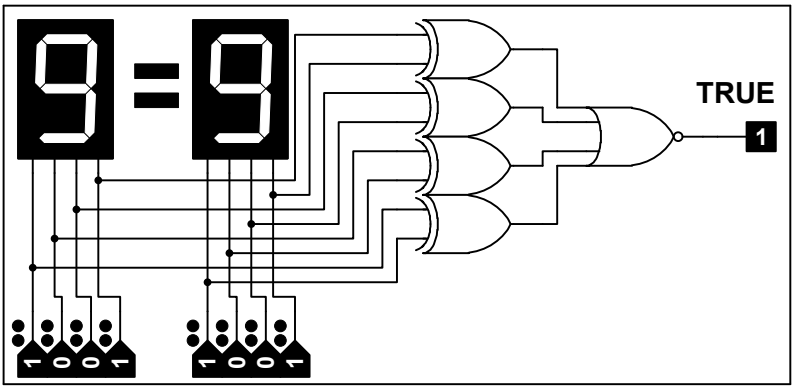


Рис. 1.9. Схема отримання ознаки рівності двійкових чисел

Проаналізуйте роботу готових моделей основних комбінаційних вузлів комп'ютерної техніки, які зображено на рис. 1.10 (A – 4-розрядна функціональна схема (FUNCTION); B – мультиплексор (SELECTOR) із 3-розрядним адресним та 8-розрядним інформаційним входом; C – демультиплексор або дешифратор (DECODER) із 3-розрядним адресним входом та 8-розрядним інформаційним виходом; D – шифратор „з 8 в 3" (ENCODER); E – 4-розрядний компаратор (COMPARATOR)).

Функціональна схема (FUNCTION) виконує роль суматора, якщо на відповідний вхід ADD (ADDITION – додавання) подати сигнал логічної одиниці.

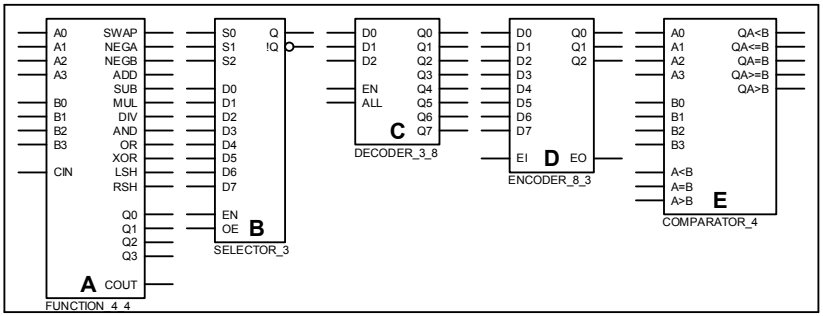


Рис. 1.10. Моделі основних комбінаційних вузлів

Контрольні запитання та завдання

1. Що являють собою базові логічні елементи? Опишіть функціонування логічних елементів AND, OR, NOT, XOR. Складіть таблиці істинності, які відображають логіку роботи базових логічних елементів.
2. Що називають алгеброю логіки (булевою алгеброю)? Які закони алгебри логіки ви знаєте?
3. Що називають суматорами двійкових чисел? Опишіть процес синтезу однорозрядного повного суматора. Де використовуються суматори? Наведіть приклад додавання двох довільних 4-розрядних двійкових цілих чисел. Як можна здійснювати нарощування розрядності суматорів?
4. Опишіть функціонування мультиплексора. Поясніть вигляд булевої функції, яка відображає логіку роботи мультиплексора. Які функції можуть виконувати мультиплексори? В який спосіб мультиплексор може реалізувати довільну булеву функцію? Як можна нарощувати розрядність мультиплексорів?
5. Опишіть функціонування демультиплексора. У чому полягає різниця між демультиплексором та дешифратором? Де використовуються дешифратори? Як можна здійснювати нарощування розрядності демультиплексорів?
6. Опишіть функціонування шифратора. Де використовуються шифратори? Що таке пріоритетний шифратор? Як можна нарощувати розрядність шифраторів?
7. Опишіть функціонування компаратора. Де використовуються компаратори? Спроектуйте схему компаратора, яка б виконувала порівняння двох 4-розрядних двійкових цілих чисел та визначала між ними відношення: „дорівнює” $F_{A=B}$, „більше” $F_{A>B}$ і „менше” $F_{A<B}$. В який спосіб можна здійснити нарощування розрядності компараторів?

Лабораторна робота № 2
Дослідження роботи елементарних запам'ятовуючих пристроїв – тригерів та основних послідовних вузлів комп'ютерної техніки

Теоретична частина

Переважна більшість цифрових пристроїв поєднують функції по переробці інформації із функцією зберігання. Невід'ємною частиною таких пристроїв є елемент пам'яті. В цифрових пристроях для зберігання інформації частіше за все використовується елемент з двома стійкими станами – тригер. Структуру тригера можна зобразити у вигляді запам'ятовуючої комірки (ЗК) та схеми керування (СК) (рис. 2.1).

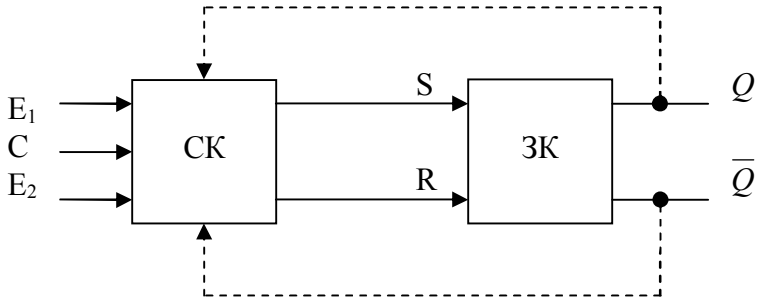


Рис. 2.1. Узагальнена структура тригерної схеми (СК – схема керування, ЗК – запам'ятовуюча комірка, E_1, E_2 – логічні входи, C – вхід синхронізації, S – вхід встановлення в „1", R – вхід скидання в „0")

Запам'ятовуюча комірка – це схема, яка має два виходи: Q і $\overline{Q}$, дозволені сигнали на яких завжди протилежні, і два входи: S (Set – встановлення) і R (Reset – скидання). Сигнал переключення по входу S встановлює запам'ятовуючу комірку у стан „1" (Q = 1, $\overline{Q}$ = 0), а по входу R – у стан „0" (Q = 0, $\overline{Q}$ = 1). В загальному випадку запам'ятовуюча комірка може мати кілька установочних входів. Зазвичай вона складається з двох логічних елементів, які взаємно охоплені зворотним зв'язком.

Запам'ятовуючу комірку називають асинхронним (несинхронізованим) RS-тригером. Така назва пояснюється способом запису інформації. Стан асинхронного тригера визначається в кожен момент часу станом входів, тобто зміни вхідного сигналу безпосередньо передаються на вихід тригера.

Схема керування перетворює інформацію, яка надходить на входи E_1, E_2, у сигнали, які подаються на встановлювальні входи запам'ятовуючої комірки. В деяких схемах вихідні сигнали тригера надходять на вхід схеми керування (на рис. 2.1 цей зв'язок показано пунктиром).

Тригери, які використовуються у послідовних схемах, мають, як правило, ще один вхід – вхід для синхронізуючих сигналів. Імпульси, які надходять на нього, визначають момент прийому тригером вхідної інформації. У зв'язку з цим на узагальненій структурі тригера (рис. 2.1) показано ще й вхід синхронізації С. Оскільки прийом вхідної інформації визначається моментом подачі імпульсу на вхід синхронізації С, то такий тригер називають синхронним тригером.

Синхронні тригери в залежності від організації схеми керування поділяють на:

1) синхронні тригери зі статичним керуванням записом;
2) синхронні двоступеневі тригери (MS-тригери);
3) синхронні тригери з динамічним керуванням записом.

У синхронних тригерах зі статичним керуванням записом вхідна інформація сприймається лише тоді, коли значення синхронного сигналу відповідає одиниці, тобто протягом всієї тривалості сигналу синхронізації. Це означає, що при С = 1 переключення сигналів на логічних входах викликає зміну стану тригера, тобто значення на виході Q. У зв'язку з цим інформацію на логічних входах тригера, як правило, змінюють при С = 0.

Синхронний двоступеневий тригер – це тригерна схема, яка складається з двох частин, тобто з двох послідовно з'єднаних тригерів зі статичним керуванням записом, одночасний прийом інформації в які заборонений. Це досягається шляхом включення інвертора в коло синхронізуючих імпульсів до одного з тригерів

(зазвичай другого). Такі схеми називають схемами, які працюють за принципом „ведучий-ведений” („Master-Slave”), а самі тригери називають MS-тригерами.

Логіка роботи синхронного двоступеневого тригера така. При C = 0 прийом вхідної інформації у перший тригер закрито. А оскільки на синхронізуючий вхід другого тригера завдяки інвертору надходить одиничний сигнал, то прийом інформації у другий тригер відкритий. Другий тригер приймає (копіює) інформацію, яка зберігається у першому тригері. Отже, при C = 0 стан обох частин двоступеневого тригера однаковий. В цьому стані можна змінювати сигнали на логічних входах без зміни стану двоступеневого тригера. При C = 1 картина змінюється: перший тригер відкритий, а другий, завдяки інвертору, закритий по синхронізуючому входу. Інформація, яка знаходиться на логічних входах двоступеневого тригера, приймається у першу частину, а друга частина залишається незмінною.

Отже, при зміні „0” → „1” синхросигналу вхідна інформація приймається у першу частину, а при зміні „1” → „0” синхросигналу новий стан першої частини передається у другу частину, тобто з’являється на виході Q двоступеневого тригера.

Тригер, в якому відсутня наскрізна передача сигналу із входу на вихід, також можна отримати, блокуючи входи в той момент, коли зчитувана інформація передається на вихід. Подібні схеми називаються тригерами з динамічним впливом по входу синхронізації або просто динамічними тригерами. Пари цьому розрізняють два типи таких тригерів: тригери, в яких передача інформації відбувається на передньому фронті тактового імпульсу, та тригери, в яких передача інформації здійснюється на задньому фронті цього імпульсу. Серед цих тригерів найбільш розповсюджений D-тригер, однак є і JK-тригери такого типу.

Синхронні тригери з динамічним керуванням записом володіють більш високою у порівнянні з двоступеневими тригерами швидкодією. Дані тригери також нечутливі до хибних сигналів на логічних входах як при високому, так і при низькому рівнях сигналу на вході синхронізації C.

Для правильної роботи цього типу тригерів сигнали на логічні входи повинні подаватися перед приходом активного фронту синхроімпульсу (час попереднього встановлення). При цьому необхідно витримувати їх незмінними ще деякий час (час утримування) після закінчення цього фронту синхроімпульсу. Порушення даних вимог призводить до непередбачуваного переключення тригера.

Двійкові лічильники

Досить важливе використання логічних схем для рахування імпульсів. Як лічильник можна використовувати довільну схему, встановивши для неї в певних межах однозначну відповідність між кількістю імпульсів, які надійшли, та станом вихідних сигналів. Оскільки кожен вихідний сигнал може набувати лише два значення, то для n вихідних сигналів існує 2^n можливих станів. Часто використовується лише частина з них. Узагалі відповідність між кількістю вхідних імпульсів та вихідним кодом може бути довільною. Однак у лічильниках раціональним є вибір такого подання чисел, з яким легко оперувати в подальшому. Для простих схем надають перевагу двійковому зображенню чисел.

Нижче наведена таблиця 2.1 відповідності між кількістю вхідних імпульсів Φ і значеннями вихідних змінних z_i для 4-розрядного двійкового лічильника. Розглядаючи цю таблицю зверху вниз, можна відмітити дві закономірності:

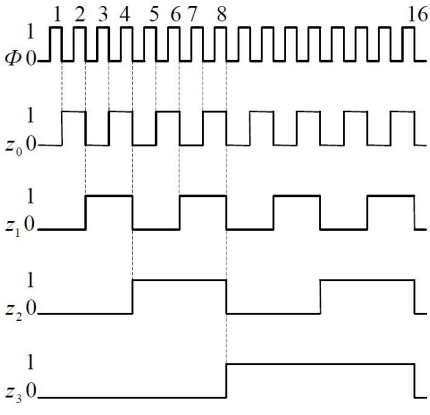
1. Значення змінної z_i змінюється тоді, коли змінна в сусідньому молодшому розряді z_{i-1} переходить зі стану „1” в стан „0”.
2. Значення вихідної змінної z_i змінюється при черговому імпульсі лічби в тому випадку, коли змінні у всіх молодших розрядах z_{i-1}, ..., z_0 знаходяться в стані „1”.

Ці ж висновки можна зробити і при розгляданні часової діаграми на рис. 2.2. Перша закономірність вказує на можливість реалізації лічильника асинхронного типу, друга дозволяє побудувати синхронний лічильник. Іноді необхідний лічильник, в якому при кожному імпульсі лічби вихідний код зменшується на

одиницю. Закон функціонування такого лічильника можна отримати з таблиці 2.1, читаючи її знизу вверх.

Таблиця 2.1

Стани двійкового лічильника



Ф	z_3	z_2	z_1	z_0
	2^3	2^2	2^1	2^0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
16	0	0	0	0

Рис. 2.2. Часові діаграми вихідних станів лічильника підсумовування

Двійкові лічильники досить широко використовуються у комп'ютерній техніці, зокрема як програмні лічильники – вказівники команд, що виконуються.

Регістри зсуву

Як уже зазначалося на початку даної лабораторної роботи, за допомогою тригерів можна зберігати двійкову інформацію. Для синхронних тригерів вхідна інформація вводиться по певному фронту тактового імпульсу в запам'ятовуючу комірку, тобто передається на вихід. Якщо з'єднати декілька таких тригерів послідовно, тобто виходи кожного тригера з'єднати із входами сусідів, інформація з приходом кожного наступного тактового сигналу буде зсуватися з кожного окремого тригера в сусідній. Тому такий пристрій називається тактовим регістром зсуву. Напрямок зсуву інформації в регістрі визначається напрямком з'єднання відповідних інформаційних ліній між тригерами. Крім того, можливе з'єднання виходу старшого тригера із входом молодшого. У такий спосіб можна організувати циклічний зсув

інформації в регістрі вліво. З'єднавши вихід молодшого тригера із входом старшого, можна організувати циклічний зсув інформації в регістрі вправо. У таблиці 2.2 зображено приклад функціонування чотирьохрозрядного регістра циклічного зсуву. Після запису у регістр деякої двійкової інформації $D_3\ D_2\ D_1\ D_0$ з кожним приходом чергового фронту синхроімпульсу Φ інформація у регістрі спочатку циклічно зсувається на три розряди вправо, а потім циклічно зсувається на стільки ж розрядів вліво.

Таблиця 2.2

Таблиця функціонування 4-розрядного регістра циклічного зсуву

Φ	Q_3	Q_2	Q_1	Q_0
1	D_3	D_2	D_1	D_0
2	D_2	D_1	D_0	D_3
3	D_1	D_0	D_3	D_2
4	D_0	D_3	D_2	D_1
5	D_1	D_0	D_3	D_2
6	D_2	D_1	D_0	D_3
7	D_3	D_2	D_1	D_0

Практична частина

Складіть схему для дослідження особливостей роботи запам'ятовуючої комірки (RS-тригера) в базисі АБО-НЕ, яка зображена на рис. 2.3.

Тригер можна отримати, охопивши, як це показано на рис. 2.3, два логічних елементи АБО-НЕ зворотними зв'язками. Він має, як вже було зазначено, два вихідних сигнали: Q та $\overline{Q}$, інверсні один до одного, і два вхідних: S (установка – Set) і R (скидання – Reset).

Якщо вхідні сигнали взаємоінверсні, причому $S = 1$ і $R = 0$, то:

$$\overline{Q} = \overline{S + Q} = \overline{1 + Q} = 0, \qquad Q = \overline{R + \overline{Q}} = \overline{0 + 0} = 1.$$

Отже, обидва вихідних сигнали дійсно знаходяться в інверсних один до одного станах. При $S = 0$ і $R = 1$ можна

отримати обернені значення вихідних сигналів тригера. Якщо $R = S = 0$, стан вихідних сигналів зберігається. Саме тому RS-тригер можна використовувати для запам'ятовування інформації. При $R = S = 1$ обидва вихідних сигнали Q та $\overline{Q}$ дорівнюють нулю; однак у цьому випадку стан вихідних сигналів тригера не буде однозначно визначений, якщо в який-небудь момент обидва вхідних сигнали одночасно дорівнюватимуть нулю. Під невизначеністю тут слід розуміти перехід RS-тригера у деякий стійкий стан (Q = 1, $\overline{Q}$ = 0 або Q = 0, $\overline{Q}$ = 1), але в який саме стан перейде тригер, передбачити неможливо. Тому комбінація вхідних сигналів $R = S = 1$, як правило, заборонена. Під час моделювання у системі Proteus ситуації переходу з $R = S = 1$ у $R = S = 0$ виникає аварійна помилка. Всі можливі стани тригера відображені в таблиці переключень 2.3.

Таблиця 2.3

Таблиця переключень RS-тригера

S	R	Q
0	0	Зберігається попереднє значення
0	1	0
1	0	1
1	1	Заборонено (Q = 0, $\overline{Q}$ = 0)

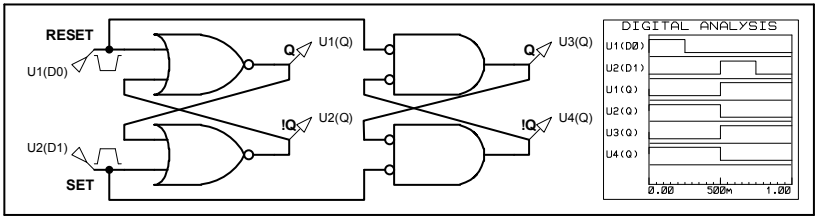


Рис. 2.3. Схеми RS-тригерів у базисах АБО-НЕ та І-НЕ

Проаналізуйте роботу готових моделей синхронних D-тригера та JK-тригера, які зображені відповідно на рис. 2.4 та рис. 2.5. D-тригер має лише один інформаційний вхід для запису відповідного значення логічного сигналу. Його можна легко побудувати на основі *RS*-тригера, якщо вхід R приєднати через

елемент інверсії до входу S, який буде відігравати роль інформаційного D-входу для записування даних (Data). JK-тригер має два входи: J і K. Якщо $J = K = 1$, то тригер із приходом кожного чергового фронту синхроімпульсу змінює свій стан на протилежний. При інших значеннях входів J і K тригер повторює таблицю переходів RS-тригера, а саме: J діє як вхід установки, а K – як вхід скиду. Єдина різниця виникає при забороненій комбінації вхідних сигналів $R = S = 1$, коли вихідний сигнал змінюється при кожному такті синхронного сигналу C.

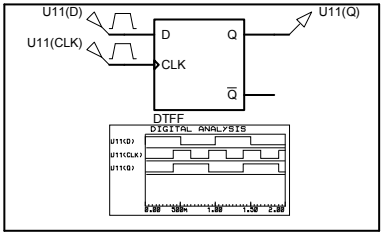


Рис. 2.4. Модель D-тригера

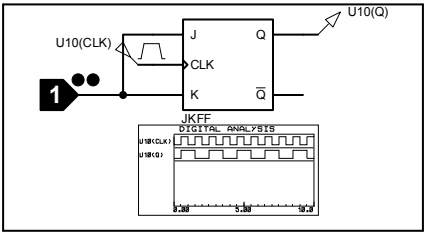


Рис. 2.5. Модель JK-тригера

Складіть схему асинхронного двійкового лічильника, зображеного на рис. 2.6. В даному випадку лічильник реалізовано у вигляді ланцюга тригерів, тактовий вхід кожного з яких підключений до інверсного виходу $\overline{Q}$ попереднього тригера. Для отримання лічильника підсумовування тригери повинні змінювати свій стан при переході тактового сигналу з „0” в „1”. Отже, необхідні тригери, які спрацьовують по передньому фронту імпульсу, наприклад JK-тригери, що працюють у режимі T-тригерів при умові $J = K = 1$. Розрядність у таких лічильниках можна нарощувати. Наприклад, за допомогою 10-розрядного двійкового лічильника можна підрахувати 1023 імпульсів.

Можна використовувати тригери, які спрацьовують при переході тактового сигналу з „1” в „0”. З’єднавши їх так, як показано на рис. 2.6, отримаємо лічильник віднімання. Для отримання лічильника додавання необхідно інвертувати сигнали на тактових входах. Для цього достатньо просто підключити

тактові входи тригерів до неінверсних виходів Q попередніх тригерів.

Кожний лічильник водночас є дільником частоти. Частота на виході наймолодшого тригера дорівнює половині тактової частоти. На виході другого тригера частота складає 1/4 вхідної частоти, на виході третього – 1/8 і т.д. Це ділення частоти добре видно з часових діаграм, побудованих за допомогою цифрового аналізатора.

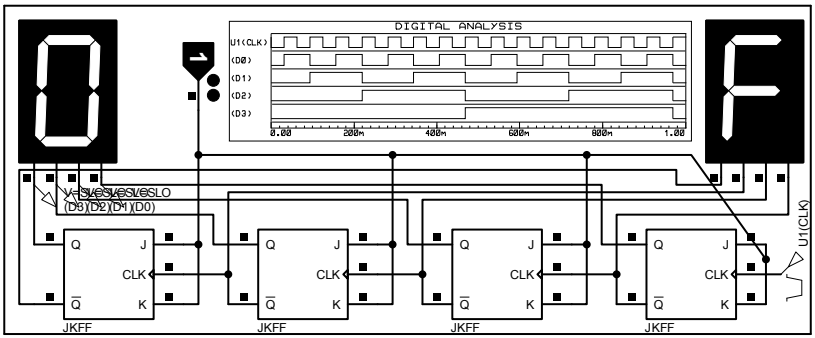


Рис. 2.6. Асинхронний (послідовний) лічильник

Іноді виникає необхідність у побудові лічильників, які рахують імпульси у двійково-десяткових кодах, тобто від 0 до 9. Отримати такий лічильник можна шляхом уведення деякої допоміжної комбінаційної схеми та передбачення входу скидання у нуль тригерів, з яких складається лічильник. Складіть схему двійково-десяткового лічильника з використанням готової моделі чотирирозрядного лічильника, як це показано на рис. 2.7. Виходи $Q0$ та $Q3$ лічильника через двовходовий елемент І з'єднано із входом reset (скидання). Таким чином, коли лічильник порахує до дев'яти ($9_{(10)} = 1001_{(2)}$), на виході логічного елемента І, а відповідно і на вході скидання, встановиться сигнал логічної одиниці, що переведе лічильник у нульовий стан із приходом наступного фронту синхроімпульсу.

Для організації зворотної лічби у двійково-десятковому коді слід передбачити вхід завантаження цифри „дев'ять" у

лічильник. Складіть схему двійково-десяткового лічильника зворотної лічби, як це показано на рис. 2.8. Виходи $Q0$, $Q1$, $Q2$ та $Q3$ лічильника через чотиривходовий елемент АБО-НЕ з'єднано із входом load (завантаження). Таким чином, коли лічильник порахує до нуля, на виході логічного елемента АБО-НЕ, а відповідно, і на вході завантаження встановиться сигнал логічної одиниці, що призведе до завантаження у лічильник цифри „дев'ять" із приходом наступного фронту синхроімпульсу.

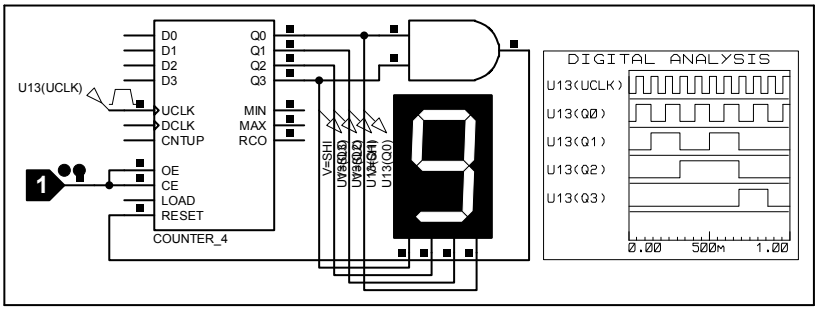


Рис. 2.7. Двійково-десятковий лічильник прямої лічби

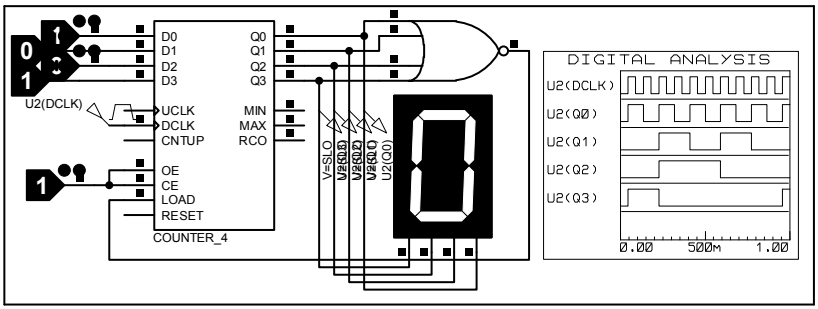


Рис. 2.8. Двійково-десятковий лічильник зворотної лічби

Аналогічно можна організувати лічильники з будь-яким коефіцієнтом перерахунку. Використовуючи два чотирирозрядних, два трирозрядних, два дворозрядних лічильники та відповідний набір комбінаційних схем, складіть схему електронного годинника, загальний вигляд якого зображений на рис. 2.9.

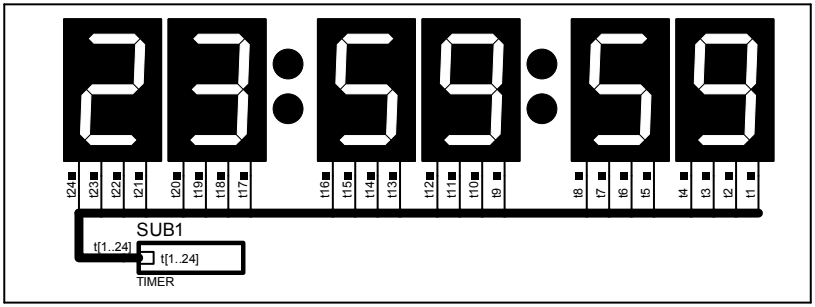


Рис. 2.9. Електронний годинник

Складіть схему чотирирозрядного регістра зсуву з можливістю організації циклічного зсуву інформації як вправо, так і вліво. Схема зображена на рис. 2.10 і побудована на базі D-тригерів. Для організації запису інформації та зсувів використано моделі цифрових ключів. У регістрі, крім чотирьох інформаційних входів, передбачено один вхід синхронізації U8(CLK) та два керуючих входи – вхід SW12(EN) керує завантаженням інформації у регістр, а вхід U12(D) керує напрямком циклічного зсуву інформації у регістрі. Якщо сигнал керування завантаженням одиничний, то активується запис інформації у регістр, а зсув забороняється. При нульовому сигналі керування завантаженням навпаки – запис інформації у регістр забороняється, а зсув дозволяється. При нульовому сигналі керування напрямком циклічного зсуву з кожним приходом чергового фронту синхроімпульсу інформація у регістрі зсувається вліво, а при одиничному – вправо.

Послідовність роботи окремих тригерів відображена в таблиці 2.2 та видна на часових діаграмах рис. 2.10, побудованих за допомогою цифрового аналізатора. В початковий момент часу сигнал керування завантаженням інформації у регістр SW12(EN) одиничний, отже з приходом переднього фронту синхроімпульсу U8(CLK) відбувається запис числа $0001_{(2)}$ у регістр. В наступний момент часу сигнал SW12(EN) нульовий, отже, в подальшому запис інформації у регістр заборонений, натомість дозволений її

зсув. Упродовж перших трьох тактів синхронізації сигнал керування напрямком зсуву U12(D) нульовий, тому інформація зсувається на три розряди вліво, тобто: $0010_{(2)}$, $0100_{(2)}$, $1000_{(2)}$. Протягом наступних трьох тактів синхронізації сигнал керування напрямком зсуву U12(D) одиничний, отже, інформація зсувається на три розряди вправо, тобто: $0100_{(2)}$, $0010_{(2)}$, $0001_{(2)}$.

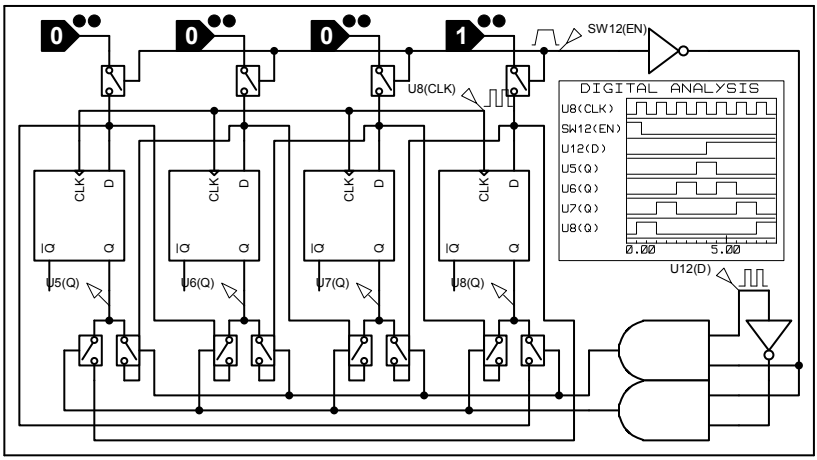


Рис. 2.10. Регістр зсуву

Якщо розірвати кола зв'язків між виходом старшого тригера та входом молодшого, а також між виходом молодшого та входом старшого, то отримаємо простий регістр зсуву. Такий регістр надає можливість запису інформації, яка подана у послідовному вигляді (послідовна передача інформації, яка використовується при передачі даних по одній лінії зв'язку, що є найдешевшим способом для великих відстаней). За чотири такти імпульсів синхронізації у регістр запишеться деяка інформація, яка, в свою чергу, може бути передана у паралельному вигляді. Таким чином, регістр можна використовувати для перетворення послідовної інформації у паралельну, яку можна передавати за один такт синхронізації, що надає очевидні переваги у швидкості передачі та обробки даних. Аналогічно дослідіть роботу готової моделі регістра зсуву SHIFTREG_4.

Контрольні запитання та завдання

1. Що являє собою елементарний запам'ятовуючий пристрій – тригер? Опишіть функціонування RS-тригера. Побудуйте граф алгоритму роботи RS-тригера та таблицю переходів. Опишіть функціонування JK-тригера. Як можна отримати D- і T- тригери на основі JK-тригера? Які ви знаєте методи синхронізації тригерів?
2. Складіть схему синхронного двоступеневого RS-тригера (схему, яка працює за принципом „ведучий-ведений" („Master-Slave")). Дослідіть особливості її роботи.
3. Що являють собою двійкові лічильники? Опишіть функціонування асинхронного двійкового лічильника на основі JK-тригерів, включених у режимі T- тригерів. Де використовуються двійкові лічильники? Як можна змінити коефіцієнт перерахунку лічильників?
4. Складіть схему синхронного чотирирозрядного лічильника імпульсів із можливістю запису в нього деякого двійкового числа в момент активації відповідного керуючого сигналу.
5. Що являють собою регістри зсуву? Де використовуються регістри зсуву? Що таке циклічний зсув і як його організувати?
6. Складіть схему на основі восьмирозрядних регістрів зсуву, яка б перетворювала паралельне подання двійкової інформації у послідовне та навпаки. Схема повинна передавати інформацію у послідовному вигляді по каналу зв'язку між двома регістрами зсуву.
7. Складіть схему, яка виконувала б завдання, аналогічне попередньому (див. п. 6), але була б побудована на основі трирозрядних двійкових лічильників, мультиплексора та демультиплексора. Схема повинна передавати інформацію у послідовному вигляді по каналу зв'язку між мультиплексором (передавач) та демультиплексором (приймач). Для зберігання інформації на приймальному кінці використовуйте синхронні D-тригери, а для відображення інформації – семисегментні світлодіодні індикатори.

Лабораторна робота № 3
Складання схем пам'яті з адресним способом доступу до даних та дослідження особливостей їх роботи

Теоретична частина

Способи доступу до даних у напівпровідниковій пам'яті

У напівпровідникових запам'ятовуючих пристроях (ЗП) виділяють адресні, послідовні й асоціативні способи доступу до даних [11].

В адресних ЗП усі комірки пам'яті в момент звернення рівнодоступні. До адресних ЗП відносять: RAM (Random Access Memory) – ОЗП (оперативний ЗП) або ЗПДВ (ЗП із довільною вибіркою) та ROM (Read Only Memory) – ПЗП (постійний ЗП).

Оперативні ЗП зберігають дані, необхідні при виконанні поточної програми; вони можуть бути змінені в будь-який момент часу. Оперативні ЗП здебільшого енергонезалежні. У постійних ЗП вміст комірок або взагалі не змінюється, або змінюється рідко в спеціальних режимах.

Запам'ятовуючі пристрої RAM поділяються на статичні SRAM (Static RAM) і динамічні DRAM (Dynamic RAM). У статичних RAM елементами пам'яті є тригери. Вони зберігають свій стан, поки схема має напругу живлення і нові дані не записуються. У динамічних RAM дані зберігаються у вигляді зарядів конденсаторів, створюваних компонентами МОН-транзисторів. Саморозряд конденсаторів веде до руйнування даних, тому вони періодично (кожні 2 – 30 мс) мають регенеруватися. Але щільність упакування динамічних елементів пам'яті в 4 – 5 разів перевищує такий самий показник для статичних RAM. Регенерація даних здійснюється за допомогою спеціальних контролерів. Розроблені також DRAM із внутрішніми схемами регенерації; такі ЗП називаються квазістатичними.

Постійна пам'ять типу ROM(M) програмується при виготовленні за допомогою масок, тому такі ПЗП називають масковими. В подальших різновидах ROM у позначеннях є буква P (від Programmable). Це – пам'ять, що одноразово програмується користувачем PROM (ППЗП – програмовані ПЗП) і

багаторазово – EPROM, EEPROM. Пам'ять типу Flash елементами пам'яті подібна до EEPROM, але має структурні та технологічні особливості, які дозволяють виділити її в окремий тип.

У ЗП із послідовним доступом дані, що записуються, створюють чергу. Зчитування виконується слово за словом у порядку записів або навпаки. Прямий порядок зчитування використовується в буферах FIFO з дисципліною „перший прийшов – перший вийшов (First In – First Out)", а також у файлових та циклічних ЗП.

Різниця між пам'яттю FIFO та файловим ЗП полягає в тому, що у FIFO записування в порожній буфер зразу доступне для читання (тобто надходить у кінець ланцюга), а у файлових ЗП доступ до читання можливий лише після деякої кількості звернень, яке дорівнює кількості елементів у ланцюзі.

У циклічних ЗП слова доступні одне за одним із постійним періодом, який визначається ємністю пам'яті. До них належить відеопам'ять VRAM.

Зчитування у зворотному порядку властиве стековим ЗП із дисципліною „останнім прийшов – першим вийшов". Такі ЗП називаються буферами LIFO (Last In – First Out).

Час доступу до конкретної одиниці інформації, що зберігається в послідовних ЗП, є випадковою величиною. В найгіршому випадку для такого доступу треба переглянути весь об'єм інформації, що зберігається у цій пам'яті.

Асоціативний доступ реалізує пошук інформації за деякою ознакою, а не за адресою. В найбільш повній версії всі слова, які зберігаються в пам'яті, можуть одночасно перевірятися на відповідність ознаці, наприклад, на збіг визначених полів слів – тегів (від tag) за ознакою, яку задає вхідне слово (тегова адреса). На вихід передаються слова, які задовольняють ознаку. Дисципліна видавання слів, якщо тегу задовольняє кілька слів, та дисципліна записування нових даних можуть бути різними. Основна область використання асоціативної пам'яті в комп'ютерах – кешування даних.

Основні структури напівпровідникової пам'яті

Елементний базис пам'яті сучасних комп'ютерів складають мікросхеми різного ступеня інтеграції. Основою будь-якого запам'ятовуючого пристрою (ЗП) є елемент пам'яті (ЕП) статичного або динамічного типу, призначений для записування, зберігання і зчитування одного біта інформації – цифри 0 або 1. Сукупність ЕП, які утворюють *n*-розрядне слово, називають коміркою пам'яті (КП). Множина КП утворює запам'ятовуючий масив, який називається матрицею М елементів пам'яті.

Кожна матриця М у пристрої пам'яті має систему адресних і розрядних ліній (провідників). Адресні (словникові) лінії служать для виділення за адресою будь-якої КП. Сукупність різних адресних кодів утворює адресний простір пам'яті. Розрядні лінії записування (ЛЗП) служать для введення в кожний розряд вибраної КП цифри 0 або 1 відповідно до вхідної інформації. Розрядні лінії зчитування (ЛЗЧ) служать для знімання інформації, яка зберігається, з розряду вибраної КП. Часто використовують спільну лінію записування-зчитування (ЛЗЗ). Адресні та розрядні лінії разом називаються лініями вибірки. Якщо довжина адресного коду дорівнює k, то кількість слів N, які зберігаються в пам'яті як окремі одиниці даних, визначаються зі співвідношення $N=2^k$.

Структуру пам'яті визначає спосіб розподілу КП між адресними та розрядними лініями. За цією ознакою виділяють такі структури пам'яті: 2D, 3D, 2,5D і модифіковану – 2DM (D від Dimension – розмірність).

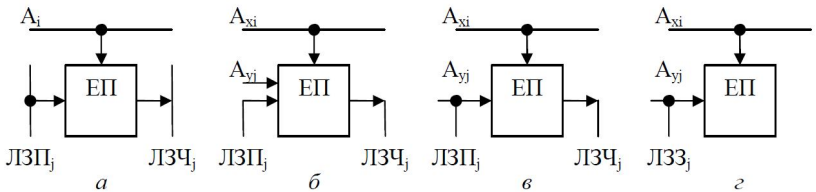


Рис. 3.1. Узагальнене поняття структури пам'яті:
а – 2D, *б* – 3D, *в* – 2,5D, *г* – 2DM

У системі 2D кожний ЕП має одну адресну лінію A_i (одне D), лінії записування ЛЗП$_j$ і зчитування ЛЗЧ$_j$, які спільно утворюють друге D (рис. 3.1,*а*). У структурі 3D адресу розділяють на дві частини: старша A_x визначає адреси рядків, а молодша A_y – адреси стовпців; разом вони утворюють 2D. Лінії записування і зчитування утворюють третє D (рис. 3.1,*б*). У структурі 2,5D одна з ЛЗП або ЛЗЧ суміщена з напівадресою A_{xi} або A_{yi} (рис. 3.1,*в*). У модифікованій системі 2DM використовується спільна лінія ЛЗЗ$_j$, яка поєднується з адресною лінією A_{yi} мультиплексорами. Розглянуті структури характерні для статичних ОЗП і пам'яті типу ROM. Структури динамічних ОЗП мають свою специфіку.

До складу мікросхем пам'яті входять:

- матриця елементів пам'яті М, яка містить N рядків і m стовпців (за кількістю розрядів слова);
- буфер і дешифратор k-розрядної адреси з кількістю виходів $N = 2^k$;
- буферні формувачі вхідних і вихідних інформаційних сигналів у режимах записування і зчитування й буфер даних;
- блок місцевого керування.

Розглянемо коротко організацію матриць пам'яті, що мають різну структуру. Матриця М пам'яті зі структурою 2D організована так, що при звертанні до пам'яті вибираються ЕП, які розміщені на збудженому виході дешифратора адреси. Недоліком структури 2D є складність побудови дешифратора адреси з кількістю виходів N, яке дорівнює кількості слів, що зберігаються у пам'яті. Тому структура типу 2D використовується в ЗП малої ємності.

У пам'яті зі структурою 3D адресний код розділяється на дві частини A_x і A_y, кожна з яких декодується окремими дешифраторами. Матриця М складається з m підматриць за кількістю розрядів слова. Кожна матриця зберігає значення свого *i*-го розряду всіх N слів. При звертанні до пам'яті вибирають m запам'ятовуючих елементів (по одному з кожної підматриці), які знаходяться на перетині рядка і стовпця збуджених виходів дешифраторів молодшої і старшої частин адреси. Така структура часто використовується з однорозрядною організацією N*1 біт.

Для цієї пам'яті ємністю 1 К слів треба мати два дешифратори з кількістю виходів у кожному $N=2^5=32$. Для пам'яті зі структурою 2D із тією самою ємністю дешифратор має 1024 виходи.

Недоліком структури 3D є використання складних ЕП, які допускають двокоординатну вибірку та складнішу структуру матриці М. У пам'яті з модифікованою структурою 2DM поєднуються переваги структур 2D і 3D.

У ЗП зі структурою 2DM адресний код довжиною k розбивається на дві частини: $A_x=A_kA_{k-1}\ldots A_{r+1}$, що надходить на дешифратор рядків, та $A_y=A_rA_{r-1}\ldots A_1$, що подається на входи мультиплексорів з організацією „$2^k \to 1$". Дешифратор обслуговує 2^{k-r} рядків, кожний з яких зберігає 2^k m-розрядних слів. У кожній групі зберігаються значення однойменних розрядів, і обслуговується вона своїм мультиплексором з організацією „$2^k \to 1$". Таким чином, активований вихід дешифратора вибирає рядок, а молодші розряди адреси за допомогою мультиплексорів забезпечують формування вихідного слова (по одному розряду з кожної групи). Організація пам'яті зі структурою 2DM найбільш розповсюджена, особливо для схем великої ємності.

Практична частина

Складіть схему для дослідження функціонування чотирирозрядної пам'яті з адресним способом доступу до даних, як зображено на рис. 3.2. Виходи чотирирозрядного двійкового лічильника приєднайте до адресних входів схем пам'яті та через буферні схеми – до входів даних. Передбачте в схемі лінію керування зчитуванням / записом інформації в пам'ять. У підсхемі SUB17 використайте готову модель мікросхеми пам'яті MEMORY_12_8, приєднавши до входу WR мікросхеми через двовходовий елемент логічного множення (AND_2), до входів якого приєднайте лінії WR та CLK. У підсхемі SUB18 зберіть схему пам'яті зі структурою 2D, як це показано на рис. 3.3. Вміст чотирирозрядних комірок пам'яті SUB1 – SUB16 побудуйте на основі синхронних D-тригерів зі статичним керуванням записом, як це показано на рис. 3.4. З рисунка видно, що під час запису інформації у комірки пам'яті (WR = 1) цифрові ключі комутують

між собою лінії даних та входи тригерів, а під час зчитування інформації з комірок пам'яті (RD = 1) цифрові ключі комутують між собою лінії даних та виходи тригерів. Окрім того, необхідною умовою можливості зчитування / запису інформації в деяку комірку пам'яті є наявність сигналу логічної одиниці на відповідній адресній лінії з виходу дешифратора адреси.

Результатом роботи складеної схеми повинен бути процес запису у схеми пам'яті двійкових кодів цифр 0, 1, 2, ..., E, F упродовж перших 16 тактів генератора синхроімпульсів CLK та зчитування записаної інформації впродовж наступних 16 тактів, що видно з часових діаграм цифрового аналізатора на рис. 3.2. Керування процесом запису / зчитування відбувається за допомогою імпульсного сигналу WR/!RD, який з'єднаний також із буферними схемами для переведення у високоомний стан лінії від лічильника під час зчитування інформації зі схем пам'яті.

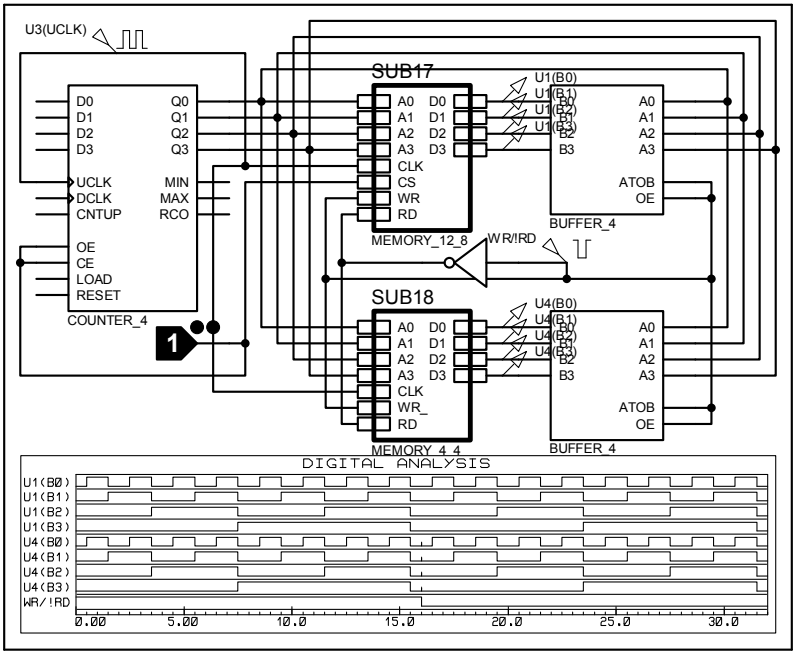


Рис. 3.2. Основна схема

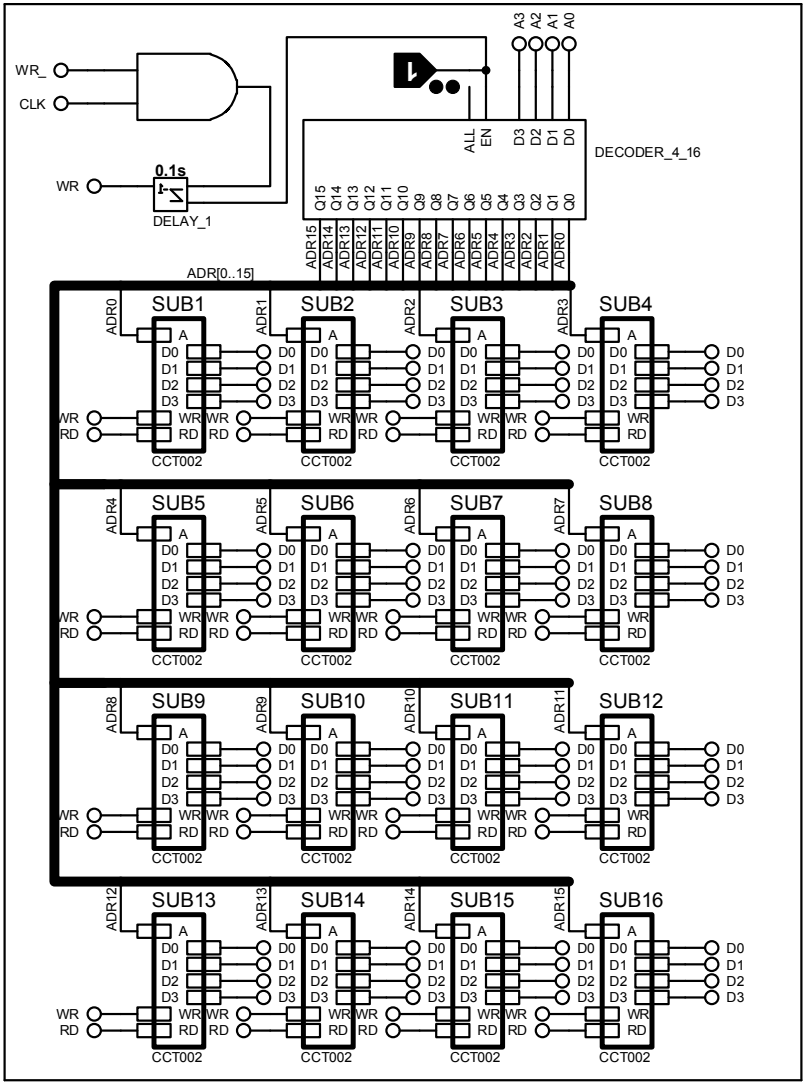


Рис. 3.3. Вміст SUB18 на рис. 3.2

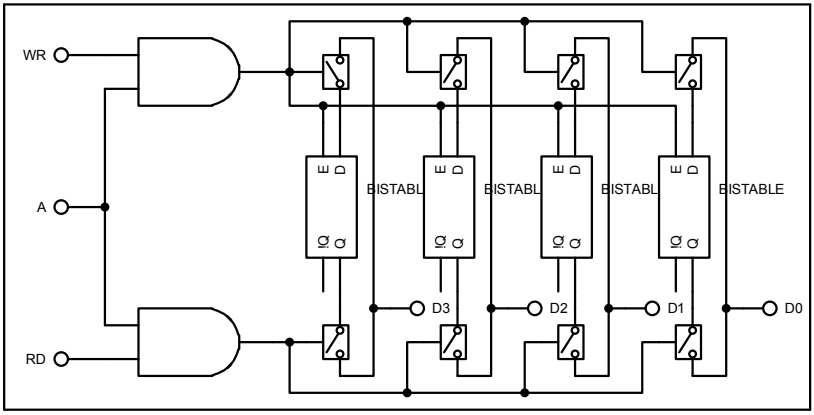


Рис. 3.4. Вміст SUB1 – SUB16 на рис. 3.3

Контрольні запитання та завдання

1. Які способи доступу до даних у мікросхемах пам’яті ви знаєте?
2. Яка різниця між статичними та динамічними запам’ятовуючими пристроями?
3. Яка різниця між постійними та оперативними запам’ятовуючими пристроями?
4. Охарактеризуйте запам’ятовуючі пристрої з дисциплінами FIFO і LIFO.
5. Як організовано доступ до даних в асоціативних запам’ятовуючих пристроях?
6. Що таке структура пам’яті та які її різновиди ви знаєте? Спроектуйте відомі вам структури 4-розрядних схем пам’яті.
7. Як організовується нарощування розрядності пам’яті? Продемонструйте можливість нарощування розрядності пам’яті на основі 4-розрядних схем пам’яті як у бік збільшення розрядності комірок, так і у бік розширення адресного простору.

Лабораторна робота № 4
Операційні та керуючі пристрої.
Проектування операційних пристроїв

Теоретична частина

Поняття мікропрограмного керування

Переважна більшість цифрових пристроїв, в тому числі й комп'ютери, переробляють інформацію, виконуючи над нею деякі операції. Для виконання операцій над інформацією використовуються операційні пристрої – процесори, канали введення-виведення, пристрої управління зовнішніми пристроями і т.д. Функцією операційного пристрою є виконання заданої множини операцій $F=\{f_1,...,f_G\}$ над вхідними словами $D=\{d_1,...,d_H\}$ з метою обчислення слів $R=\{r_1,...,r_Q\}$, які подають результати операцій $R=f_g(D)$, де $g=1,2,...,G$.

Функціональна і структурна організація операційних пристроїв базується на принципі мікропрограмного управління, який полягає в такому:

1. Будь-яка операція $f_g(g=1,...,G)$, що реалізується пристроєм, розглядається як складна дія, яка розділяється на послідовність елементарних дій над словами інформації. Ці елементарні дії називаються *мікрооперaціями*.

2. Для управління порядком проходження мікрооперацій використовуються *логічні умови*, які залежно від значень слів, що перетворюються мікрооперaціями, набувають значення *"істина"* або *"брехня"*, („true" або „false"), (1 або 0).

3. Процес виконання операцій в пристрої описується у формі алгоритму, який представляється в термінах мікрооперацій і логічних умов і називається *мікропрограмою.* Мікропрограма визначає порядок перевірки значень логічних умов і проходження мікрооперацій, необхідний для отримання потрібних результатів.

4. Мікропрограма використовується як форма подання функції пристрою, на основі якої визначається структура і порядок функціонування пристрою в часі.

Таким чином, із принципу мікропрограмного управління випливає, що структура і порядок функціонування операційних

пристроїв визначається алгоритмом виконання операції $F=\{f_1,...,f_G\}$.

До елементарних дій над словами інформації, тобто мікрооперацій, належать: передача інформації з одного регістра в інший, утворення оберненого коду, зсув, додавання тощо.

Поняття операційного та керуючого автоматів

Як показав академік В.М. Глушков, у будь-якому пристрої обробки цифрової інформації можна виділити два основні блоки – операційний автомат (ОА) і керуючий автомат (КА) (рис. 4.1).

Операційний автомат (ОА) служить для зберігання слів інформації, виконання набору мікрооперацій і обчислення значень логічних умов, тобто операційний автомат є структурою, організованою для виконання дій над інформацією. Мікрооперації, що виконуються ОА, задаються множиною керуючих сигналів $Y=\{y_1,....,y_M\}$, з кожним з яких ототожнюється певна мікрооперація.

Значення логічних умов, що обчислюються в операційному автоматі, відображаються множиною інформаційних сигналів $X=\{x_1,...,x_L\}$, кожний з яких ототожнюється з певною логічною умовою.

Керуючий автомат (КА) генерує послідовність керуючих сигналів, вказану мікропрограмою і відповідну значенням логічних умов. Інакше кажучи, керуючий автомат задає порядок виконання дій в ОА, що визначається з алгоритму виконання операцій. Найменування операції, яку необхідно виконати в пристрої, визначається кодом g операції, що надходить у КА ззовні. По відношенню до КА сигнали $g_1,...,g_h$, за допомогою яких кодується найменування операції та інформаційні сигнали $x_1,...,x_L$, сформовані в операційному автоматі, відіграють однакову роль: вони впливають на порядок вироблення керуючих сигналів Y. Тому сигнали $g_1,...,g_h$ і $x_1,...,x_L$ належать до одного класу – до класу інформаційних сигналів, що надходять на вхід КА.

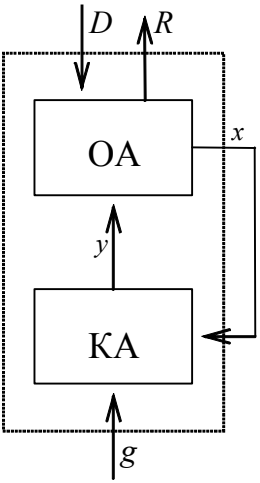


Рис. 4.1. Декомпозиція цифрового пристрою на керуючий автомат (КА) та операційний автомат (ОА)

Таким чином, будь-який операційний пристрій – процесор, канал введення-виведення і т.д. – є композицією операційного та керуючого автоматів. Операційний автомат, реалізовуючи дії над словами інформації, є виконавчою частиною пристрою, роботою якого управляє керуючий автомат, що генерує необхідні послідовності керуючих сигналів.

Операційний та керуючий автомати можуть бути визначені своїми функціями – переліком виконуваних ними дій.

Функція ОА визначається такою сукупністю відомостей:

1) множиною вхідних слів $D=\{d_1,...,d_H\}$, що вводяться в автомат як операнди;

2) множиною вихідних слів $R=\{r_1,...,r_Q\}$, що являють собою результати операцій;

3) множиною внутрішніх слів $S=\{s_1,...,s_N\}$, що використовуються для подання інформації в процесі виконання операцій. Можна вважати, що вхідні і вихідні слова збігаються з певними внутрішніми $D\subseteq S$, $R\subseteq S$;

4) множиною мікрооперацій $Y=\{y_m\}$, що реалізують перетворення $S=\varphi_m(s)$ над словами інформації, де φ_m – обчислювана функція;

5) множиною логічних умов $X=\{x_i\}$, де $x_i=\psi_i(s_i)$, ψ_i – булева функція.

Отже, функція ОА задана, якщо задана (визначена) множина D, R, S, Y, X. Час не є аргументом функції ОА. Функція встановлює список дій-мікрооперацій і логічних умов, які може виконувати автомат, але ніяк не визначає порядок проходження цих дій у часі. Тобто функція ОА характеризує засоби, які можуть бути використані для обчислень, але не сам обчислювальний процес.

Порядок виконання дій у часі визначається у формі функцій керуючого автомата.

Функція керуючого автомата – це операторна схема алгоритму (мікропрограми), функціональними операторами якої є символи $y_1...,y_m$, ототожнювані з мікрооперaціями, і як логічні умови використовуються булеві змінні $x_1...,x_L$. Операторна схема алгоритму найбільш часто зображується у вигляді граф-схеми алгоритму (ГСА). ГСА визначає обчислювальний процес послідовно в часі, встановлюючи порядок перевірки логічних умов $x_1 - x_L$ і порядок проходження мікрооперацій $y_1 - y_m$.

Операційні елементи

Як уже зазначалося вище, згідно із принципом мікропрограмного керування, будь-яка складна операція розпадається на ряд мікрооперацій, які виконуються операційним автоматом (ОА). Різні мікрооперації виконуються елементарними ОА – так званими операційними елементами (ОЕ), які є складовими частинами основного ОА.

Під операційним елементом розуміють пристрій, що реалізує одну з нижченаведених функцій або їх довільну комбінацію:

- зберігання слова інформації;
- виконання мікрооперацій, в результаті яких обчислюється нове значення слова інформації;
- обчислення логічної умови, яка залежить від слова інформації;

Таким чином, функція ОЕ визначена, якщо задані:

- опис слова, що зберігається або обчислюється;
- опис множини мікрооперацій, що виконуються цим елементом;

- опис обчислюваних цим елементом логічних умов.

Для побудови ОА операційні елементи з'єднуються між собою за допомогою ланцюгів передачі слів інформації від виходів одних елементів до входів інших.

Залежно від виконуваних мікрооперацій ОЕ діляться на різновиди: шина, регістр, лічильник, суматор, схема порівняння, дешифратор, шифратор тощо.

Практична частина

Реалізуємо цифровий операційний пристрій, який являє собою арифметично-логічний блок для виконання операцій додавання, віднімання, множення та ділення двох восьмирозрядних цілих чисел без знака (див. алгоритми двійкової арифметики [11 – 14]). Структура пристрою буде являти собою композицію операційного та керуючого автоматів. Керуючий автомат побудуємо у вигляді мікропрограмного автомата із програмованою логікою (це буде зроблено у лабораторній роботі № 5). В рамках даної лабораторної роботи ми спроектуємо загальну структуру операційного пристрою та схему операційного автомата.

На рис. 4.2 показано загальний вигляд схеми операційного пристрою. Тут виділено операційний автомат (підсхема SUB1), керуючий автомат (підсхема SUB3), регістри операндів B і A (ініційовані для прикладу числами $B2_{(16)}$ та $93_{(16)}$ відповідно), допоміжні буферні схеми, схема запуску операції та набір логічних констант для задання адреси початку мікропрограми виконання тієї чи іншої операції. Так, після виконання операції множення вміст регістрів B і A набуде значення $66_{(16)}$ та $36_{(16)}$ відповідно, оскільки $B2_{(16)} \times 93_{(16)} = 6636_{(16)}$. Арифметичні операції будуть розпочинатися при натисканні кнопки „Запуск операції". Результат буде висвітлюватися практично миттєво за рахунок високої тактової частоти роботи схеми (10 кГц). Схему операційного автомата зображено на рис. 4.3.

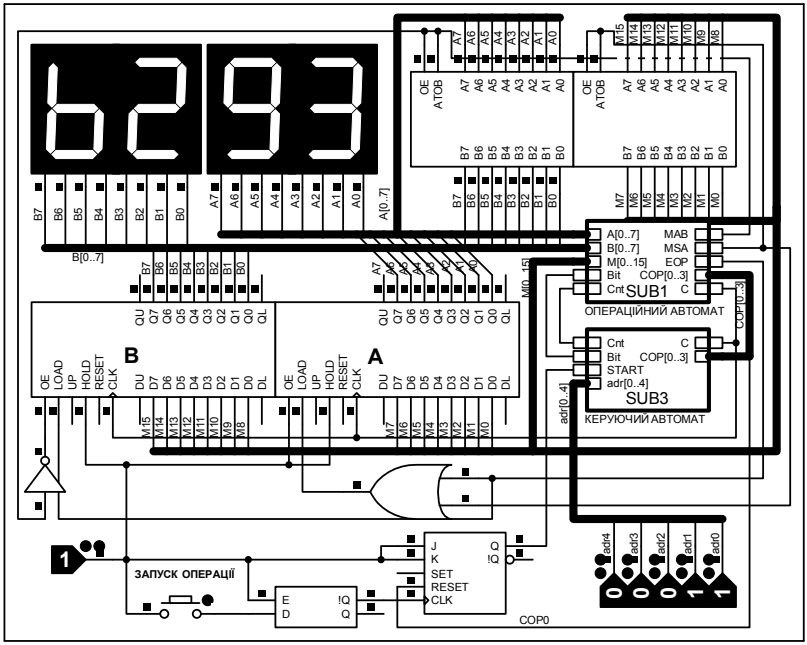


Рис. 4.2. Загальний вигляд схеми операційного пристрою

Операційний пристрій, який буде реалізовано в цій і наступній лабораторних роботах (рис. 4.2), має три логічні умови, в залежності від значень яких відбуваються або ігноруються умовні переходи, та ознаку безумовного переходу 11. У таблиці 4.1 наведено опис кодів логічних умов для команд умовних переходів. Ця інформація буде використана при проектуванні мікропрограмного автомата. В рамках даної лабораторної роботи нам необхідно виділити множину мікрооперацій, які б забезпечували виконання арифметичних операцій додавання, віднімання, множення та ділення двійкових цілих беззнакових чисел. Після визначення множини мікрооперацій необхідно синтезувати структуру операційного автомата. У таблиці 4.2 наведено опис кодів операцій, які подаються на дешифратор мікрооперацій в операційну схему. Таким чином, тут реалізовано вертикальне кодування операційних мікрокоманд (див. л. р. № 5).

Таблиця 4.1

Опис кодів логічних умов для команд умовних переходів мікропрограмного керуючого автомата

№	x_1x_0	Назва	Призначення
0	00	START	Перевірка натискання кнопки START для запуску операції
1	01	Bit	Аналіз значення тригера Bit
2	10	Cnt	Аналіз вмісту лічильника тактів
3	11	Логічна 1	Безумовний перехід

Таблиця 4.2

Опис кодів операцій, які подаються на дешифратор в операційному автоматі

№	$y_3y_2y_1y_0$	Назва	Призначення
0	0000	ONE	Заборонити виведення результату операції
1	0001	LAB	Завантажити в регістри A і B операнди (із зовнішніх регістрів A і B)
2	0010	L_S	Додавання до регістра-суматора вмісту регістра B
3	0011	SH_R	Зсув вправо вмісту регістра A та регістра-суматора
4	0100	R_CA	Скидання лічильника та регістра-суматора
5	0101	INC	Збільшення на одиницю (інкремент) вмісту лічильника
6	0110	OE	Дозволити виведення результату операції
7	0111	SHLA	Зсув вліво вмісту регістра A
8	1000	SH_L	Зсув вліво вмісту регістра A та регістра-суматора
9	1001	S_L	Віднімання від регістра-суматора вмісту регістра B
10	1010	W_1	Встановити в 1 тригер для запису в молодший біт A
11	1011	W_0	Скинути в 0 тригер для запису в молодший біт A
12	1100	SUB	Порівняти вміст регістра-суматора і регістра B шляхом віднімання та занести знак результату у тригер Bit
13	1101	L_B	Встановити тригер Bit відповідно до молодшого розряду регістра A
14	1110	MAB	Запис із зовнішнього регістра A у внутрішній B
15	1111	MSA	Запис із внутрішнього регістра B у зовнішній A

На рис. 4.2 наведено схему операційного автомата. Тут, окрім вищезгаданого дешифратора мікрооперацій, можна виділити: подвоєний регістр A із регістром-суматором, вхід якого підключено до 8-розрядного суматора (рис. 4.4), регістр B, лічильник тактів, тригер заборони/дозволу виведення результатів, тригер для запису логічних значень у молодший біт регістра A, тригер-прапорець Bit зберігання логічної умови для мікропрограмного керуючого автомата та ряд допоміжних логічних елементів.

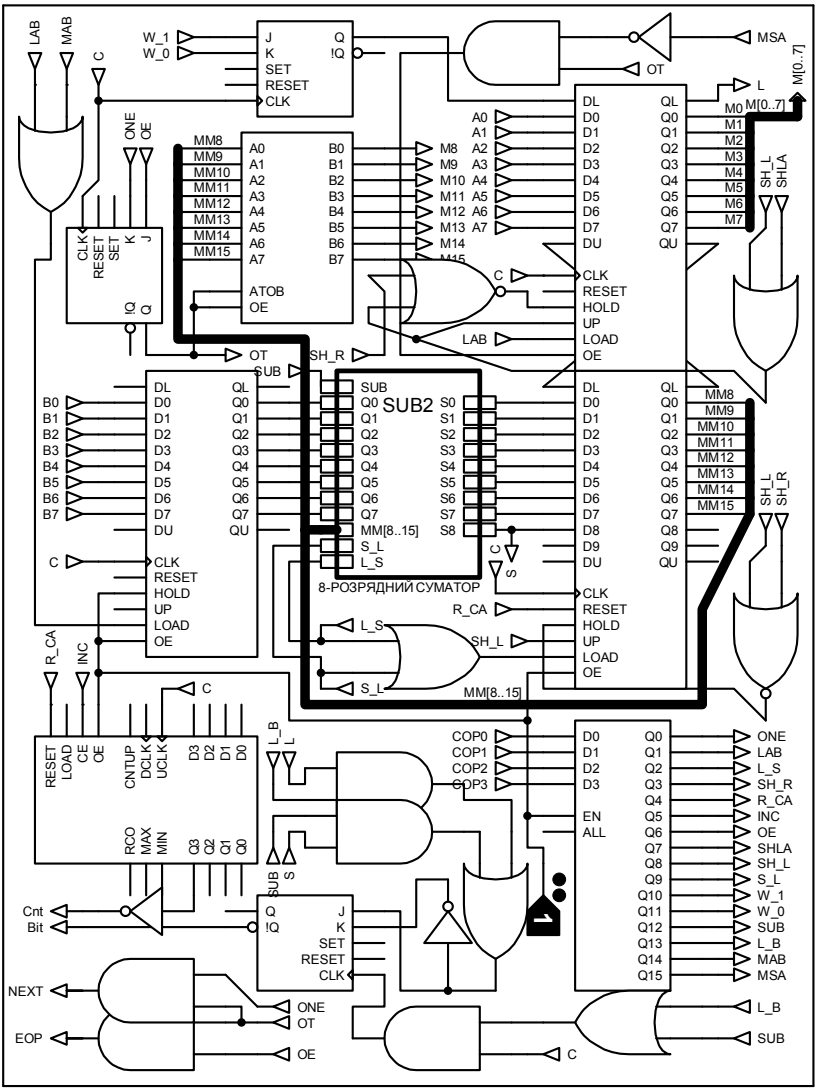


Рис. 4.3. Схема операційного автомата

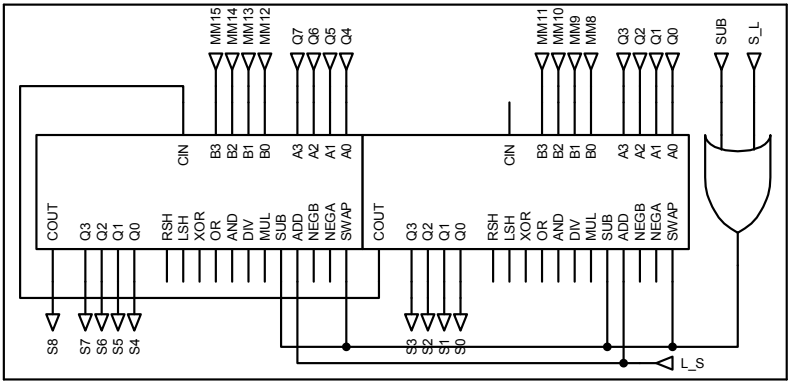


Рис. 4.4. Вміст підсхеми SUB2 на рис. 4.3 – 8-розрядний суматор

Контрольні запитання та завдання

1. Опишіть принципи мікропрограмного керування.
2. Які функції виконують операційні та керуючі автомати?
3. Спроектуйте операційний автомат для виконання операції додавання двох цілих 8-розрядних чисел без знака. Додавання повинно виконуватись послідовно впродовж восьми тактів синхронізації.
4. Спроектуйте операційний автомат для виконання операції віднімання двох цілих 8-розрядних чисел без знака. Віднімання повинно виконуватись паралельним чином. Від'ємні числа подайте в оберненому коді.
5. Які алгоритми множення та ділення ви знаєте? Охарактеризуйте їх вплив на структуру операційних автоматів. Які методи прискорення множення ви знаєте?
6. Спроектуйте операційний автомат для виконання операції множення двох цілих 8-розрядних чисел без знака. Алгоритм множення: множення, починаючи з молодших розрядів множника, із зсувом суми часткових добутків вправо та при нерухомому множеному.
7. Спроектуйте операційний автомат для виконання операції ділення двох цілих 8-розрядних чисел без знака. Алгоритм ділення: ділення з нерухомим дільником і діленим, що зсувається вліво (метод ділення без відновлення залишку).

Лабораторна робота № 5
Проектування пристроїв керування на основі мікропрограмних автоматів

Теоретична частина

Особливості організації керуючих автоматів у складі операційних пристроїв

Виконання будь-якої команди в операційних пристроях, наприклад у мікропроцесорах (МП), можна розбити на два цикли: циклу вибірки команди і циклу виконання команд.

У циклі вибірки команди пристрій керування (ПК) проводить зчитування коду операції і його декодування. У циклі виконання команди, відповідно до типу реалізованої команди, здійснюється визначення адрес операндів, що беруть участь в операції, безпосередньо виконання команди і збереження результатів операції.

Будь-яка команда, що виконується операційним блоком, описується деякою мікропрограмою і реалізується за певну кількість тактів, в кожному з яких виконується одна або декілька мікрооперацій. При виконанні мікропрограми на відповідні керуючі лінії операційного блока подається певним чином розподілена в часі послідовність керуючих функціональних сигналів (мікрооперацій). Порядок виконання мікрооперацій може змінюватися в залежності від ознак операції, що виробляються в арифметично-логічному пристрої (АЛП) і є вхідними сигналами ПК.

Отже, пристрій керування формує розподілену в часі і просторі послідовність зовнішніх і внутрішніх керуючих сигналів (ВКС), що забезпечують вибірку і виконання команди.

Однією з найважливіших характеристик ПК є можливість зміни послідовності керуючих слів (мікрооперацій). За цим критерієм ПК підрозділяються на ПК із «жорсткою» логікою, або спеціалізовані ПК, і на універсальні, або мікропрограмні ПК (із «гнучкою» логікою). На рис. 5.1 зображена структура ПК із «жорсткою» логікою.

Вхідною інформацією для ПК є вміст регістра команд, що визначає тип виконуваної команди, тобто код операції (КОП), і ознаки операцій, що виробляються АЛП.

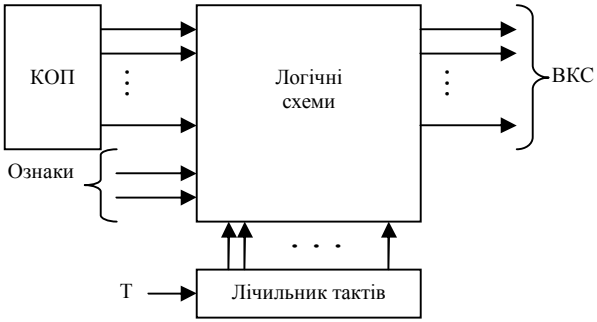


Рис. 5.1. Пристрій керування з «жорсткою» логікою

Вихідна інформація є сукупністю керуючих сигналів, що виробляються ПК відповідно до заданої мікропрограми. Інтервал часу, що відводиться на виконання мікрооперації, називається робочим тактом або тактом операційного пристрою. Тривалість такту встановлюється по найтривалішій мікрооперації. Синхронізація ПК здійснюється за допомогою лічильника тактів, керованого зовнішнім генератором тактових імпульсів Т.

До складу ПК входять запам'ятовуючі і комбінаційні схеми, що виконують функції запам'ятовування поточного стану, визначають сукупність ВКС і формують наступний стан відповідно до вхідних ознак. Мікропрограма в такому ПК задається взаємозв'язками між елементами логічних схем, основу яких складають лічильники, регістри, дешифратори. Зміна мікропрограми у ПК призводить до задання нових взаємозв'язків між елементами, що рівносильно проектуванню нової логічної схеми ПК. Таким чином, основною властивістю ПК із «жорсткою» логікою є фіксований набір системи команд, який заданий на етапі проектування. ПК із «жорсткою» логікою використовуються в спеціалізованих і однокристальних МП.

Структурна схема мікропрограмного ПК (МПК) – пристрою керування з логікою, що зберігається в пам'яті («гнучка» логіка), показана на рис. 5.2.

До складу пристрою входять контролер послідовності мікрокоманд (КПМК), регістр адреси мікрокоманд (РАМК), пам'ять мікрокоманд (ПМК), регістр мікрокоманди (РМК) і дешифратор мікрооперацій (ДШ МО).

Основним призначенням КПМК є реалізація алгоритмічних керуючих структур, що зустрічаються в мікропрограмах: лінійної послідовності, алгоритмічної структури типу «якщо P, то X, інакше Y» і алгоритмічної структури типу «поки P, роби X». При цьому контролер реалізує такі функції:

- проводить дешифрацію коду операції команди (КОП) для звернення до першої мікрокоманди мікропрограми, що інтерпретує дану команду;
- формує адреси наступних мікрокоманд по вказаних вище трьох керуючих структурах;
- зберігає ознаки переходів, що надходять з операційного блока та використовуються при виконанні мікрокоманд умовного переходу;
- здійснює управління перериваннями на мікропрограмному рівні.

Пам'ять мікрокоманд призначена для зберігання мікрокоманд, її місткість і розрядність однозначно визначаються набором реалізовуваних мікропрограм. Шляхом зміни набору мікропрограм можна гнучко міняти систему команд мікропроцесора і тим самим орієнтувати його функціональну спрямованість.

Регістр мікрокоманди призначений для зберігання мікрокоманди при виконанні поточного мікрокомандного циклу. Мікрокоманда містить три основні поля: код мікрооперації КМО, адресу наступної мікрокоманди АНМК, поле коду ознак КО, в якому указується, яку ознаку розгалуження в мікропрограмі необхідно аналізувати КПМК.

Дешифратор мікрооперацій служить для декодування коду мікрооперації і формування керуючих сигналів, що ініціюють виконання відповідних мікрооперацій в операційному блоці.

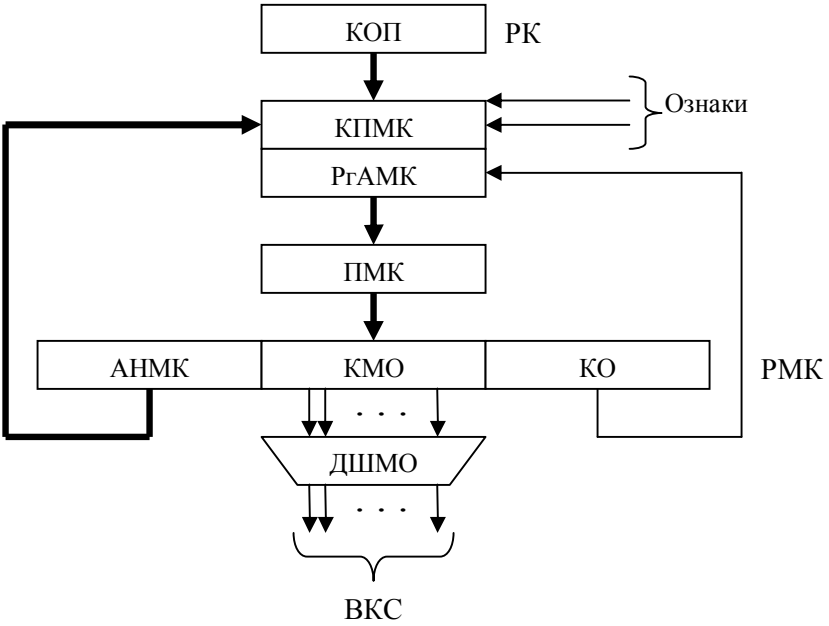


Рис. 5.2. Мікропрограмний керуючий пристрій

Мікропрограмний пристрій керування функціонує так. КОП із регістра команд надходить на вхід КПМК. КПМК дешифрує його і на виході регістра адреси мікрокоманд контролера формується адреса першої мікрокоманди виконуваної мікропрограми. Мікрокоманда, що підлягає реалізації в поточному мікрокомандному циклі, зчитується з пам'яті у регістр мікрокоманд. ДШМО декодує код мікрооперації і формує керуючі слова, які ініціюють виконання мікрокоманди в операційному блоці. АНМК може вказуватися в мікрокоманді явно або формуватися природно, як це має місце при вибірці команд. Після виконання вибраної мікрокоманди мікрокомандний цикл повторюється.

Основними питаннями при проектуванні мікропрограмного ПК, які доводиться вирішувати з метою досягнення оптимальних параметрів ПК, є:

1) аналіз способів формування наступної адреси мікрокоманди;

2) вибір способу кодування мікрокоманд;

3) визначення суміщення в часі циклів вибірки і виконання мікрокоманд;

4) вибір типу синхронізації при формуванні мікрооперації.

Спосіб адресації мікрокоманд визначає правило формування адреси наступної мікрокоманди. У мікропрограмних автоматах використовується два основні способи адресації: примусовий і природний методи.

Примусова адресація зводиться до вказівки в кожній мікрокоманді адреси наступної мікрокоманди, а при природній адресації – адреса наступної мікрокоманди утворюється шляхом приросту адреси попередньої, як це має місце при формуванні наступної команди. Природна адресація дозволяє за рахунок виключення поля адреси з операційних мікрокоманд зменшити розрядність пам'яті.

Щодо способів кодування мікрокоманд у мікропрограмних автоматах, то можна виділити такі три способи: горизонтальне, вертикальне та змішане кодування.

Горизонтальний спосіб кодування є простим варіантом кодування, при якому кожен розряд поля коду мікрооперації однозначно визначає керуючий сигнал для виконання мікрооперації. У випадку великого набору мікрооперацій (10 – 100) горизонтальне кодування може вимагати великої розрядності пам'яті мікрокоманд. З іншого боку, за рахунок обмежень на сумісність мікрооперацій у більшості мікрокоманд лише невелика кількість розрядів у полі КМО міститиме одиниці, в основному ж мікрокоманда складатиметься з нулів. Це призводить до неефективного використання пам'яті мікропрограм.

Перевагою горизонтального кодування є можливість паралельної роботи декількох операційних блоків, що дозволяє істотно підвищити швидкодію і зумовлює високий ступінь завантаження устаткування, а також простоту формування ВКС.

При вертикальному кодуванні мікрооперація визначається не станом одного з розрядів мікрокоманди, а двійковим кодом, що

міститься в операційній частині мікрокоманди. Кількість розрядів операційної частини мікрокоманди визначається як: m=]$\log_2$ n[. Звідси видно, що основною перевагою є невелика довжина мікрокоманди, що спричинює скорочення ємності ПМК. Проте в цьому випадку потрібні складні дешифратори на велику кількість мікрооперацій, збільшуються часові витрати на дешифрацію, а головне – в кожній мікрокоманді вказується лише одна мікрооперація, що призводить до збільшення довжини мікропрограм у порівнянні з горизонтальним кодуванням.

Розвитком способів кодування мікрокоманд з метою усунення основних недоліків, властивих горизонтальному і вертикальному способам, є горизонтально-вертикальне, або змішане, кодування мікрокоманд.

Практична частина

Реалізуємо керуючий автомат цифрового операційного пристрою (арифметично-логічного), який ми почали проектувати у попередній лабораторній роботі № 4. Керуючий автомат побудуємо у вигляді мікропрограмного автомата із програмованою логікою (природна адресація мікрокоманд та вертикальне кодування мікрооперацій).

Для цього необхідно, перш за все, визначити формати мікрокоманд. Оскільки ми проектуємо мікропрограмний автомат із природною адресацією, то слід виділити операційні та адресні мікрокоманди. Для реалізації пристрою оберемо 8-розрядну мікросхему пам'яті мікрокоманд. Формати мікрокоманд зображено на рис. 5.3, а схему керуючого автомата – на рис. 5.4.

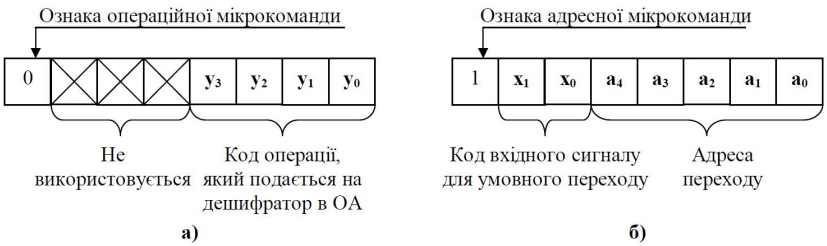


Рис. 5.3. Формати мікрокоманд: **а** – операційна мікрокоманда; **б** – адресна мікрокоманда

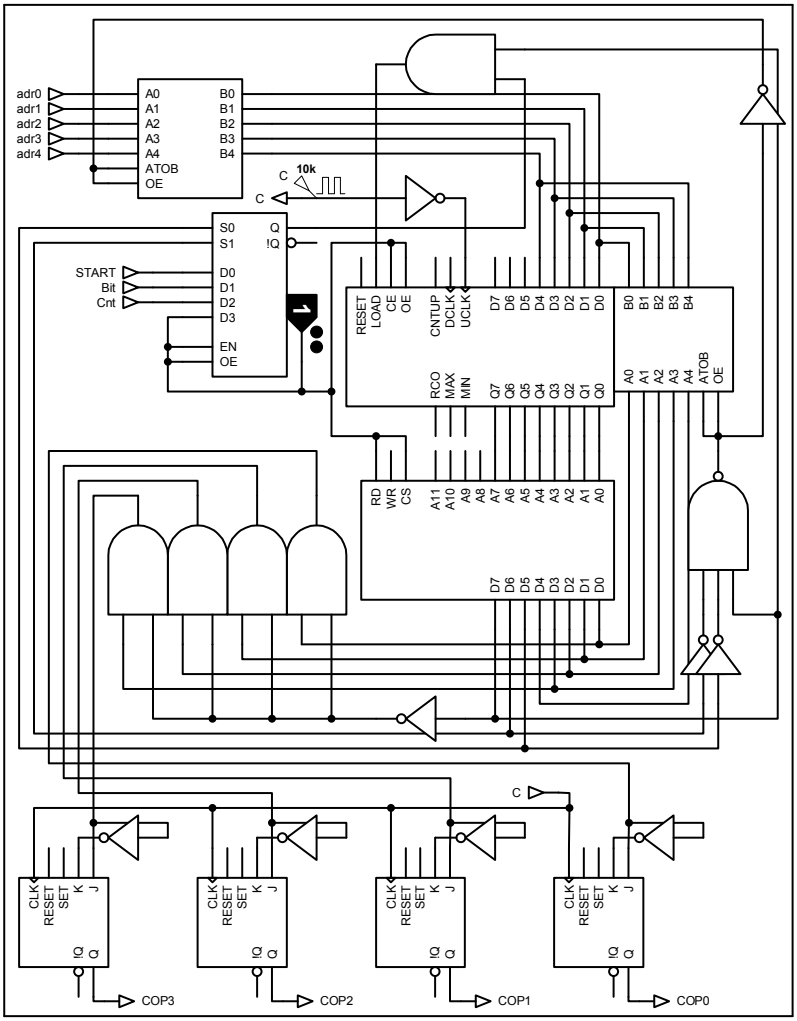


Рис. 5.4. Схема керуючого мікропрограмного автомата

Правильність роботи спроектованої нами схеми цифрового операційного пристрою перевіримо на прикладі виконання операцій множення та віднімання двох цілих двійкових чисел. Насамперед складемо блок-схему алгоритму множення (рис. 5.5).

Рис. 5.5. Блок-схема алгоритму роботи пристрою множення

Використовуючи опис кодів логічних умов та мікрооперацій, наведені у таблицях 4.1 і 4.2 попередньої лабораторної роботи, ми можемо подати алгоритм множення у вигляді мікропрограми для керуючого автомата, доповнивши її мікропрограмою віднімання. У таблиці 5.1 показано вміст пам'яті мікрокоманд (мікропрограма) у двійковому форматі подання кодів мікрокоманд. Дана мікропрограма керує всіма процесами, які відбуваються у схемі цифрового операційного пристрою, тобто визначає логіку його роботи.

Таблиця. 5.1

Мікропрограми операцій множення (3 – 13) та віднімання A=B–A (14 – 22) для мікропрограмного керуючого автомата

№	Код мікро-команди	Призначення мікрокоманди
0	00000000	Немає операції. Заборонити виведення результату операції
1	10000000	Завантаження адреси переходу, якщо натиснуто кнопку START
2	11100000	Безумовний перехід на адресу 0
3	00000001	Завантажити в регістри A і B операнди
4	00000100	Скидання лічильника та регістра-суматора
5	00000101	Збільшення на одиницю (інкремент) вмісту лічильника
6	00001101	Встановити Bit відповідно до молодшого розряду регістра A
7	10101001	Перехід на адресу 9, якщо мол. біт рег. множника дорівнює 0
8	00000010	Додавання до регістра-суматора вміст регістра множеного B
9	00000011	Зсув вправо вмісту регістра A та регістра-суматора
10	11000101	Перехід на адресу 5, якщо вміст лічильника не дорівнює 8
11	00000110	Дозволити виведення результату множення
12	00000110	Вивести результат множення
13	11100000	Безумовний перехід на адресу 0
14	00000000	Немає операції. Заборонити виведення результату операції
15	00000001	Завантажити в регістри A і B операнди
16	00000100	Скидання регістра-суматора
17	00000010	Додавання до регістра-суматора вміст регістра B
18	00001110	Запис операнда із зовнішнього регістра A у внутрішній B
19	00001001	Віднімання від регістра-суматора вміст регістра B
20	00000110	Дозволити виведення результату операції віднімання
21	00001111	Запис із внутрішнього регістра B у зовнішній A
22	11100000	Безумовний перехід на адресу 0

Обирати, яка саме операція буде виконана (множення чи віднімання), слід набором логічних констант для задання адреси початку мікропрограми adr[0..4] (див. рис. 4.2 л. р. № 4). При встановленому значенні 3 ($0011_{(2)}$) буде виконуватись мікропрограма множення, а при встановленні значення 14 ($1110_{(2)}$) – операція віднімання.

Контрольні запитання та завдання

1. Що являє собою мікропрограмний автомат із програмованою логікою? Які переваги та недоліки програмованої („гнучкої”) логіки у порівнянні зі схемною („жорсткою”) логікою?
2. Які способи кодування мікрооперацій ви знаєте. Назвіть їх переваги і недоліки. Як вони впливають на структуру автомата?
3. Які методи адресації мікрокоманд ви знаєте. Вкажіть їх переваги і недоліки. Як вони впливають на структуру автомата?
4. Реалізуйте керуючий мікропрограмний автомат із програмованою логікою та примусовою адресацією мікрокоманд.
5. Реалізуйте керуючий мікропрограмний автомат із програмованою логікою та горизонтальним кодуванням операційних мікрокоманд. Оптимізуйте складені мікропрограми множення та віднімання за рахунок зменшення кількості мікрокоманд.
6. Розробіть мікропрограми для виконання операцій додавання та ділення для спроектованого у практичній частині даної лабораторної роботи керуючого автомата. Результат операції додавання повинен заноситись у зовнішній регістр A. Результат операції ділення – ціле 8-розрядне число в зовнішньому регістрі A та 8-розрядний цілочисловий залишок від ділення у зовнішньому регістрі B.
7. Реалізуйте повноцінний арифметично-логічний цифровий операційний пристрій, який може виконувати операції додавання, віднімання, множення та ділення двох восьмирозрядних цілих чисел без знака. Для цього необхідно у пам’ять мікрокоманд занести всі чотири мікропрограми відповідних операцій та доповнити схему вказівником адрес початку мікропрограм замість наборів логічних констант adr[0..4] (див. рис. 4.2 л. р. № 4).

Лабораторна робота № 6
Проектування елементарної мікропроцесорної системи нейманівської архітектури

Теоретична частина
Принципи побудови та функціонування комп'ютерів

Основні принципи побудови комп'ютерів виклали в 1946 р. американські математики Дж. фон Нейман, К. Голдстайн і А. Беркс. Сукупність цих принципів породила класичну нейманівську архітектуру, яка зберігає актуальність і сьогодні.

Загалом нейманівська архітектура має такі основні ознаки:

- наявність одного обчислювача, що має процесор, пам'ять, засоби введення-виведення інформації та керування;
- використання двійкової системи числення як для подання інформації, так і для виконання арифметико-логічних операцій;
- розміщення в єдиній спільній пам'яті команд і чисел фіксованої довжини;
- лінійну структуру адресації комірок пам'яті, що вимагає наявності в процесорі лічильника команд;
- централізоване автоматичне послідовне зчитування команд із пам'яті та інтерпретацію їх процесором; дані обробляються паралельно – одночасно над усіма розрядами машинного слова;
- низький рівень машинної мови.

Комп'ютер класичної архітектури вміщує (рис. 6.1):

- арифметико-логічний пристрій (АЛП);
- оперативну пам'ять (ОП);
- засоби зберігання і введення-виведення інформації: зовнішні запам'ятовуючі пристрої (ЗЗП); пристрої введення інформації (ПВв); пристрої виведення інформації (ПВив); усі ці пристрої називають зовнішніми або периферійними (ПП);
- пристрій керування (ПК). Разом з АЛП він утворює процесор. При наявності в машині декількох процесорів виділяють центральний (ЦП).

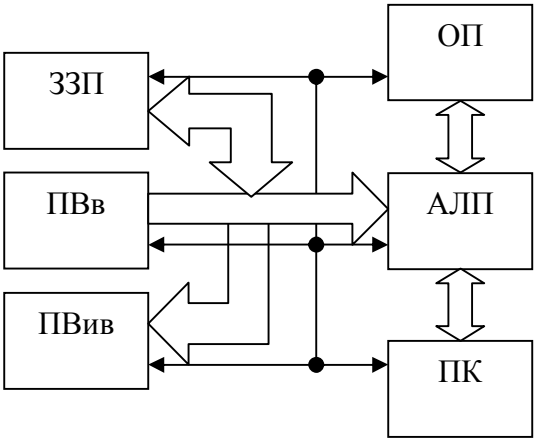


Рис. 6.1. Спрощена структура комп'ютера

Арифметично-логічний пристрій призначений для виконання арифметичних і логічних операцій, передбачених системою команд даного комп'ютера. До складу АЛП входять регістри і комбінаційні схеми. Дані для обробки в АЛП надходять з ОП і називаються операндами. Результати операцій пересилаються в ОП або тимчасово зберігаються в регістрах.

Пристрій керування зчитує і дешифрує у відповідній послідовності команди, формує й подає керуючі сигнали для інших пристроїв комп'ютера.

Оперативна пам'ять призначена для тимчасового зберігання програм і даних, в ній виконуються операції записування і читання інформації. Крім ОП, використовують також постійну пам'ять, в якій здійснюються тільки операції читання. Оперативну (ОЗП) і постійну пам'ять (ПЗП) та регістри АЛП називають внутрішньою пам'яттю. Процесор і ОП разом створюють ядро комп'ютера.

Операції введення-виведення – це обмін інформацією між ядром машини і ПП. Операція введення передає інформацію з ПП в ядро комп'ютера, а операція виведення – навпаки.

Зовнішні запам'ятовуючі пристрої призначені для тривалого і енергонезалежного зберігання великих об'ємів інформації. Фізично її реалізують у вигляді накопичувачів.

Пристрої введення і виведення інформації (ПВВ) розглядають як єдину функціональну частину комп'ютера. Різні за своїми функціями, принципами побудови та характеристиками ПВВ і ЗЗП разом створюють групу дуже різноманітних зовнішніх або периферійних пристроїв.

Зв'язок між функціональними частинами машини здійснюють за допомогою ***інтерфейсу*** – сукупності шин, сигналів, допоміжних мікросхем та алгоритмів, призначених для обміну інформацією між пристроями комп'ютера.

Виділяють три шини (рис. 6.2):

- адреси (ША), призначена для передачі адреси комірок ОП і регістрів ПП;
- даних (ШД), призначена для передачі даних;
- керування (ШК), призначена для передачі керуючих сигналів від процесора до пристроїв і навпаки.

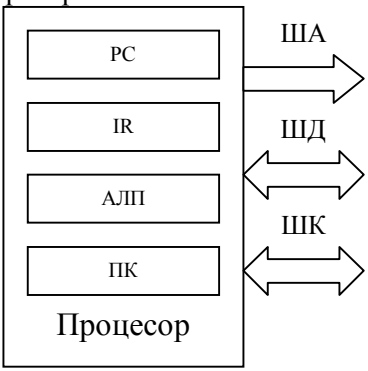


Рис. 6.2. Спрощена структура процесора

Принцип програмного керування

У комп'ютері реалізують принцип програмного керування, суть якого така. Для розв'язання кожної задачі розробляють алгоритм на основі числових методів обчислення. Алгоритм переводиться на мову, властиву даній машині, у вигляді ***програми*** – мовної конструкції, яка є впорядкованою послідовністю описів і команд, призначених для обробки інформації. Кожна команда визначає дії комп'ютера щодо виконання будь-якої операції, які реалізують апаратні (технічні) і

програмні засоби. Програма записується в ОП у вигляді машинних слів, які кодуються цифрами 0 і 1 та розрізняються тільки способом використання. Код операції надходить у регістр команд IR (instruction register) і потім дешифрується, а дані – в регістри АЛП (рис. 6.2).

Команди програми розміщені в ОП лінійно (одна за одною) і виконуються послідовно. Номер команди в ОП визначається програмним лічильником PC (program counter), який іноді називають вказівником команд IP (instruction pointer). Пристрій керування (ПК) виробляє множину керуючих сигналів, які подаються на всі пристрої машини. Регістр команд, програмний лічильник і керуючий автомат входять до складу ПК. Послідовне керування зумовлене наявністю одного процесора. Команди умовного й безумовного розгалуження змінюють лінійний порядок зчитування і виконання команд.

Множина всіх операцій, що реалізуються в комп'ютері, складає його ***операційні ресурси***. Комп'ютери, операційні ресурси яких забезпечують виконання будь-якого алгоритму обробки інформації, називають ***універсальними***. Для цього теоретично достатньо мати в операційних ресурсах тільки чотири операції: пересилку слова між будь-якими комірками ОП, додавання (віднімання) одиниці до слова, умовний перехід за збігом слів та безумовну зупинку комп'ютера. Проте в комп'ютерах операційні ресурси складаються з десятків і сотень команд, що спрощує програмування.

Загалом у комп'ютерах використовують список команд, що забезпечує виконання таких груп операцій:

- пересилки даних між регістрами АЛП, регістрами і ОП;
- арифметичних операцій над двійковими числами із фіксованою та плаваючою комою: додавання, віднімання, знакового і беззнакового множення і ділення;
- логічних операцій заперечення, диз'юнкції, кон'юнкції, додавання за модулем два;
- установлення відношень – більше, менше, нерівно, більше-рівно та ін.;
- зсуву вліво чи вправо – арифметичного, логічного, циклічного;

- керування програмою: умовними та безумовними переходами та викликами процедур, безумовними та умовними поверненнями з процедур, перериванням програми; деякі комп'ютери мають спеціальні команди для організації циклів;
- введення-виведення даних між ядром машини і ПП;
- спеціальних операцій для машин із співпроцесорами (математичними розширювачами): обчислення квадратного кореня, синуса, косинуса, логарифмічні та ін.;
- перетворення з одного формату в інший (наприклад, із 8-бітного в 16-бітний або в двійково-десятковий формат);
- системних операцій – завантаження службових регістрів, захист пам'яті;
- мультимедійних операцій для виконання дій зі звуком, графікою, зображенням.

Зі зростанням продуктивності процесора збільшується і кількість команд.

Практична частина

Спроектуємо елементарну 8-розрядну мікропроцесорну систему (МПС), яка буде містити арифметично-логічний пристрій, розроблений у лабораторних роботах № 4, 5. У своєму складі МПС буде містити:

1. Оперативний запам'ятовуючий пристрій, в якому буде розміщена деяка програма.

2. Мікропроцесорну частину, яка буде складатися із:

а) програмного лічильника;

б) регістра коду команди;

в) блока синхронізації;

г) блока дешифрації коду команди;

д) блока керування виконанням команди;

е) арифметично-логічного блока з регістрами A і B;

ж) блоку додаткових регістрів C і D для лічби та індексації.

3. Набір семисегментних світлодіодних індикаторів, приєднаних до регістрів для візуалізації процесу роботи МПС.

Система буде мати можливість виконання таких команд: NOP (0); MOV register, data (1); MOV register, [address] (2);

MOV [address], register (2); MOV register, [address + D] (3); MOV [address + D], register (3); INC register (4); DEC register (5); ADD register1, register2 (6); SUB register1, register2 (7); MUL (8); DIV (9); JMP address (10); JZ address (11); JS address (12).

Для простоти побудови схеми МПС та незмінності структури АЛП, розробленої у попередніх лабораторних роботах, накладемо деякі обмеження на функціональні можливості команд:

1) команди пересилки MOV працюють зі всіма регістрами;
2) команди INC, DEC працюють лише з регістрами C і D;
3) арифметичні команди працюють лише з регістрами A і B;
4) команда JZ аналізує лише вміст регістра лічби C;
5) команда JS аналізує лише біт знака регістра-акумулятора A.

Відзначимо, що команди в нашому випадку можуть бути як однобайтні, наприклад ADD A, B або INC A, так і двобайтні, наприклад JMP address або MOV A, data, у яких другий байт команди – це або адреса, або деяке числове значення.

Оберемо такий формат коду команди (рис 6.3): молодші 4 розряди (0 – 3) міститимуть код операції (номери, наведені у круглих дужках системи команд), розряд 4 визначатиме напрямок передачі даних (наприклад, 0 – зчитування, 1 – запис), розряди 5 – 6 визначатимуть регістр (наприклад, 00 – A, 01 – D, 10 – C, 11 – B), старший розряд 7 визначатиме довжину команди (наприклад, 0 – однобайтна, 1 – двобайтна команда).

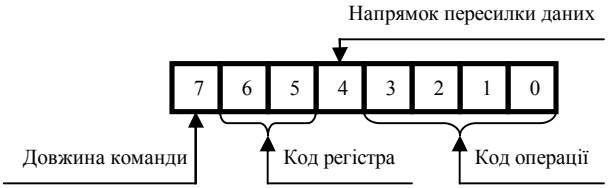


Рис. 6.3. Формат коду команди

Для аналізу працездатності спроектованої мікропроцесорної системи складемо деяку програму (використовуючи набір команд МПС). Для цього складену програму, закодовану відповідно до таблиці кодування команд, буде занесено в оперативний запам'ятовуючий пристрій. Іншими словами, компіляцію програми виконаємо вручну.

На рис. 6.4 зображено загальну схему проектованої мікропроцесорної системи. Блок керування мікропроцесора (підсхема SUB4, рис. 6.5) зчитує та виконує команди програми, що міститься в оперативному запам'ятовуючому пристрої (справа від SUB4). Стан мікропроцесора (вміст регістрів загального призначення) висвітлюється індикаторами, що дозволяє візуально спостерігати за роботою мікропроцесорної системи.

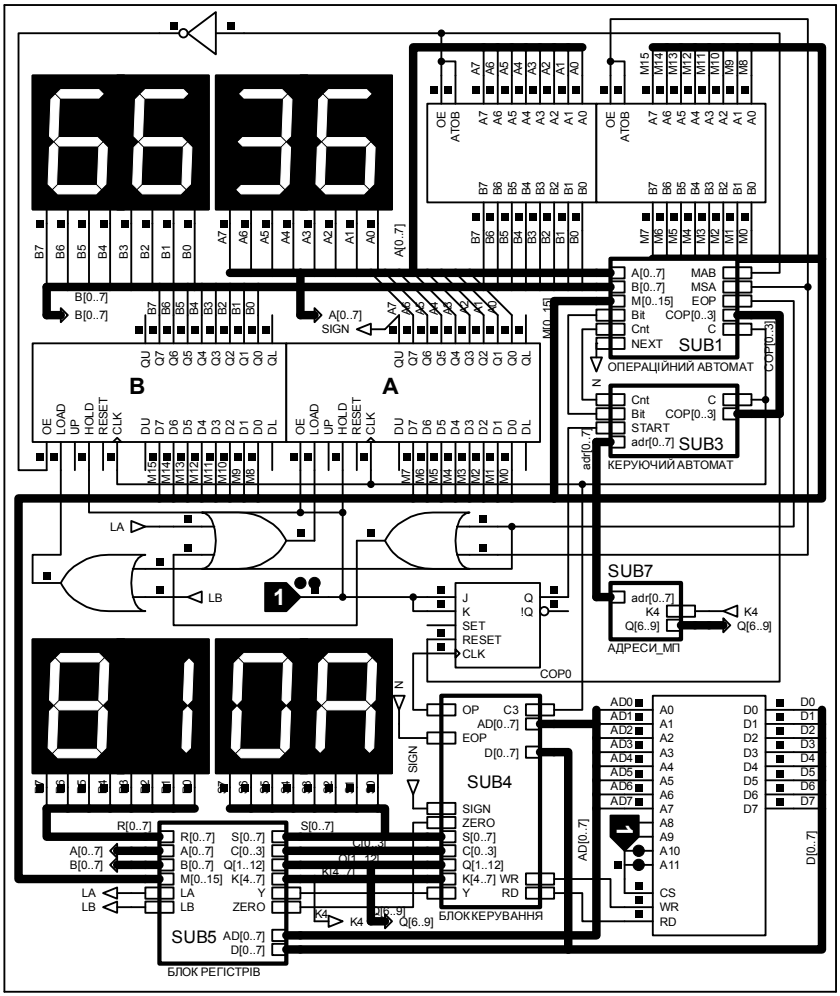


Рис. 6.4. Загальний вигляд схеми мікропроцесорної системи

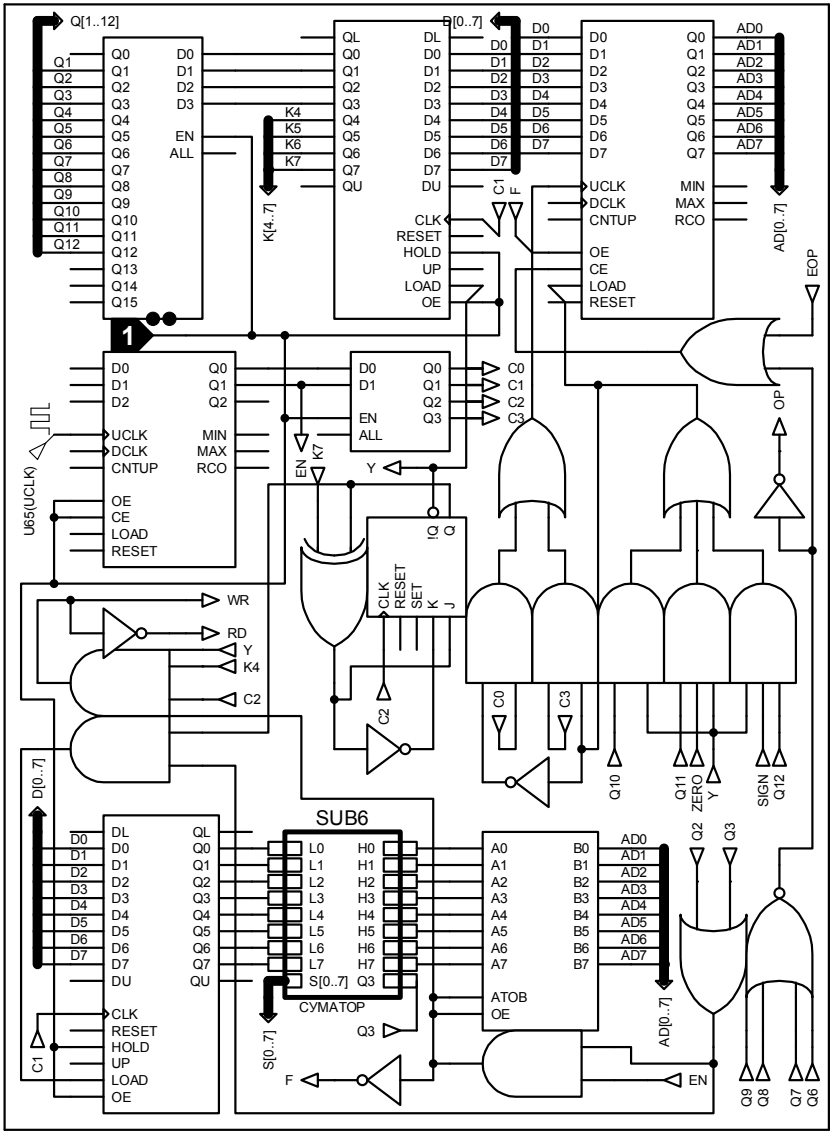


Рис. 6.5. Вміст SUB4 – блок керування МПС
(генератор синхроімпульсів перенесено сюди з керуючого автомата)

На рис. 6.5 зображена схема блоку керування проектованої МПС. Тут можна виділити: блок генерації синхросигналів (лічильник із дешифратором із виходами C0 – C3), блок формування керуючих сигналів (дешифратор із виходами Q0 – Q12 та схема керування виконанням команд із виходом Y), програмний лічильник (схема з виходами AD0 – AD7), регістр коду команди (схема з виходами K0 – K7), блок формування адреси операндів з опосередкованою адресацією (схема, що містить допоміжний регістр, приєднаний до суматора SUB6).

На рис. 6.6 зображено вміст підсхеми SUB6 – суматор для формування адреси операндів при опосередкованій адресації. Тут при виконанні команд MOV register, [address + D] або MOV [address + D], register (при активному сигналі Q3) вміст індексного регістра D додається до деякої базової адреси, записаної у допоміжний регістр (відносно-індексна адресація).

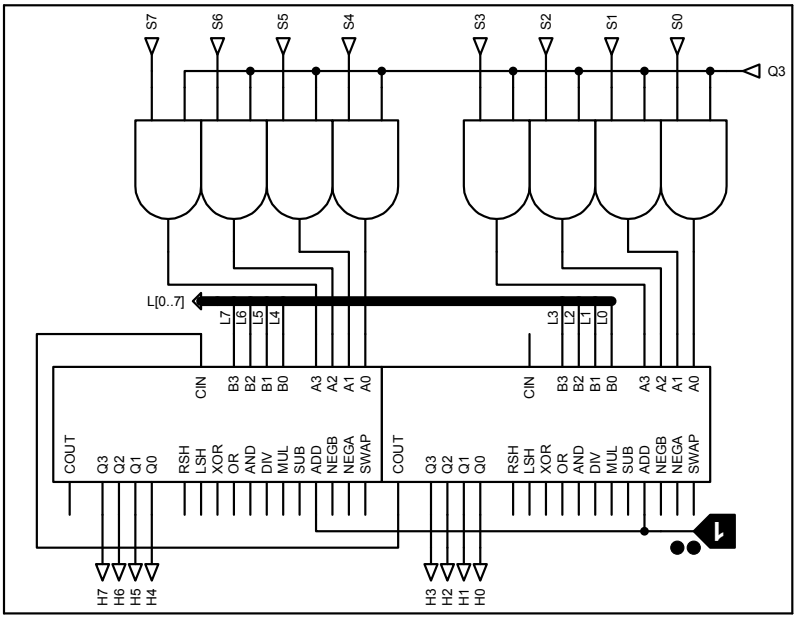


Рис. 6.6. Вміст SUB6 – суматор для формування адреси операндів при опосередкованій адресації

Зображена на рисунку 6.5 організація блока синхронізації зумовлена необхідністю конвеєрного (послідовного) функціонування частин МПС під час виконання команд. Призначення синхросигналів С0 – С3 таке (рис. 6.7):

С0 – приріст програмного лічильника (Program Counter – PC);

С1 – запис у регістр команд (Instruction Register – IR), якщо скинуто тригер дозволу запису (Y = 1); запис адреси операнда у допоміжний регістр у випадку опосередкованої адресації при встановленому тригері дозволу запису (Y = 0);

С2 – встановлення тригера дозволу запису, якщо в IR записалася двобайтова команда (заборона запису в IR наступному машинному циклі) або скидання тригера дозволу запису, якщо в попередньому машинному циклі він був встановлений; виконання команд запису операндів у пам'ять;

С3 – виконання команди, якщо скинуто тригер дозволу.

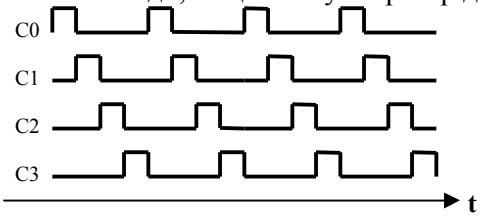


Рис. 6.7. Часові діаграми синхроімпулсів

На рис. 6.7 зображено часові діаграми сигналів синхронізації. З рисунка видно, що на чотирьох окремих лініях синхронізації (С0 – С3), приєднаних до виходів дешифратора (рис. 6.5), почергово з'являються активні сигнали. Таким чином, у кожен момент часу лише одна лінія синхронізації активна, тобто виконується лише одна специфічна мікрооперація.

Опишемо блок керування процесом виконання команд, що містить тригер дозволу запису із виходом Y. Код команди (перший байт у двобайтових командах) записується у регістр команд (IR) і запам'ятовується там упродовж усього циклу виконання команди. Сигнал Y керує процесом дозволу/заборони запису даних у регістр IR, що забезпечує запис у цей регістр ***лише коду команди***, оскільки запис у цей регістр будь-якої іншої

інформації призведе до неправильної її дешифрації і до збою в роботі мікропроцесорної системи.

Однобайтні команди виконуються за один машинний цикл, а для двобайтних необхідні два машинні цикли, причому на першому машинному циклі відбувається завантаження коду команди у регістр команд (IR) та її дешифрація, а на другому машинному циклі – власне виконання команди з використанням операндів або адрес, що з'являються на шині даних. Під час виконання двобайтних команд відповідні керуючі лінії на виході дешифратора коду операції активні протягом двох машинних циклів. Отже, виникає необхідність у проектуванні блока керування процесом виконання команд. Даний блок повинен забороняти виконання двобайтних команд на першому машинному циклі та дозволяти на другому. Це виконується за допомогою генерації керуючого сигналу Y.

Згідно з описаною логікою роботи блока керування процесом виконання команд можна спроектувати його схему, яка являє собою тригер, охоплений зворотним зв'язком у вигляді елемента XOR, на другий вхід якого подано старший розряд коду команди, тобто біт, що визначає кількість байтів у команді (див. рис. 6.5).

Арифметичні команди ADD, SUB, MUL та DIV є однобайтними, але час їх виконання визначається декількома машинними циклами, упродовж яких виконуються відповідні мікропрограми у арифметично-логічному блоці. Під час виконання арифметичних команд приріст програмного лічильника блокується сигналом OP (operation). По завершенні виконання мікропрограми генерується сигнал EOP (end operation), програмний лічильник розблоковується та розпочинається виконання наступної команди.

На рис. 6.8 показано вміст підсхеми SUB5 – блок допоміжних регістрів (регістра-лічильника C та індексного регістра D), що містить також допоміжні буферні схеми, які служать для передачі операндів між пам'яттю та регістрами арифметично-логічного блока A і B.

Рис. 6.9 демонструє вміст підсхеми SUB7 – блок формування адреси мікропрограми для відповідної арифметичної команди.

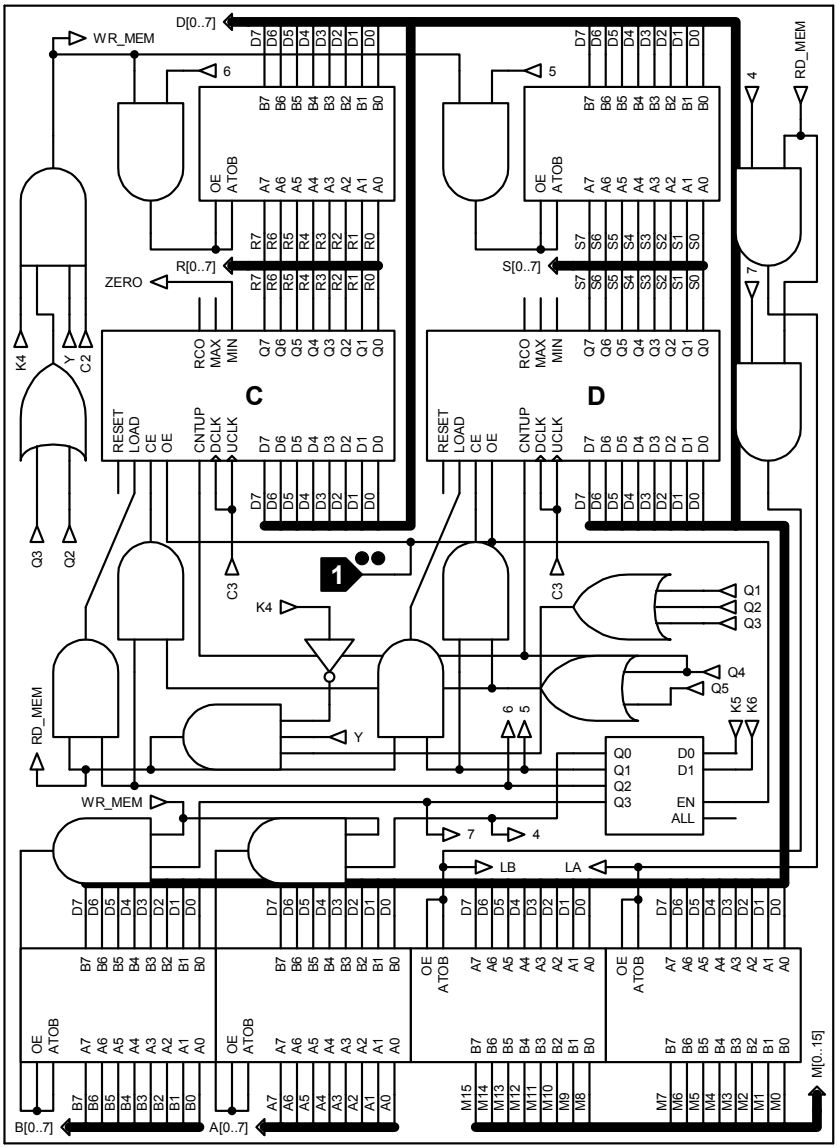


Рис. 6.8. Вміст SUB5 – блок допоміжних регістрів (регістра-лічильника C та індексного регістра D), що містить також допоміжні буферні схеми (у регістрах-лічильниках C і D встановіть Use CNTUP Direction Control)

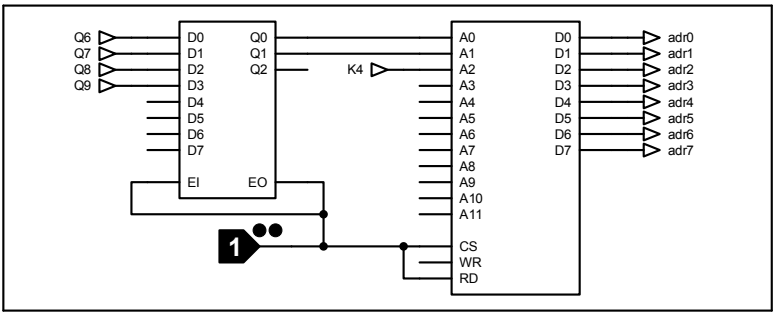


Рис. 6.9. Вміст SUB7 – блок формування адреси мікропрограми АЛП за кодом відповідної арифметичної команди

Для аналізу працездатності спроектованої мікропроцесорної системи складемо деяку програму, яка б сортувала елементи масиву згідно з ознакою парності, причому по закінченні роботи програми, на початку масиву повинні бути розташовані непарні числа, а в кінці – парні. Програма має такий вигляд:

Мнемонічний запис програми	*Адреси*	*Двійкове зображення команд*	*Коментарі*
JMP begin	0	10001010	Безумовний перехід
	1	00001101	Адреса переходу
array DB 3,6,9,2,8,4,5,7,1,3	2	00000011	Масив чисел: 3
	3	00000110	6
	4	00001001	9
	5	00000010	2
	6	00001000	8
	7	00000100	4
	8	00000101	5
	9	00000111	7
	10	00000001	1
	11	00000011	3
help_var DB 0	12	00000000	Допоміжна комірка пам’яті
begin: MOV C, 9	13	11000001	Занесення у регістр С числа 9
	14	00001001	(к-ть елементів array мінус 1)
MOV D, 0	15	10100001	Занесення у регістр D
	16	00000000	числа 0
next: MOV A, array[D]	17	10000011	Занесення у регістр А вмісту
	18	00000010	комірки за адресою [array+D]
MOV help_var, A	19	10010010	Занесення у допоміжну
	20	00001100	комірку пам’яті вмісту рег. А
MOV B, 2	21	11100001	Занесення у регістр В числа 2
	22	00000010	(для визначення А mod 2)

DIV	23	00001001	Ділення вмісту A на B
MOV A, 128	24	10000001	Занесення у регістр A числа
	25	10000000	128 (одиниця у 7-му розряді)
MUL	26	00001000	Множення вмісту A на B
JS n_xch	27	10001100	Перехід, якщо SIGN = 1
	28	00110101	Адреса переходу
MOV A, array[D+1]	29	10000011	Занесення у A вмісту комірки
	30	00000011	за адресою [array + D + 1]
MOV B, 2	31	11100001	Занесення у регістр B числа 2
	32	00000010	(для визначення A mod 2)
DIV	33	00001001	Ділення вмісту A на B
MOV A, 128	34	10000001	Занесення у регістр A числа
	35	10000000	128 (одиниця у 7-му розряді)
MUL	36	00001000	Множення вмісту A на B
JS ljns	37	10001100	Перехід, якщо SIGN = 1
	38	00101001	Адреса переходу
JMP n_xch	39	10001010	Безумовний перехід
	40	00110101	Адреса переходу
ljns: MOV A, array[D+1]	41	10000011	Занесення у A вмісту комірки
	42	00000011	за адресою [array + D + 1]
MOV array[D], A	43	10010011	Занесення вмісту регістра A у
	44	00000010	комірку за адресою [array+D]
MOV A, help_var	45	10000010	Занесення у регістр A вмісту
	46	00001100	допоміжної комірки пам'яті
MOV array[D+1], A	47	10010011	Занесення вмісту A у комірку
	48	00000011	за адресою [array + D + 1]
INC C	49	01000100	Збільшення на одиницю C
DEC D	50	00100101	Зменшення на одиницю D
JMP next	51	10001010	Безумовний перехід
	52	00010001	Адреса переходу
n_xch: INC D	53	00100100	Збільшення на одиницю D
DEC C	54	01000101	Зменшення на одиницю C
JZ end	55	10001011	Перехід, якщо ZERO=1 (C=0)
	56	00111011	Адреса переходу
JMP next	57	10001010	Безумовний перехід
	58	00010001	Адреса переходу
end: MOV help_var, C	59	11010010	Занесення у допоміжну
	60	00001100	комірку пам'яті вмісту C (0)
fin: NOP	61	00000000	Порожня операція
JMP fin	62	10001010	Безумовний перехід
	63	00111101	Адреса переходу

Наведена програма сортує елементи масиву *array*, який являє собою сукупність із десяти чисел, а саме: 3, 6, 9, 2, 8, 4, 5, 7, 1, 3.

Переконайтеся у правильності роботи складеної програми. Коли процесор виконуватиме порожній цикл (fin: NOP JMP fin) елементи масиву *array* будуть розташовані в такому порядку: 3, 9, 5, 7, 1, 3, 6, 2, 8, 4, тобто перші шість елементів – непарні числа, а останні чотири – парні, як показано на рис. 6.10.

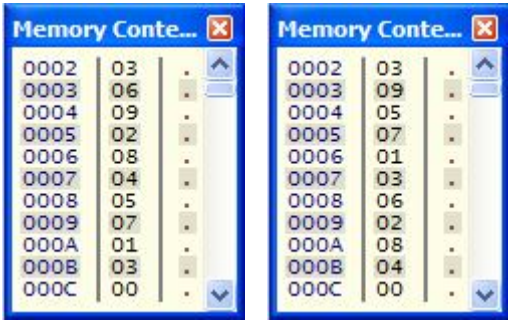


Рис. 6.10. Розташування елементів масиву в пам'яті до (зліва) і після (справа) роботи програми

Контрольні запитання та завдання

1. Які нейманівські принципи побудови та функціонування комп'ютерів?
2. Опишіть спрощену структуру комп'ютерів і вкажіть призначення основних пристроїв, які входять до їх складу. Які шини комп'ютерів ви знаєте? Опишіть їх призначення.
3. Назвіть основні етапи автоматичного виконання команд програми процесором.
4. На які групи можна поділити систему команд процесорів?
5. Модифікуйте мікропроцесорну систему, розроблену в даній лабораторній роботі в такий спосіб, щоб вона мала можливість виконання команд: INC A, DEC A, INC B, DEC B.
6. Використовуючи будь-яку зручну для вас мову програмування розробіть програму-транслятор, яка б перетворювала мнемонічний запис команд мікропроцесорної системи в їх двійкове зображення. Після цього вам буде зручно розробляти програмне забезпечення для спроектованої МПС, записуючи команди у мнемонічному вигляді, а не у двійковому.
7. Реалізуйте свій варіант деякої алгоритмічної задачі на основі мікропроцесорної системи, розробленої в даній лабораторній роботі. Наприклад, обчислення суми елементів масиву, пошук максимального та мінімального елементів масиву тощо. Опишіть функціонування МПС, якщо в програмі, наведеній у даній роботі, видалити початкову команду JMP begin.

Лабораторна робота № 7
Дослідження програмної моделі 16-розрядного мікропроцесора 8086 фірми Intel, способів адресації операндів та системи команд

Теоретична частина

Сімейство процесорів Intel x86

Першим представником сімейства Intel x86, або, згідно з офіційною класифікацією фірми Intel (Integrated Electronics, США), сімейства процесорів IA (Intel Architecture), є мікропроцесор 8086, розроблений до 1978 року. Програми, написані для нього, виконуються на всіх подальших процесорах сімейства, включаючи всі сучасні моделі. Попередні процесори – 8080, 8085 і 4004 (розробка 1967 р.) – через несумісність за об'єктним кодом залишаються поза сімейством, будучи, проте, важливими етапами на шляху розвитку Intel x86. На сьогоднішній день процесори цього сімейства де-факто стали стандартом для більшості персональних комп'ютерів у всьому світі.

Структура процесорів 8086

Мікропроцесор 8086 орієнтований на виконання команд паралельно з їх вибіркою і може бути умовно розділений на дві частини, що працюють асинхронно (рис. 7.1): пристрою сполучення із зовнішніми шинами (ПС) і пристрою обробки (ПО). Пристрій сполучення забезпечує формування 20-розрядної фізичної адреси пам'яті, вибірку команд і операндів із пам'яті, організацію черговості команд і запам'ятовування результатів виконання команд у пам'яті. До складу ПС входять шість 8-розрядних регістрів черги команд, чотири 16-розрядні сегментні регістри, 16-розрядний регістр обміну і 16-розрядний суматор адреси, інтерфейс із зовнішніми шинами. Регістри черги команд організовані за принципом FIFO «першим прийшов – першим вийшов». ПС готове виконати цикл вибірки 16-розрядного слова з пам'яті всякий раз, коли в черзі звільняються, щонайменше, два байти, а ПО витягує з черги команди по мірі їх виконання. При виконанні команд передачі управління, наприклад умовних і

безумовних переходів, черга очищається і починає заповнюватися наново.

Пристрій обробки призначений для виконання операцій по обробці даних і складається з пристрою мікропрограмного керування (ПМК), 16-розрядного АЛП, восьми 16-розрядних регістрів загального призначення і регістра ознак. Команди з черги, сформованої ПС, надходять у ПМК, де декодуються і виконуються в 16-розрядному АЛП згідно із процедурами, записаними у пам'яті мікропрограм. Послідовне виконання команд забезпечується селектором команд, частина якого (регістр лічильника команд IP) зображена у складі ПС, оскільки саме ПС записує в IP адресу зміщення наступної команди, тобто положення нової команди відносно початку сегмента команд. ПО обмінюється даними з ПС через внутрішню 16-розрядну шину і регістр обміну (рис. 7.1).

Регістри процесора 8086

Мікропроцесор 8086 має 12 програмно-доступних шістнадцятирозрядних регістрів (рис. 7.1), регістр лічильника команд IP (Instruction Pointer) і регістр прапорців (або регістр стану процесора) FLAGS. Серед програмно-доступних регістрів виділяють такі групи:

- Регістри даних: AX – акумулятор (Accumulator); BX – базовий регістр (Base); CX – регістр лічильника (Counter); DX – регістр даних (Data), які, в свою чергу, складаються з двох 8-бітових половин, до яких можна незалежно звертатися по іменах AH, BH, CH, DH – старші байти і AL, BL, CL, DL – молодші байти.

- Регістри-вказівники (індексні регістри): SI – індекс джерела (Source Index); DI – індекс приймача (Destination Index); BP – вказівник бази (Base Pointer); SP – вказівник стека (Stack Pointer).

- Сегментні регістри: SS – сегмент стека (Stack Segment); DS – сегмент даних (Data Segment); ES – додатковий сегмент (Extended data Segment); CS – сегмент коду (Code Segment).

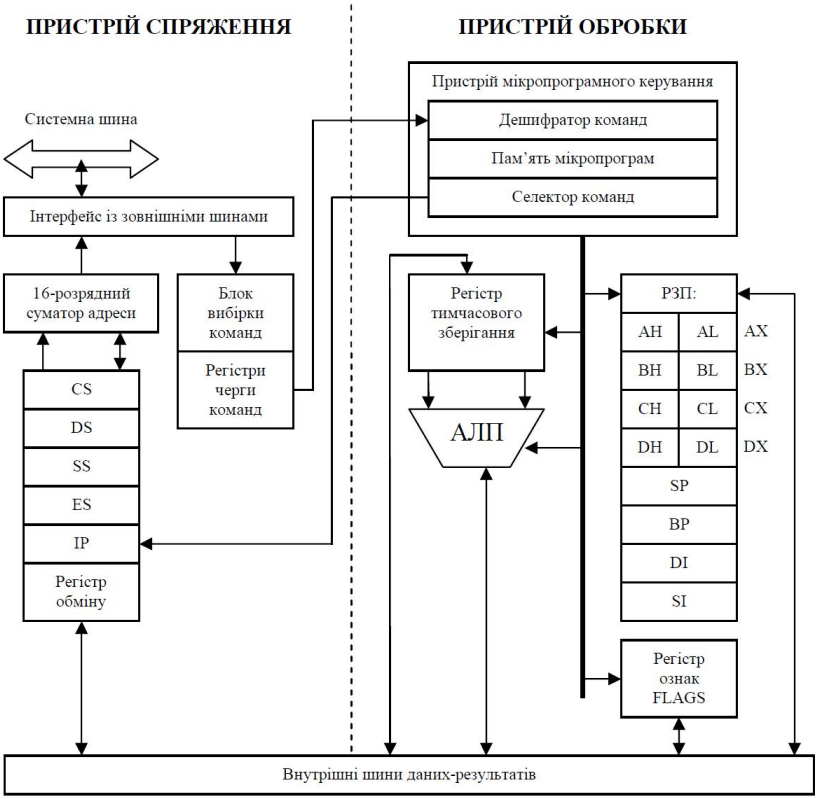


Рис. 7.1. Структура процесора 8086

Біти (або прапорці) регістра ознак FLAGS розділяються на умовні прапорці, які відображають результат попередньої операції АЛП, і керуючі прапорці, від яких залежить виконання спеціальних функцій.

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
				OF	DF	IF	TF	SF	ZF		AF		PF		CF

Рис. 7.2. Прапорці процесора 8086

Прапорці TF, IF, DF відносять до керуючих прапорців, інші – до умовних (прапорам стану). Регістр прапорців умов має 16

бітів. Кожен біт називається прапорцем (flag) і може набувати значення 1 або 0. Прапори відображають такі умови:

- Carry Flag (CF) (прапорець корекції (перенесення/позики)). Встановлюється при виході результату додавання (віднімання) беззнакових операндів за межі діапазону. У командах зсуву прапорець CF фіксує значення старшого біта.
- Zero Flag (ZF) (прапорець нульового результату). Встановлюється при отриманні нульового результату операції.
- Sign Flag (SF) (прапорець знака). Дублює значення старшого біта результату; SF=0 для додатних чисел і SF=1 для від'ємних чисел.
- Overflow Flag (OF) (прапорець переповнення). Встановлюється при виході знакового результату за межі діапазону.
- Parity Flag (PF) (прапорець паритету (парності)). Встановлюється при парній кількості одиниць у результаті.
- Auxiliary Flag (AF) (прапорець допоміжного перенесення/позики з молодшої тетради в старшу). Використовується при десятковій арифметиці.
- Interrupt enable Flag (IF) (прапорець дозволу переривань). При IF=1 дозволяється виконання маскованих апаратних переривань.
- Direction Flag (DF) (прапорець керування напрямом у рядкових операціях). При DF = 1 індексні регістри SI, DI, що беруть участь у рядкових операціях, автоматично декрементуються на кількість байтів операнда, при DF = 0 – інкрементуються.
- Trap Flag (TF) (прапорець трасування (покрокового режиму)). При його встановленні після виконання кожної команди викликається внутрішнє переривання 1 (INT 1).

Організація пам'яті

Хоча процесор має 20-розрядну адресну шину, яка сполучає його з фізичною пам'яттю, він оперує з 16-бітовими логічними адресами, що складаються з 16-розрядної базової адреси сегмента і 16-розрядного зміщення в сегменті. Фізичні, 20-розрядні адреси даних і команд формуються шляхом складання вмісту регістрів-вказівників і зсунутого на 4 біти вліво вмісту сегментних регістрів. Так звана *ефективна адреса* (EA) даних отримується як сума вмісту регістрів BX або BP, вмісту регістрів SI або DI і

зміщення. Потім з ефективної адреси і вмісту сегментного регістра формується фізична адреса (рис. 7.3). У формуванні фізичної адреси команди беруть участь регістри IP і CS. Таким чином, адресний простір розбивається на 4 сегменти місткістю 64К адрес за кількістю сегментних регістрів. Регістр CS вказує на поточний сегмент коду (програми), звідки вибираються команди. Регістр DS вказує на поточний сегмент даних, в якому містяться змінні. Регістр SS адресує поточний сегмент стека, в якому реалізуються всі стекові операції. Нарешті регістр ES визначає поточний додатковий сегмент даних. Зміщений вміст сегментного регістра визначає положення сегмента у 20-розрядному адресному просторі, а регістри-вказівники – положення команди або даних усередині сегментів.

Оскільки при формуванні ефективної адреси вміст сегментного регістра зсувається на 4 біти, сегмент завжди починається з адреси, яка кратна 16, тобто на межі 16-байтового блока пам'яті (параграфа). Сегменти в пам'яті можуть розташовуватися як послідовно, так і з накладанням один на одного. Якщо програма перевищує 64 Кбайт, то необхідно перезавантажувати сегментний регістр CS новим значенням базової адреси. Відповідно, якщо дані перевищують 64 Кбайт, то необхідно перезавантажити регістр DS.

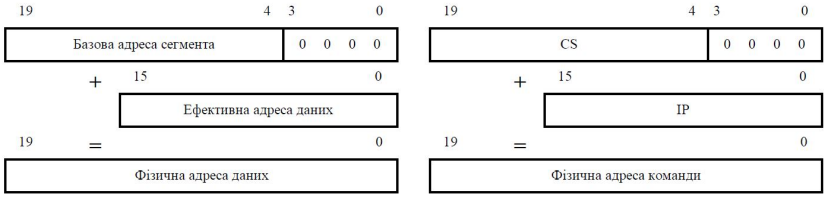


Рис. 7.3. Формування фізичної адреси даних і команди

Пам'ять організована у вигляді одновимірного масиву байтів із фізичними адресами від $00000_{(16)}$ до $FFFFF_{(16)}$. Дві області адресного простору пам'яті зарезервовані для виконання спеціальних функцій, які пов'язані з обробкою переривань і системним скиданням. Цими областями є перші 128 байтів (фізичні адреси 00000 – 0007F) і останні 16 байтів (фізичні адреси

FFFF0 – FFFFF). Задані області використовувати для інших цілей не можна.

Байти в пам'яті організовуються в слова в такий спосіб, що байту, який має меншу адресу, відповідають менш значущі позиції розрядів в слові. Кожен байт або слово пам'яті адресується за допомогою 20-бітової адреси, причому в разі адресації слова адреса вказує на його молодшу частину. Наприклад, адреса $00000_{(16)}$ може позначати і байт із цією адресою, що умовно записується у вигляді [00000] = 34h, і слово з такою ж адресою, що записується у вигляді [00000] = 1234h. Тоді старший байт слова [00001] = 12h. Квадратні дужки позначають елемент пам'яті, адреса якого знаходиться в цих дужках, h – шістнадцятькову систему числення. Команди, байти і слова можна розміщувати за будь-якою адресою байта, однак рекомендується розміщувати слова в пам'яті за парними адресами, оскільки процесор може передавати такі слова за один цикл звернення до пам'яті. Слово з парною адресою називається таким, що вирівнюється на границю слова. Слова з непарними адресами (що не вирівнюються) також допустимі, проте вони зчитуються в два рази повільніше (вимагають два цикли звернення до пам'яті).

Способи адресації операндів

Способи адресації визначаються різними комбінаціями регістрів, символічних імен та виразів, що мають місце при записі операндів, за допомогою яких у командах ідентифікуються дані, що обробляються.

Розрізняють два основних типи операндів: регістровий і операнд в пам'яті, який завжди визначає область пам'яті, де знаходяться дані, що обробляються. Операнд у регістрі завжди визначається ім'ям деякого регістра. Операнди в пам'яті вказуються за допомогою регістрів, символічних імен і констант, які використовуються в різних комбінаціях для ідентифікації даних, що беруть участь у виконанні команди. Різноманітні комбінації цих елементів називаються способами адресації. Кожному способу адресації відповідає певний спосіб вказівки операнда у вигляді комбінації таких чотирьох елементів:

Елемент	*Спосіб подання в команді*
Безпосередні дані	Символічне ім’я або константа
Індексний регістр	Регістр SI або DI
Базовий регістр	Регістр BX або BP
Зміщення	Символічне ім’я або константа

Розглянемо коротко кожен із перерахованих елементів. Безпосередніми є такі дані, які записуються прямо в самому коді команди, як, наприклад, елемент 01 в команді MOV AH, 01. При виконанні цієї команди величина 01 буде завантажуватися в регістр AH.

Наступними двома елементами з наведеного переліку є регістри, які містять адреси. Коли такий регістр використовується для вказівки операнда в пам’яті, він містить не дані, що беруть участь у виконанні команди, а посилання на те місце пам’яті, де зберігаються ці дані.

Останнім елементом є зміщення, яке формою нагадує безпосередні дані, оскільки також задається в команді у вигляді деякої величини. Однак, на відміну від даних, наявних у коді команди, це значення використовується для формування адреси даних, а не як дані. Якщо операнд складається лише зі зміщення, то останнє є просто адресою даних.

Компілятор програм завжди аналізує мнемонічний запис операнда, з тим щоб визначити, до якого способу адресації він належить. Операнд, заданий константою, обробляється при цьому як безпосередні дані, тобто використовується як дані, а не як адреса. Якщо операнд заданий символічним ім’ям, то вважається, що має місце режим прямої адресації. Коли як операнд вказується індексний або базовий регістр (або обидва разом) без додаткових констант і символічних імен, то компілятор обробляє цей операнд відповідно до одного з режимів регістрової адресації. Така адресація часто називається опосередкованою, оскільки звернення до пам’яті в даному випадку здійснюється не напряму, а через регістр. Якщо в записі операнда поряд з індексним або базовим регістром фігурує ще й деяка константа або символічне ім’я, то останні сприймаються як

зміщення, що відповідає режиму прямої адресації з використанням регістрів.

Операнд, що працює з пам'яттю, при виконанні команди, в якій він знаходиться, перетворюється у виконавчу адресу. Ця виконавча адреса додається до адреси сегмента, і в результаті формується абсолютна адреса. Кожна комбінація елементів, яка відповідає певному способу адресації, вказує процесору, як повинна формуватися виконавча адреса.

Мікропроцесор 8086 має такі способи адресації:

1. *Пряма адресація*. При такій адресації адреса операнда вказується безпосередньо в команді.

2. *Регістрова адресація*. В цьому випадку в записі команди вказується позначення регістра загального призначення (РЗП), у якому знаходиться операнд.

3. *Безпосередня адресація*. У цьому режимі в команді вказується 8- або 16-бітовий операнд. Операнд у цьому випадку знаходиться у пам'яті програм.

4. *Базова адресація*. Ефективна адреса операнда (ЕА) обчислюється додаванням вмісту базових регістрів BX або BP зі зміщенням (8- або 16-розрядним знаковим числом). В окремому випадку зміщення може не бути.

5. *Індексна адресація*. При індексній адресації як адреса операндів використовуються вмісти індексних регістрів SI або DI та зміщення у вигляді числа.

6. *Базово-індексна адресація*. Ефективна адреса операнда (ЕА) дорівнює сумі вмісту базових регістрів BX або BP, індексних регістрів SI або DI та зміщення. Числового зміщення може не бути.

Базова та індексна адресації зазвичай застосовуються для звернення до елементів одновимірного масиву, базово-індексна – до двовимірного масиву даних.

Система команд мікропроцесора 8086

Повний перелік команд в алфавітному порядку наведено у відповідній документації, що додається до даної лабораторної роботи в електронному вигляді. Загалом усі команди мікропроцесора 8086 умовно можна поділити на п'ять груп:

1) команди передачі інформації (команди пересилки, роботи зі стеком, введення-виведення);

2) команди обробки інформації (арифметичні, логічні, команди зсуву);

3) рядкові команди;

4) команди передачі керування, включаючи команди переривань;

5) команди керування станом мікропроцесора.

Практична частина

У документації до програми-емулятора мікропроцесора 8086 **Emu8086.exe** наведено повний перелік команд мікропроцесора в алфавітному порядку з типами операндів, описом алгоритмів роботи, впливом команд на прапорці умов та прикладами виконання команд. Дослідіть особливості виконання команд мікропроцесора 8086 шляхом реалізації наведених у документації прикладів у програмі-емуляторі.

Досліджувані приклади виконання команд реалізуйте також на основі моделі мікропроцесора 8086, який є у складі системи схемотехнічного моделювання **Proteus**.

Контрольні запитання та завдання

1. Опишіть структуру мікропроцесора 8086 та його програмну модель.
2. Опишіть призначення регістрів мікропроцесора 8086.
3. Опишіть організацію пам’яті „з точки зору” мікропроцесора 8086.
4. Які способи адресації операндів ви знаєте? Опишіть їх.
5. На які групи можна розділити систему команд мікропроцесора 8086? Наведіть приклади команд із кожної групи.
6. Наведіть приклади виконання команд, які впливають на окремі прапорці умов (CF, OF, PF, SF, ZF, AF). Наведіть приклад команди, результат роботи якої залежить від керуючого прапорця DF.
7. Опишіть цикл функціонування мікропроцесора 8086 в часі.

Лабораторна робота № 8
Реалізація базових алгоритмічних структур та розробка елементарних програм для мікропроцесора 8086

Теоретична частина

Основи програмування мовою асемблер

У мікропроцесорних системах використовується програмування мовою асемблер. Асемблером називається і мова програмування у мнемокодах команд, і спеціальна програма-транслятор, що переводить (транслює) мнемокоди у машинні коди, які зчитуються мікропроцесором із пам'яті програм, дешифруються і виконуються. Процес переведення у машинні коди називається асемблюванням.

Програма на мові асемблер містить два типи виразів:

— команди, що транслюються в машинні коди;

— директиви, що керують ходом трансляції.

Вираз має вигляд

{<мітка>}: <мнемокод> {<операнд>} {,} {<операнд>} {; коментар}

У фігурних дужках наведено елементи виразу, яких може не бути в деяких командах.

Приклади написання простих програм

Прості програми доцільно оформляти у вигляді командних файлів (COM-файли). Першою директивою таких програм є директива ORG 100H, останньою – END.

Приклад 8.1. Написати програму додавання вмісту двох 8-розрядних комірок пам'яті, що знаходяться в сегменті даних DS зі зміщеннями 1000H і 1001H. Результат розмістити у комірці пам'яті з адресою DS:1002H.

У цьому прикладі для простоти не будемо враховувати можливість виникнення перенесення. Програма має вигляд:

```
Мнемокод    Операнд(и)      Коментар
ORG         100H            ; Початок програми
MOV         AL, [1000H]     ; Переслати у 8-розрядний регістр AL вміст комірки пам'яті
                            ; з адресою DS:[1000H]
ADD         AL, [1001H]     ; Додати до вмісту AL вміст комірки DS:[1001H]
MOV         [1002H], AL     ; Переслати вміст AL у комірку DS:[1002H]
END                         ; Завершення програми
```

Відзначимо, що запис MOV AL, [1000H] рівнозначний запису MOV AL, DS:[1000H], оскільки сегмент DS прийнятий за замовчуванням.

Приклад 8.2. Написати програму, яка забезпечує розділення вмісту 16-розрядної комірки пам'яті з адресою ES:[2000H] на чотири тетради. Тетради мають бути записані в молодші частини чотирьох послідовних 8-розрядних комірок пам'яті, починаючи з адреси DS:[1000H], причому старша тетрада має бути записана у комірку зі старшою адресою.

У цьому прикладі для запису результату зручно використовувати непряму адресацію. Програма має вигляд:

```
Мнемокод    Операнд(и)          Коментар
ORG         100H                ; Початок програми
MOV         AX, ES:[2000H]      ; Переслати вміст 16-розрядної комірки ES:[2000H] у
                                ; 16-розрядний регістр AX
MOV         DX, AX              ; Зберегти початкове число в DX
AND         AX, 000FH           ; AX←AX * 0000 0000 0000 1111.
                                ; Виділити молодшу тетраду (скинути всі розряди AX,
                                ; окрім чотирьох молодших)
MOV         SI, 1000H           ; Записати в SI початкову адресу результату
MOV         [SI], AL            ; Переслати у комірку пам'яті з адресою DS:SI
                                ; молодшу 8-розрядну частину AL регістра AX
MOV         AX, DX              ; Переслати початкове число з DX у AX
AND         AX, 00F0H           ; AX←AX*0000000011110000. Виділити другу тетраду
MOV         CL, 4               ; Завантажити у CL кількість розрядів зсуву
ROR         AL, CL              ; Циклічний зсув AL на чотири розряди вправо,
                                ; у результаті виділене чотирирозрядне число
                                ; переміститься в AL
INC         SI                  ; Збільшити SI для запису другого числа
MOV         [SI], AL            ; Запам'ятати другу тетраду у комірці DS:[SI]
MOV         AX, DX              ; Переслати початкове число з DX у AX
AND         AX, 0F00H           ; AX←AX*0000111100000000. Виділити третю тетраду
INC         SI                  ; Збільшити адресу результату
MOV         [SI], AH            ; Запам'ятати третю тетраду
MOV         AX, DX              ; Переслати початкове число з DX у AX
AND         AX, 0F000H          ; AX←AX*1111000000000000. Виділити 4-ту тетраду
INC         SI                  ; Збільшити адресу результату
MOV         CL, 4               ; Завантажити у CL кількість розрядів зсуву
ROR         AH, CL              ; Циклічний зсув CL на чотири розряди вправо
MOV         [SI], AH            ; Запам'ятати четверту тетраду
END                             ; Завершення програми
```

Щоб зменшити громіздкість програми, доцільно зводити її до однотипних кроків і використовувати циклічні операції. Розглянутий приклад можна спростити, якщо виконати зсув 16-розрядного числа так, щоб тетрада, яка виділяється, завжди була

молодшою. Алгоритм такої програми разом із командами показано на рис. 8.1.

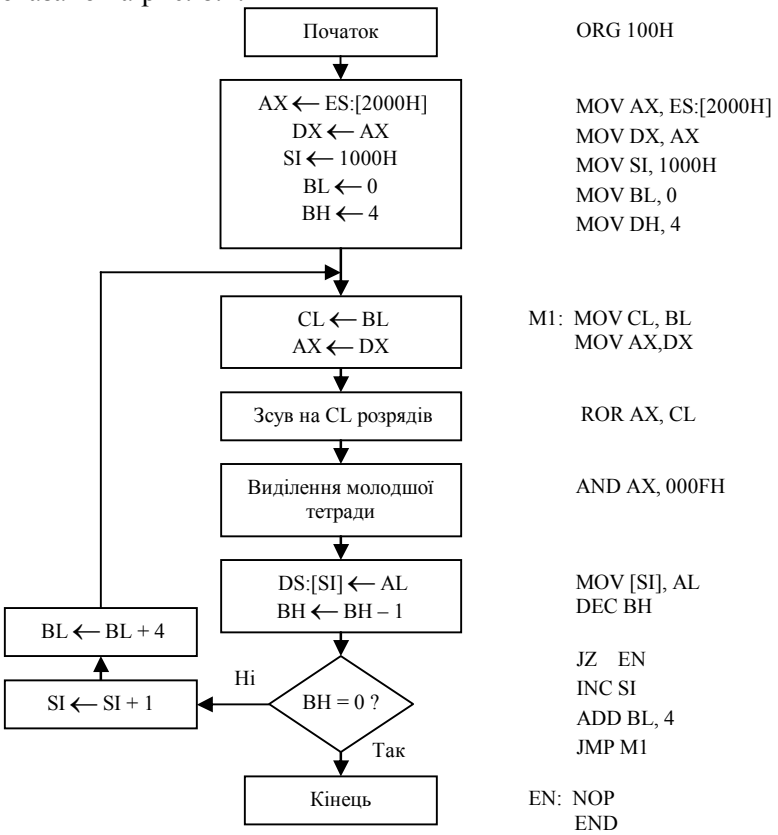


Рис. 8.1. Алгоритм розв’язку задачі прикладу 8.2

Типові обчислювальні процедури

Алгоритм типової обчислювальної процедури ЯКЩО-ТО-ІНАКШЕ показано на рис. 8.2. Ця процедура застосовується тоді, коли треба реалізувати перехід до однієї з двох обчислювальних процедур залежно від умови. Написання програм мовою асемблер виконуються командами переходів за умовами встановлення (скидання) прапорців.

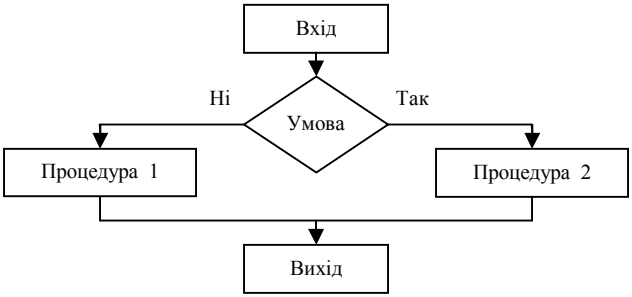


Рис. 8.2. Алгоритм процедури ЯКЩО-ТО-ІНАКШЕ

Приклад 8.3. Написати програму ділення вмісту AX на вміст BL. Результат помістити у 8-розрядну комірку пам'яті з адресою DS:[1000H]. Остачею від ділення знехтувати. Якщо вміст BL = 0, то ділення не виконувати, на місце результату помістити число 0FFH.

Програма має вигляд:

```
Мнемокод    Операнд(и)      Коментар
ORG         100H            ; Початок програми
CMP         BL, 0           ; Порівняти вміст BL з нулем; результат команди впливає на
                            ; встановлення прапорця нуля Z
JZ          M1              ; Якщо Z=1 (BL=0), то перехід на мітку M1,
DIV         BL              ; інакше виконати ділення AL←AX / BL, остача в AH
JMP         M2              ; Безумовний перехід на мітку M2
M1: MOV     AL, 0FFH        ; Занести число 0FFH у AL
M2: MOV     [1000H], AL     ; Запам'ятати результат у комірці DS:[1000H]
END                         ; Завершення програми
```

Процедура ЯКЩО-ТО (рис. 8.3) є частковим випадком процедури ЯКЩО-ТО-ІНАКШЕ і використовується в тому випадку, коли треба реалізувати одну обчислювальну процедуру залежно від умови.

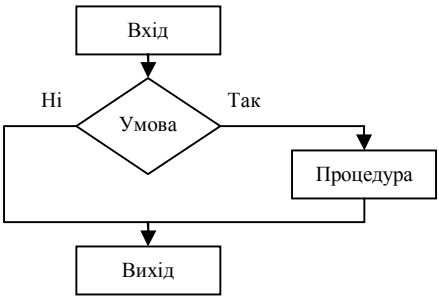


Рис. 8.3. Алгоритм процедури ЯКЩО-ТО

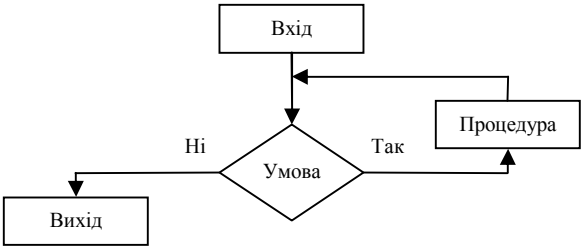


Рис. 8.4. Алгоритм процедури РОБИ-ПОКИ

Процедура РОБИ-ПОКИ (рис 8.4) використовується для повторення однотипних дій до моменту виконання умови закінчення циклу.

Приклад 8.4. Написати програму додавання за модулем 256 масиву з 100H байтів, розташованих за початковою адресою 7000H:3000H. Результат у вигляді одного байта записати в комірку з адресою 7000H:5000H.

Програма має вигляд:

```
Мнемокод    Операнд(и)    Коментар
ORG         100H          ; Початок програми
MOV         AX, 7000H     ; Завантажити а AX адресу сегмента
MOV         DS, AX        ; Завантажити в DS адресу сегмента
MOV         SI, 3000H     ; Завантажити в SI зміщення першого елемента масиву
MOV         CX, 101H      ; Завантажити у лічильник CX довжину масиву + 1
MOV         AL, [SI]      ; Завантажити у AL перший елемент масиву
M1:LOOP     M0            ; Зменшити вміст CX на 1, якщо CX≠0, то перейти на
                          ; мітку M0,
MOV         [5000H],AL    ; інакше запам'ятати результат у DS:[5000H]
JMP         HLT           ; Перехід на вихід
M0: INC     SI            ; SI ← SI + 1 - адреса наступного елемента
ADD         AL,[SI]       ;Додати вміст SI до попередньої суми в акумуляторі AL
JMP         M1            ; Перейти на мітку M1 для перевірки умови виходу із циклу
HLT:NOP
END                       ; Завершення програми
```

Процедура ПОВТОРЮЙ-ДО-ТОГО-ЯК (рис. 8.5) аналогічна попередній, але однотипні дії виконуються перед перевіркою умови.

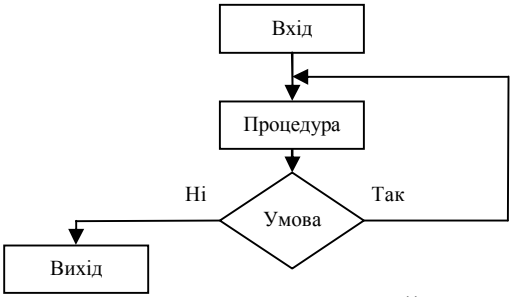


Рис. 8.5. Алгоритм процедури ПОВТОРЮЙ-ДО-ТОГО-ЯК

Програма виконання завдання прикладу 8.4 згідно з алгоритмом (рис. 8.5) має вигляд:

Мнемокод	*Операнд(и)*	*Коментар*
ORG	100H	; Початок програми
MOV	AX, 7000H	; Завантажити а AX адресу сегмента
MOV	DS, AX	; Завантажити в DS адресу сегмента
MOV	SI, 3000H	; Завантажити в SI зміщення першого елемента масиву
MOV	CX, 100H	; Завантажити у лічильник CX довжину масиву
MOV	AL, [SI]	; Завантажити у AL перший елемент масиву
M0: INC	SI	; SI ← SI + 1 - адреса наступного елемента
ADD	AL,[SI]	; Додати вміст SI до попередньої суми в акумуляторі AL
LOOP	M0	; Зменшити CX на 1, якщо CX≠0, то перейти на мітку M0,
MOV	[5000H],AL	; інакше запам'ятати результат у DS:[5000H]
END		; Завершення програми

Практична частина

Складіть програму, яка б сортувала елементи масиву згідно з ознакою парності, причому по закінченні роботи програми, на початку масиву повинні бути розташовані непарні числа, а у кінці – парні (завдання з лабораторної роботи № 6).

У програмі-емуляторі 16-розрядної системи на основі мікропроцесора 8086 реалізуйте цю програму.

Програма буде мати такий вигляд:

```
#make_COM#
ORG  100H
  JMP begin
ARRAY DB 3,6,9,2,8,4,5,7,1,3
begin:     MOV CX, 9
  MOV BX, 0
next:      MOV AL, ARRAY[BX]
```

```
    TEST AL, 1
    JNZ n_xch
    PUSH AX
    MOV AL, ARRAY[BX+1]
    TEST AL, 1
    JZ n_xch
    MOV ARRAY[BX], AL
    POP AX
    MOV ARRAY[BX+1], AL
    INC CX
    DEC BX
    JMP next
n_xch:  INC BX
    LOOP next
    END
```

Дана програма сортує елементи масиву ARRAY, який являє собою сукупність з десяти чисел, а саме: 3, 6, 9, 2, 8, 4, 5, 7, 1, 3.

Переконайтеся у правильності виконання складеної програми. Після зупинки роботи програми елементи масиву ARRAY будуть розташовані в такому порядку: 3, 9, 5, 7, 1, 3, 6, 2, 8, 4, тобто перші шість елементів – непарні числа, а останні чотири – парні.

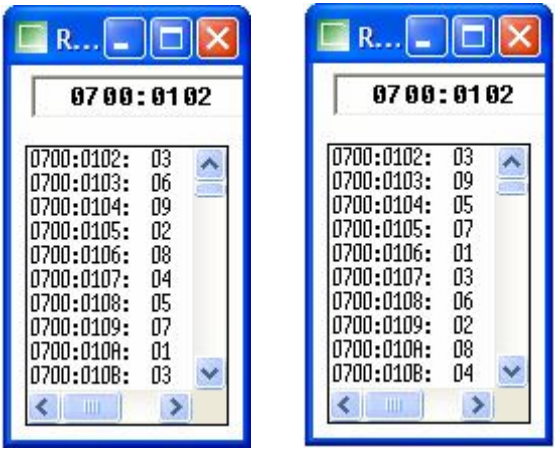


Рис. 8.6. Розташування елементів масиву в пам’яті до (зліва) і після (справа) роботи програми

Простежте за зміною вмісту регістрів мікропроцесора, які задіяні до процесу виконання програми, та зміною стану регістра прапорців умов. До процесу виконання програми задіяні три регістри загального призначення, а саме: AX, CX та BX, вказівник команд IP, вказівник стека SP та прапорець нуля ZF. У молодшій частині регістра AX (AL) тимчасово зберігаються елементи масиву ARRAY, тобто значення, які набуває AL, лежать у діапазоні від 1 до 9. Регістр CX використовується як лічильник циклів оброки масиву і використовується командою LOOP next, під час виконання якої автоматично декрементується. Регістр BX використовується для індексації елементів масиву ARRAY. Значення, які набувають CX та BX, лежать у діапазоні від 0 до 9. Вказівник команд IP містить адреси команд, які виконуються (записуються у регістр команд). Оскільки програма розташована в адресному просторі від 0700H:0100H до 0700H:0132H (0700H – адреса програмного сегмента, тобто вміст регістра CS, який є незмінним), то відповідно значення, які набуває IP, лежить у діапазоні від 0100H до 0131H. Стек використовується під час виконання програми як проміжне зберігання елементів масиву для обміну. Вершина стека має адресу 0FFFEH (початкове значення SP), а при занесенні у стек проміжного вмісту регістра AX регістр SP автоматично декрементується і набуває значення 0FFFCH. Оскільки код програми, дані і стек розташовані в одному сегменті, то вміст відповідних регістрів CS, DS, ES та SS однаковий і становить 0700H. Прапорець нуля ZF використовується під час виконання команд умовних переходів JNZ n_xch, JZ n_xch, LOOP next та встановлюється або скидається під час виконання команд TEST AL, 1 і LOOP next.

Проаналізуйте особливості розташування програмного коду в пам’яті мікропроцесорної системи. Програма розташована в адресному просторі від 0700H:0100H до 0700H:0132H, причому область даних, яка відповідає масиву ARRAY, розташована в адресному просторі від 0700H:0102H до 0700H:010BH. Визначте кількість байтів, які займають команди програми (які саме команди є одно-, дво-, три- та чотирибайтними командами).

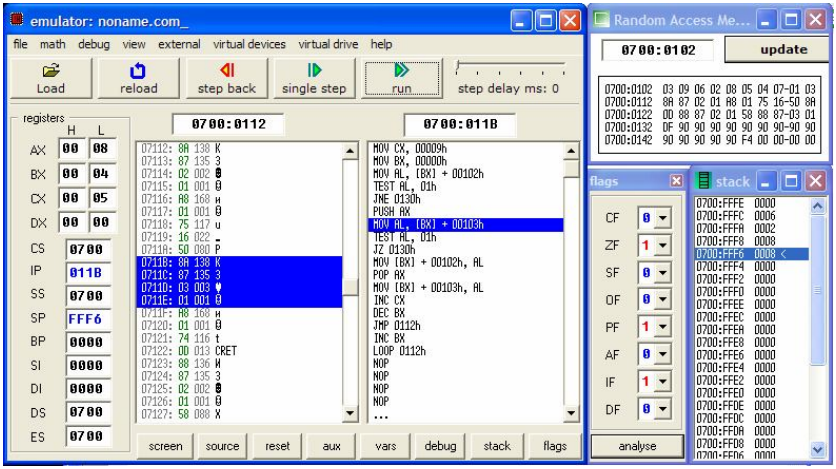


Рис. 8.7. Фрагмент стану програми-емулятора мікропроцесора 8086 під час виконання програми

Контрольні запитання та завдання

1. Яке основне призначення програми-транслятора?
2. Опишіть особливості організації алгоритмів типових обчислювальних процедур у мові програмування асемблер. Які прапорці умов зазвичай використовуються?
3. Що необхідно зробити, щоб прикладна програма почала виконуватися мікропроцесором, і що відбувається по завершенні виконання програми?
4. Реалізуйте свій варіант деякої алгоритмічної задачі, наприклад, обчислення суми елементів масиву, пошук максимального та мінімального елементів масиву тощо.
5. Реалізуйте складені вами програми у системі схемотехнічного моделювання Proteus, використовуючи відповідну модель мікропроцесора 8086.
6. Проаналізуйте виконання складених вами програм у покроковому режимі.
7. Що таке стекова область пам’яті? Опишіть особливості процесу обміну інформацією між стековою областю пам’яті та мікропроцесором. Модифікуйте програму практичної частини для унеможливлювання переповнення стека.

Лабораторна робота № 9
Дослідження форматів команд та особливостей їх кодування для мікропроцесора 8086

Теоретична частина

Формати команд

Команди мікропроцесора 8086 мають змінну довжину від 1 до 6 байтів. За кількістю операндів, які обробляються, команди поділяються на безадресні, одноадресні і двохадресні. У двохадресних командах результат завжди записується за першою адресою і лише один з операндів може знаходитися в пам'яті. При цьому в мнемоніці *операнд-приймач записується зліва від коми-роздільника операндів, а операнд-джерело – справа*. Всього існує чотири джерела операндів: тіло команди, регістр, пам'ять і порт введення/виведення. У першому випадку операнд називається безпосереднім.

Перший байт команди містить код операції (*КОП*), до складу якого можуть входити спеціальні розряди d, s і w. При w = 1 операції виконуються з 16-розрядними словами, при w = 0 – з байтами. Розряд d визначає напрям передачі даних у двохоперандних командах: із регістра в регістр/пам'ять (d = 0) або з регістра/пам'яті в регістр (d = 1). S – визначає розширення 8-бітових безпосередніх даних до повного розміру (s = 1) чи ні (s = 0). При деяких поєднаннях команд і методів адресації (регістровий метод адресації) положення операнда може задаватися безпосередньо у байті коду операції, але частіше для цього використовується так званий постбайт.

У командах, що мають довжину 2 і більш байти, другий байт називається *постбайтом*. Він виконує функції кодування адрес операндів. Байти 3 – 6 присутні в команді залежно від типу адреси операнда, описаного постбайтом і наявність безпосереднього операнда.

Постбайт складається з трьох полів: *режиму – MOD*, *регістра/частини коду операції – REG/OC* і *регістра/пам'яті – R/M*. Поле MOD займає 2 біти (6 і 7) постбайта. Поле REG займає 3 біти (3-5) постбайта. Поле R/M займає 3 біти (0-2) постбайта.

Полями MOD і R/M спільно кодується тип адреси операнда, що знаходиться в пам'яті або регістрі. 32 значення цих полів визначають знаходження операнда 24 можливими методами адресації або в одному з 8 регістрів.

Нижче наведена структура байта коду операції і постбайта, а в таблицях показано формування адрес регістрових операндів і адрес операндів пам'яті. DISP8 і DISP16 – зміщення завдовжки 8 і 16 бітів – розташовані в команді безпосередньо за байтом адресації.

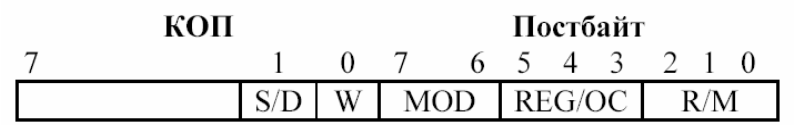


Рис. 9.1. Формат байта КОП та постбайта

Таблиця 9.1

Формат поля REG

Поле REG	Регістри		Сегментні регістри
	W=1	W=0	
000	AX	AL	ES
001	CX	CL	CS
010	DX	DL	SS
011	BX	BL	DS
100	SP	AH	-
101	BP	CH	-
110	SI	DH	-
111	DI	BH	-

У таблиці 9.2, окрім визначення режиму адресації оперативної пам'яті, вказаний також сегментний регістр, що використовується за замовчуванням для отримання фізичної адреси. Використання іншого сегментного регістра можливе введенням спеціального префікса (додаткового байта, який записується перед командою).

Таблиця 9.2

Кодування типу адреси операнда

R/M	MOD				
	00	01	10	11	
				w=1	w=0
000	(BX)+(SI) (DS)	(BX)+(SI)+disp 8 (DS)	(BX)+(SI)+disp 16 (DS)	AX	AL
001	(BX)+(DI) (DS)	(BX)+(DI)+disp 8 (DS)	(BX)+(DI)+disp 16 (DS)	CX	CL
010	(BP)+(SI) (SS)	(BP)+(SI)+disp 8 (SS)	(BP)+(SI)+disp 16 (SS)	DX	DL
011	(BP)+(DI) (SS)	(BP)+(DI)+disp 8 (SS)	(BP)+(DI)+disp 16 (SS)	BX	BL
100	(SI) (DS)	(SI)+disp 8 (DS)	(SI)+disp 16 (DS)	SP	AH
101	(DI) (DS)	(DI)+disp 8 (DS)	(DI)+disp 16 (DS)	BP	CH
110	disp16 (DS)	(BP)+disp 8 (SS)	(BP)+disp 16 (SS)	SI	DH
111	(BX) (DS)	(BX)+disp 8 (DS)	(BX)+disp 16 (DS)	DI	BH

В командах, які використовують безпосередній операнд, ознака s разом з ознакою w визначає розрядність безпосереднього операнда, що записується в команді, і розрядність виконуваної операції згідно з таблицею 9.3.

Таблиця 9.3

w	s	Операція	Безпосередній операнд
0	0	8-розрядна	8-розрядний
0	1	не використовується	
1	0	16-розрядна	16-розрядний
1	1		8-розрядний, розширюваний знаком до 16-ти розрядів при виконанні операції

Система команд та її кодування

Конкретний формат команди визначається відповідним префіксом (першим байтом) та наведений у загальній таблиці машинного подання команд, яку можна знайти в довідниковій літературі, присвяченій мікропроцесору 8086.

Відновлення символічного запису команди за її машинним поданням

Для фахівця, що працює з комп'ютером як на програмному, так і на апаратному рівні, іноді виникає необхідність ідентифікувати командну інформацію, що зберігається в оперативній пам'яті. Це може бути потрібно, наприклад, у випадку програмно-апаратного збою, причину і місце якого важко визначити традиційними методами і засобами тестування і налагодження програм. Оскільки виконуваний модуль програми зберігається в пам'яті в машинному поданні, то для кращого розуміння дій, що виконуються комп'ютером у певний момент, доцільно перетворити команду до символічного вигляду. Програми, що виконують таке перетворення, називаються дизасемблерами.

Розглянемо декілька прикладів подібних перетворень. Для правильної інтерпретації команди необхідно знати положення її першого байта. У даних прикладах вважатимемо, що воно відоме.

Приклад 9.1. Представити символічний запис команди, що має машинну форму 0000h.

Оскільки поля команди визначаються з точністю до біта, то необхідно спочатку перейти від шістнадцятирічного до двійкового зображення команди і, виходячи із загальних принципів кодування команд, визначити призначення всіх її розрядів:

$$\begin{array}{l} \qquad\qquad\quad \text{КОП}\ \ dw\ \ md\ \ reg\ \ r/m \\ 0000\,h = 000000\ 00\ \ 00\ 000\ 000\,b \end{array}$$

За таблицею машинного подання команд (див. довідник) визначимо, що КОП=000000 b відповідає загальному формату операції додавання ADD. Тоді два молодших біти першого байта кодують ознаки d і w, а другий байт є постбайтом, що визначає режими адресації операндів, що беруть участь в операції. Значення полів у постбайті дозволяє визначити, що операндами будуть регістр AL (reg = 000, w = 0) (див. табл. 9.1) і байт пам'яті, що адресується за допомогою базово-індексної адресації через

регістри BX і SI (md = 00, r/m = 000) (див. табл. 9.2). Значення d = 0 указує, що регістр AL є операндом-джерелом.

Отже, символічний запис команди має вигляд

ADD [BX+SI], AL

Приклад 9.2. Представити символічний запис команди, що має машинну форму 81475D398B h.

Переходимо до двійкового подання команди:

код	sw	md	кол	r/m	disp L	data L	data H
100000	01	01	000	111	01011101	00111001	10001011

Перший байт, згідно з таблицею машинного подання команд (див. довідник), відповідає команді додавання з безпосереднім операндом. Постбайт у цьому випадку кодує місцеположення лише одного операнда, яке визначається полями md і r/m: (BX)+disp8 (див. табл. 9.2), а середнє поле постбайта є розширенням коду операції.

Адресація операнда вимагає вказівки в команді 8-розрядного зсуву. Воно розміщується відразу ж за постбайтом. Останні байти команди кодують безпосередній операнд. Значення sw = 01 у першому байті команди указує на те, що безпосередній операнд – 16-розрядний. Враховуючи, що при кодуванні в команді двобайтових величин спочатку записується їх молодший байт, одержимо такий символічний запис команди:

ADD [BX+5D],8B39h

Приклад 9.3. Нехай машинна форма подання команди 0445h. Тоді її двійковий вигляд:

код	w	data L
0000010	0	0100 0101 b

За таблицею машинного подання команд (див. довідник) визначаємо, що це команда спеціального формату, що забезпечує додавання акумулятора з безпосереднім операндом.

Оскільки w = 0, то безпосередній операнд має довжину 1 байт, а як акумулятор використовується регістр AL. При цьому команда має вигляд

ADD AL,45h

Практична частина

На основі програм, складених під час виконання попередньої лабораторної роботи (персональні завдання), дослідіть особливості кодування команд мікропроцесора 8086, які використовуються у ваших програмах. Використовуючи програму-емулятор 16-розрядної системи на основі мікропроцесора 8086, складіть відповідне подання програм у машинних кодах. Дослідіть формат кожної з команд, використаних у програмах, шляхом розшифровки відповідних полів двійкового подання команд. Поясніть значення безпосередніх операндів та адрес зміщення.

Для прикладу розглянемо завдання з попередньої лабораторної роботи. Програма на мові асемблера та її машинне (двійкове) зображення має такий вигляд:

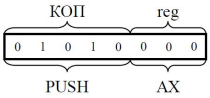
Програма на мові асемблера	Програма у машинних (двійкових) кодах				
	Адреса	1-й байт	2-й байт	3-й байт	4-й байт
#make_COM#					
ORG 100H					
JMP begin	0100:	11101011	00001010		
ARRAY DB 3,6,9,2,8,4,5,7,1,3					
begin: **MOV CX,9**	010C:	10111001	00001001	00000000	
MOV BX,0	010F:	10111011	00000000	00000000	
next: **MOV AL,ARRAY[BX]**	0112:	10001010	10000111	00000010	00000001
TEST AL,1	0116:	10101000	00000001		
JNZ n_xch	0118:	01110101	00010110		
PUSH AX	011A:	01010000			
MOV AL,ARRAY[BX+1]	011B:	10001010	10000111	00000011	00000001
TEST AL,1	011F:	10101000	00000001		
JZ n_xch	0121:	01110100	00001101		
MOV ARRAY[BX],AL	0123:	10001000	10000111	00000010	00000001
POP AX	0127:	01011000			
MOV ARRAY[BX+1],AL	0128:	10001000	10000111	00000011	00000001
INC CX	012C:	01000001			
DEC BX	012D:	01001011			
JMP next	012E:	11101011	11100010		
n_xch:**INC BX**	0130:	01000011			
LOOP next	0131:	11100010	11011111		
END					

Дослідимо формат кожної команди, використаної у програмі. У програмі використано 1-, 2-, 3- та 4-байтні команди. Розшифруємо поля двійкового подання команд. Для цього приймемо такі позначення: **КОП** – поле Коду ОПерації; **reg** – поле, яке вказує регістр регістрової пам'яті мікропроцесора; **w** –

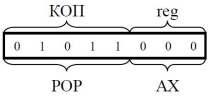
ознака, яка вказує на довжину операндів; **d** – ознака, яка вказує положення приймача результату; **md,r/m** – поля, які задають режим адресації другого операнда; **disp L** – молодший байт адреси зміщення; **disp H** – старший байт адреси зміщення; **data L** – молодший байт безпосереднього операнда; **data H** – старший байт безпосереднього операнда.

Однобайтні команди

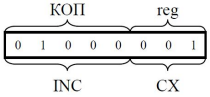
PUSH AX (Занесення в стек вмісту регістра AX).



POP AX (Запис у регістр AX вмісту вершини стека).

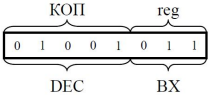


INC CX (Збільшення на одиницю вмісту регістра CX).



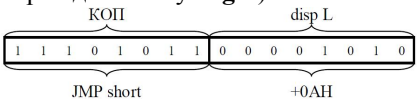
Примітка: у випадку **INC BX** у полі reg буде міститися 011.

DEC BX (Зменшення на одиницю вмісту регістра BX).



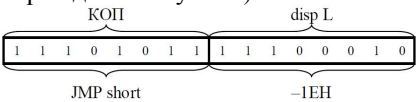
Двобайтні команди

JMP begin (Безумовний короткий перехід на мітку **begin**).



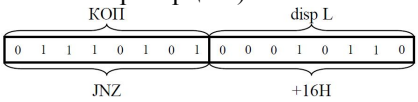
Пояснення вмісту поля disp L: Оскільки під час виконання команди переходу, до вмісту регістра IP додається адреса зміщення, а перейти треба з адреси 0102 на адресу 010C, то dispL = 010C – 0102 = +0A.

JMP next (Безумовний короткий перехід на мітку **next**).



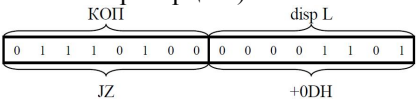
Пояснення вмісту поля disp L: Оскільки під час виконання команди перейти треба з адреси 0130 на адресу 0112, то disp L = 0112 – 0130 = –1E (в доповняльному коді).

JNZ n_xch (Короткий перехід на мітку **n_xch** при нульовому значенні прапорця **Z**).



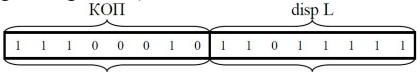
Пояснення вмісту поля disp L: Оскільки під час виконання команди перейти треба з адреси 011A на адресу 0131, то disp L = 0130 – 011A = +16.

JZ n_xch (Короткий перехід на мітку **n_xch** при одиничному значенні прапорця **Z**).



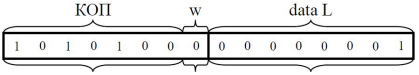
Пояснення вмісту поля disp L: Оскільки під час виконання команди перейти треба з адреси 0123 на адресу 0130, то disp L = 0130 – 0123 = +0D.

LOOP next (Короткий перехід на мітку **next** при нульовому значенні вмісту регістра CX, тобто при одиничному значені прапорця **Z**, а також автоматичне зменшення на одиницю (декремент) вмісту регістра CX).



Пояснення вмісту поля disp L: Оскільки під час виконання команди перейти треба з адреси 0134 на адресу 0112, то disp L = 0112 – 0133 = –21 (в доповняльному коді).

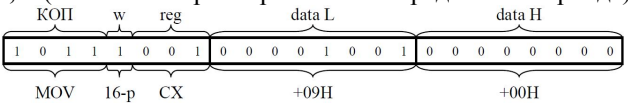
TEST AL, 1 (Логічне **I** вмісту регістра AL та безпосереднього операнда).



Пояснення вмісту поля data L: Оскільки під час виконання команди необхідно перевірити на парність вміст регістра AL, тобто визначити значення молодшого біта, то виконується операція логічного **I** із безпосереднім операндом $00000001_{(2)}$.

Трибайтні команди

MOV CX, 9 (Занесення в регістр CX безпосереднього операнда).

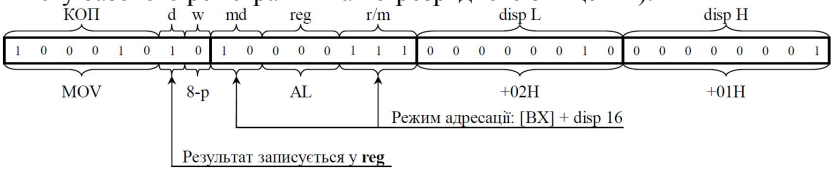


Пояснення вмісту полів data L,H: Оскільки під час виконання команди необхідно в 16-розрядний регістр CX занести число 0009H, то data H = 00H, data L = 09H.

Примітка: У випадку команди **MOV BX,0** необхідно в 16-розрядний регістр BX занести число 0000H, тоді data H = 00H, data L = 00H, а поле reg буде містити значення $011_{(2)}$.

Чотирибайтні команди

MOV AL, ARRAY[BX] (Занесення в регістр AL операнда у режимі регістрової відносної адресації, ефективна адреса якого дорівнює сумі вмісту базового регістра BX та 16-розрядного зміщення).



Пояснення вмісту полів disp L,H: Під час виконання команди необхідно у 8-розрядний регістр AL занести операнд, ефективна адреса якого містить 16-розрядну адресу зміщення, що являє собою адресу початку масиву ARRAY, тобто 0102H. Таким чином, disp H = 01H, disp L = 02H.

Примітка: У випадку команди **MOV ARRAY[BX], AL** (Занесення вмісту регістра AL у комірку пам'яті, ефективна адреса якої дорівнює сумі вмісту базового регістра BX та 16-розрядного зміщення) ознака **d=0 –** результат записується у пам'ять у режимі адресації, який визначається полями **md,r/m**.

Контрольні запитання та завдання

1. На основі програм, складених під час виконання попередньої лабораторної роботи (персональні завдання), дослідіть особливості кодування команд мікропроцесора 8086, які використовуються у ваших програмах. Дослідження виконайте аналогічно прикладу, наведеному у практичній частині даної лабораторної роботи. Результати досліджень оформіть у вигляді звіту.
2. Що таке формат команди мікропроцесора? Які поля команд мікропроцесора 8086 ви знаєте?
3. Якими елементами команди мікропроцесора 8086 визначається розрядність регістрів та операндів, задіяних при виконанні команди?
4. Якими елементами команди мікропроцесора визначається режим адресації операндів?
5. Оптимізуйте код програми, наведений у практичній частині даної лабораторної роботи, шляхом використання команд меншої довжини. Наведену програму можна зменшити мінімум на п’ять байтів без зміни її структури та функціональних можливостей.
6. Що таке дизасемблювання? При якій умові можливе відновлення символічного запису команди за її машинним зображенням?
7. Як забезпечується автоматичне „розпізнавання” мікропроцесором команд різної довжини та автоматичне „розпізнавання” команд та даних? Як себе поведе процесор у випадку спроби виконання команд, розмір яких у пам’яті штучно змінено (наприклад, для 3-байтної команди на місці третього байта записано перший байт іншої команди або елемент послідовного масиву даних)?

Лабораторна робота № 10
Дослідження системи переривань мікропроцесора 8086

Теоретична частина

Переривання є засобом запиту з боку периферійного пристрою на виконання деякого виду обробки даних у процесорі. Процесор при виконанні кожної команди перевіряє наявність запиту переривання та при його виявленні переходить до його обробки.

Дії, що здійснюються процесором при обробці переривання, залежать від типу процесора, але, як правило, апаратними засобами виконується наступне:

1) скидається тригер дозволу переривання;
2) вміст регістра прапорців умов заноситься в стек;
3) вміст програмного лічильника заноситься в стек;
4) в програмний лічильник заноситься деяка адреса з таблиці векторів переривань.

Таким чином, наступна команда виконується починаючи з нової адреси, що занесена в програмний лічильник. За цією адресою міститься підпрограма, яка використовується при обробці переривань.

Переривання існує двох видів: *апаратне* та *програмне*.

Апаратне переривання

На рис. 10.1 зображені процесор, запам'ятовуючий пристрій з таблицею адрес, контролер переривань, зовнішній пристрій, який вимагає переривання, та восьмирозрядна шина адреси, по якій із контролера переривань процесору передається вказівка про те, яка адреса з таблиці векторів переривань повинна бути використана при отриманні процесором сигналу апаратного переривання. Зв'язок системи переривань із процесором здійснюється за допомогою лінії переривань INTR; поява на цій шині активного сигналу призводить до переривання роботи процесора. Справа від контролера переривань (рис. 10.1) зображені лінії IRQ0 – IRQ7, які з'єднують її з пристроями, які можуть вимагати переривання роботи процесора, і, зокрема, пристрій, який вимагає уваги з боку процесора та використовує для переривання лінію IRQ1.

Переривання виникає тоді, коли пристрій, який вимагає переривання (наприклад, друкуючий пристрій), посилає сигнал на лінію IRQ2. Контролер переривань виявляє наявність сигналу на лінії і реагує на нього; тобто розміщує восьмирозрядне число на молодшу частину шини адреси та встановлює активний сигнал на лінії переривань INTR. Процесор, визначивши, що на лінії INTR з'явився активний сигнал, запам'ятовує у стековій області пам'яті свій стан (регістр прапорців умов FLAGS), а також адресу команди, яка повинна була б виконуватися наступною (вміст регістрів CS:IP). Потім з адресної шини зчитується число, що вказує, яку адресу необхідно витягти з таблиці векторів переривань, і процесор переходить до виконання команди, яка починається з цієї адреси.

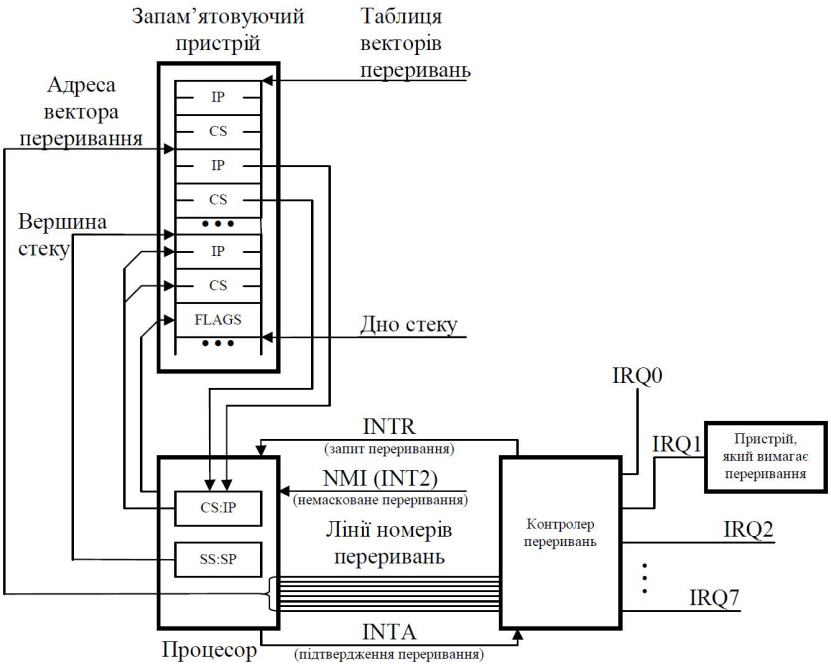


Рис. 10.1. Обробка апаратного переривання

Типовим прикладом пристрою, який вимагає переривання, є клавіатура. Після натискання клавіші на клавіатурі в контролер

переривань передається сигнал, який призводить до переривання роботи процесора. Очевидно, що процесор припиняє свою роботу і, використовуючи адресу, передану із контролера переривань, починає виконання спеціальної програми взаємодії з клавіатурою (ця програма є частиною Базової Системи Введення-Виведення (BIOS), яка записана в постійному запам'ятовуючому пристрої).

Програма взаємодії з клавіатурою вводить із клавіатури код, який відповідає натиснутій клавіші, і заносить його в область зберігання, розташовану в пам'яті. Потім програма взаємодії з клавіатурою повертає керування програмі, яка виконувалась до переривання, в точку зупинки її роботи.

Програмне переривання

Переривання можуть також мати місце при виконанні спеціальної команди: INT N (перервати, використовуючи номер N). Число N вказує одну з адрес в таблиці векторів переривань. При виконанні команди „перервати" процесор визначає і запам'ятовує адресу команди, яка повинна була б виконуватись наступною, а потім переходить до виконання програми, яка починається з адреси, отриманої з таблиці векторів переривань. Така послідовність дій називається *програмним перериванням*.

Щоб проілюструвати застосування програмних переривань, повернемося до прикладу з клавіатурою. Нагадаємо, що в цьому прикладі зберігався код, який відповідає натиснутій клавіші. Для отримання цього коду виконувана програма повинна видати запит на введення з клавіатури. Такий запит програми викликає програмне переривання. Процедура, яка запускається при цьому, полягає в отриманні коду з області зберігання і передачі його виконуваній програмі. Якщо виконувана процедура не може знайти код в області зберігання, то клавіша не натискалася. Процедура зациклюється в очікуванні, поки не буде натиснуто яку-небудь клавішу і код, який відповідає цій клавіші, не буде розміщений в область зберігання, що дозволить процедурі отримати його і передати керування програмі, яка викликала переривання (Процедура пошуку коду в пам'яті входить до складу Базової Системи Введення-Виведення (BIOS), розташованої в постійному запам'ятовуючому пристрої).

Практична частина

Дослідимо механізм апаратних переривань на прикладі мікропроцесора 8086. Для цього реалізуємо програму сортування елементів масиву згідно з ознакою парності (завдання лабораторної роботи № 8) у вигляді підпрограми обробки апаратного немаскованого переривання (INT 2). Нагадаємо, що апаратні переривання можуть бути немаскованими (вхід NMI мікропроцесора) та маскованими (вхід INTR мікропроцесора). Обробку маскованих апаратних переривань можна заборонити шляхом скидання прапорця IF регістра прапорців умов FLAGS.

Програму дослідження переривань організуємо у вигляді EXE-файла, а не COM-файла, як ми це робили раніше. Особливість будови EXE-програм полягає у відповідності сегментів даних, стека та коду різним адресним просторам, на відміну від COM-програм, в яких дані, стек і програмний код містяться в одному сегменті. Написання EXE-програм має виковуватися за умови зазначення відповідності між сегментами та сегментними регістрами. Програма дослідження апаратних переривань мікропроцесора 8086, в нашому випадку, має вигляд:

```
data segment ; Сегмент даних
  ARRAY DB 3,6,9,2,8,4,5,7,1,3
ends          ; Кінець сегмента даних
stack segment ; Сегмент стека
  DW  128  DUP(0) ; Ініціалізація у стеці 128 слів (word) значенням 0
ends          ; Кінець сегмента стека
code segment ; Сегмент коду
start: MOV AX, 0
  MOV ES, AX
  MOV AL, 2 ; Обчислення адреси вектора переривання з номером 2
  MOV BL, 4 ; Множення 2*4 та занесення результату в BX
  MUL BL
  MOV BX, AX
  MOV SI, offset [int_prog]
  MOV ES:[BX], SI ; Занесення адреси зміщення в таблицю векторів
  ADD BX, 2
  MOV AX, CS
  MOV ES:[BX], AX ; Занесення сегмента коду в таблицю векторів
  MOV AX, data
```

```
  MOV DS, AX ; Запис у регістр сегмента даних адреси масиву ARRAY
  MOV ES, AX ; Запис у регістр розширеного сегмента даних
sleep: NOP ; Порожній цикл очікування
  JMP sleep
int_prog:   ; Підпрограма обробки переривання (сортування елементів)
  MOV CX, 9
  MOV BX, 0
next:   MOV AL, ARRAY[BX]
  TEST AL, 1
  JNZ n_xch2
  PUSH AX
  MOV AL, ARRAY[BX+1]
  TEST AL, 1
  JZ n_xch1
  MOV ARRAY[BX], AL
  POP AX
  MOV ARRAY[BX+1], AL
  INC CX
  DEC BX
  JMP next
n_xch1: POP AX
n_xch2: INC BX
  LOOP next
  IRET      ; Повернення з підпрограми обробки переривання
ends        ; Кінець сегмента коду
end start
```

Розглянемо структуру та особливості функціонування даної програми. Директивою data segment оголошується сегмент даних, де буде міститися масив чисел ARRAY, який ми будемо сортувати. Директивою stack segment оголошується сегмент стека, в який заноситься 128 нульових елементів типу word (або 256 нульових елементів типу byte). Далі оголошується програмний сегмент директивою code segment, який буде містити фрагмент програми, результат роботи якої полягає у занесенні адреси підпрограми обробки переривання у таблицю векторів переривань, а також власне підпрограма обробки переривання int_prog, результат роботи якої полягає у сортуванні елементів масиву згідно з ознакою парності (завдання лабораторної роботи

№ 8). Тут слід звернути увагу, що програма сортування дещо модифікована. Суть модифікації полягає у введенні додаткової команди виштовхування зі стека проміжного елемента POP AX для забезпечення однакової кількості операцій занесення у стек PUSH AX та виштовхування POP AX (елементи, які підлягають обміну місцями, під час сортування тимчасово заносяться в стек, відповідні комірки якого відіграють роль проміжних комірок пам'яті). Якщо залишити програму сортування елементів у вигляді, наведеному в лабораторній роботі № 8, то по завершенні її роботи у стеці будуть знаходитись парні елементи, які мінялися місцями з непарними при сортуванні (див. рис. 8.7). При цьому під час виконання мікропроцесором команди IRET (повернення з підпрограми обробки переривання) відбудеться некоректне відновлення зі стеку значень регістра вказівника команд, регістра сегмента коду та регістра прапорців умов, тобто замість адрес та стану процесора у відповідні регістри будуть занесені деякі дані, що призведе до збою роботи мікропроцесорної системи.

У емуляторі мікропроцесорної системи Emu8086 скомпілюйте програму, наведену вище, створивши EXE-файл програми. Отриманий файл прикріпіть до моделі мікропроцесора 8086 у системі схемотехнічного моделювання Proteus. До входу немаскованого переривання NMI мікропроцесора приєднайте логічну константу для сигналізації зовнішнього апаратного переривання INT 2 (рис. 10.2). Запустіть процес моделювання та проаналізуйте вміст адресного простору, що відповідає сегменту даних (початковий масив чисел ARRAY). В момент перебування процесора у порожньому циклі очікування sleep: NOP; JMP sleep (рис. 10.3) змініть значення логічної константи з нуля на одиницю, тобто згенеруйте переривання INT 2.

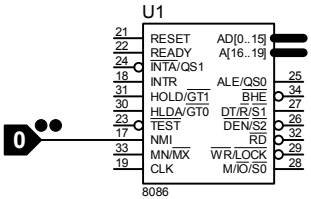


Рис. 10.2. Модель мікропроцесора 8086 у системі Proteus

Рис. 10.3. Програмна модель мікропроцесора 8086 у системі Proteus

Одразу після появи сигналу на лінії NMI мікропроцесор почне виконувати підпрограму обробки переривання, тобто відсортує елементи масиву згідно з ознакою парності (рис. 10.4).

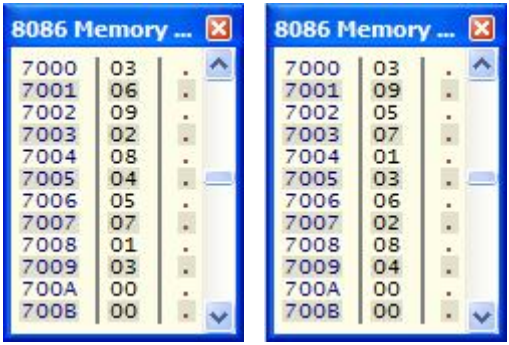


Рис. 10.4. Розташування елементів масиву в пам’яті до (зліва) і після (справа) роботи підпрограми обробки переривання

Проаналізуємо особливості роботи мікропроцесора для нашого випадку. До порожнього циклу очікування, мікропроцесор виконує фрагмент програми, результат роботи якої полягає у занесенні адреси підпрограми обробки переривання у таблицю векторів переривань (рис. 10.5, зліва). Адреса підпрограми int_prog (CS:IP = 0711:0025) заноситься у комірки пам’яті, починаючи з восьмої (4×2=8), оскільки немасковане апаратне переривання NMI є перериванням із номером 2, а вектори переривань займають 4 комірки пам’яті. Після появи сигналу

переривання мікропроцесор заносить у стек вміст регістра прапорців та адресу команди, яка повинна була виконатись наступною (в нашому випадку це команда NOP з адресою CS:IP = 0711:0022), як це показано на рис. 10.5 справа. По закінченні роботи підпрограми обробки переривання при виконанні команди IRET значення 0711:0022 буде відновлено.

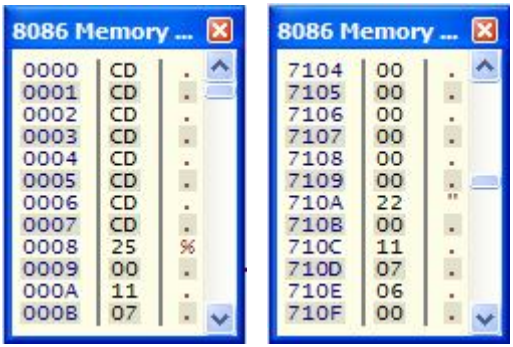


Рис. 10.5. Початковий фрагмент таблиці векторів переривань (зліва). Початковий фрагмент сегмента стека (справа).

Контрольні запитання та завдання

1. Дайте визначення поняттю „переривання”. Які види переривань ви знаєте?
2. Опишіть процес обробки переривань мікропроцесором 8086.
3. Яке призначення контролера переривань?
4. Що таке таблиця векторів переривань? Де вона знаходиться?
5. Наведіть відмінності в організації та функціонуванні EXE- та COM-файлів.
6. На основі програм, складених під час виконання лабораторної роботи № 8 (індивідуальні завдання), дослідіть особливості функціонування механізму переривань мікропроцесора 8086, організувавши складені програми у вигляді підпрограм обробки немаскованого апаратного переривання.
7. Реалізуйте ваші завдання у вигляді підпрограм обробки програмних переривань (наприклад, INT 90H) та у вигляді підпрограми обробки особливої ситуації ділення на 0 (INT 0).

Лабораторна робота № 11
Дослідження роботи мікропроцесора 8086 при взаємодії з портами введення-виведення

Теоретична частина

За допомогою портів введення-виведення організовуються операції введення та виведення даних із мікропроцесорної системи. Порти приймають дані від периферійних пристроїв та направляють їх у ці пристрої. До периферійних пристроїв відносять, наприклад, клавіатуру, мишу, дисплей тощо. Сукупність сигналів, які надійшли на адресну шину порту, що з'єднує систему портів введення-виведення і процесор, задає номер порту, з яким встановлюється зв'язок. Оскільки адресна шина порту є шістнадцятирозрядною, значення адрес лежать у діапазоні від 0000H до FFFFH, тобто процесор може адресувати 2^{16} = 65536 портів. Сигнал на шині введення-виведення / зчитування-запису вказує напрямок передачі потоку даних через систему портів введення-виведення. Якщо цей сигнал дорівнює 1, дані надходять із процесора; якщо ж сигнал дорівнює 0, дані направляються в процесор. По восьмирозрядній шині введення-виведення даних між процесором і системою портів введення-виведення одночасно передаються вісім бітів даних.

Команди введення-виведення та приклади їх виконання

Для роботи з вказаними вище шинами існують спеціальні команди, які виконуються процесором і називаються командами введення-виведення.

Команда OUT (ВИВЕСТИ)

Нехай, наприклад, необхідно вивести вміст регістра AL через порт 0023H. Для цього використовується команда OUT, при якій виконуються такі дії:

1) на шині введення-виведення / зчитування-запису встановлюється значення 1;
2) число 0023H розміщується на шину адреси порту;
3) вміст AL розміщується на шину введення-виведення даних;
4) система портів введення-виведення приймає дані по шині введення-виведення даних і виводить їх через порт 0023H.

Команда IN (ВВЕСТИ)

За цією командою здійснюється обернена операція. Нехай необхідно ввести дані через порт 0024H та занести їх у регістр AL. При цьому виконуються такі дії:

1) на шині введення-виведення / зчитування-запису встановлюється значення 0;
2) число 0024H розміщується на шину адреси порту;
3) через систему портів введення-виведення дані, які надійшли через порт 0024H, передаються на шину введення-виведення даних, яка зв'язує систему портів і процесор;
4) дані пересилаються з шини введення-виведення даних у AL.

Приклад використання

Як приклад використання операцій введення-виведення розглянемо пересилку літери G на пристрій друку. Оскільки кожний символ представляється одним байтом (у випадку ASCII-кодування), помістимо в регістр AL байт даних (ASCII-код), який відповідає літері G. Потім перемістимо літеру G на друкуючий пристрій, виводячи вміст регістру AL через порт 378H.

По команді виведення біти, які представляють літеру G, розміщуються на шину введення-виведення даних, яка зв'язує процесор із системою введення-виведення. Адреса порту (378H) розміщується на шину адреси порту, а на керуючій шині введення-виведення / зчитування-запису встановлюється значення 1. Система введення-виведення пересилає літеру G у порт 378H, до якого приєднано друкувальний пристрій.

Отримавши дані, друкувальний пристрій повідомляє обчислювальну машину про успішне завершення їх передачі, посилаючи нульовий байт на шину, яка з'єднує його з машиною. Ця восьмирозрядна шина підключена до іншого порту, наприклад із номером 379H. Програма, яка переслала символ на друкуючий пристрій, виконує зчитування з порту 379H по команді введення: *ввести дані з порту 379H і помістити їх у регістр* AL.

Програма може перевірити вміст регістра AL, щоб визначити, чи правильно виконана пересилка символу на друкувальний пристрій: якщо значення регістра AL нульове, операція виведення завершена успішно, і виконання програми продовжується.

Інтерфейси периферійних пристроїв

Комп'ютери використовуються в комбінації з різними периферійними пристроями. Типовими прикладами таких пристроїв є клавіатура, миша, дисплей, принтер, накопичувач на гнучких дисках, модем та ін. Ці пристрої за допомогою інтерфейсів приєднуються до шини (рис. 11.1).

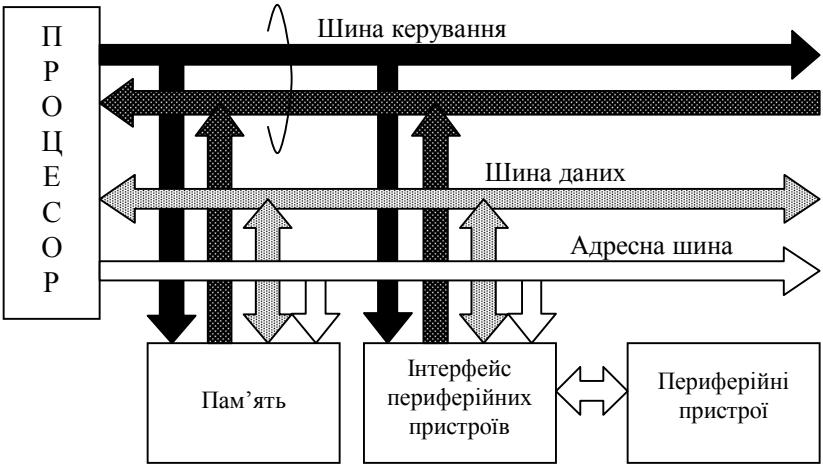


Рис. 11.1. Підключення до процесора периферійних пристроїв

Для найбільш простого з'єднання периферійного пристрою з шиною комп'ютера слід використовувати таку систему портів введення-виведення, при якій порт введення являє собою ряд тристабільних вентилів, а порт виведення є регістром (рис. 11.2). Це найпростіша апаратна реалізація інтерфейсу. Всі сигнали, які комп'ютеру необхідно отримати від периферійного пристрою, надходять до нього через порт введення. Всі сигнали, які комп'ютер повинен передати периферійному пристрою, виводяться через порт виведення. Прийом і передача сигналів повинні здійснюватись під керуванням програмних засобів. Однак такий спосіб реалізації ускладнює програмне забезпечення і, крім того, знижує продуктивність всієї системи. І навпаки: якщо є спеціальні апаратні засоби, які автоматично виконують прийом і передачу сигналів, програмне забезпечення спрощується, а продуктивність системи зростає (див. лабораторну роботу № 12).

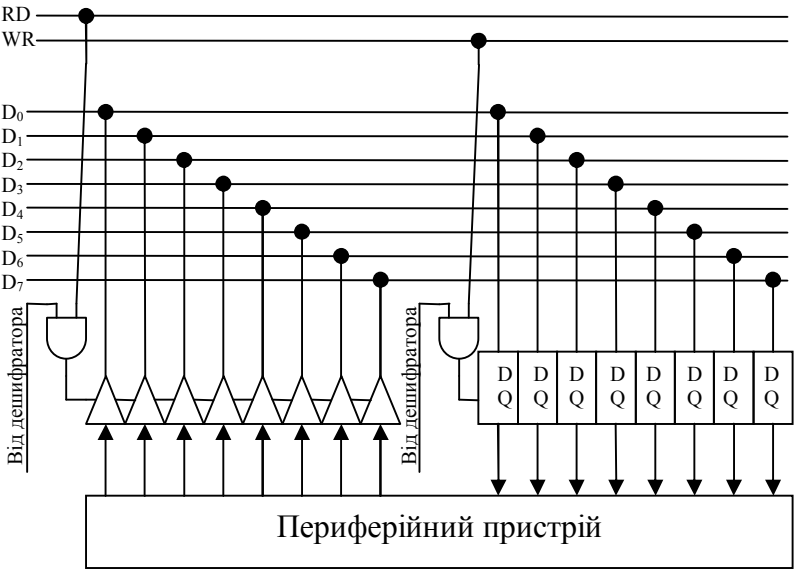


Рис. 11.2. З'єднання процесора з периферійним пристроєм (адресна шина подається на дешифратор, отже, адреса порту збігається з номером відповідної вихідної лінії дешифратора)

Практична частина

Як приклад для дослідження роботи мікропроцесора 8086 при взаємодії з портами введення-виведення спроектуємо схему мікрокалькулятора (рис. 11. 3). Інформація з матричної клавіатури (код натиснутої клавіші) буде вводитись у мікропроцесор. Процес введення організовано з використанням механізму переривань. Схема, зображена на рис 11.4, опитує матричну клавіатуру, шляхом формування коду „біжуча одиниця", отримує код натиснутої клавіші та генерує сигнал немаскованого апаратного переривання NMI при натисканні. Процедура зчитування та аналізу коду клавіші організована у вигляді підпрограми обробки переривання (див. текст програми). Основна робота мікропроцесора полягає в постійному виведенні вмісту регістра-акумулятора (комірки пам'яті OPER_1) з метою висвітлення операндів та результатів матрицею семисегментних світлодіодних індикаторів мультиплексорного типу (рис. 11. 3).

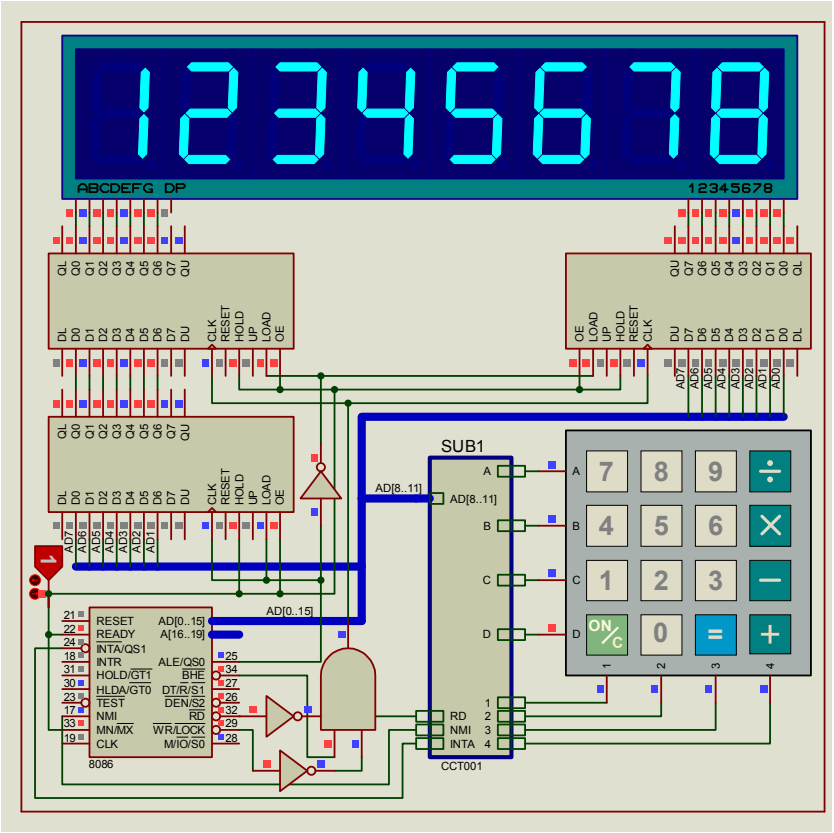


Рис. 11. 3. Вигляд схеми мікрокалькулятора на основі МП 8086

Оскільки адресна шина (молодші 16 розрядів) і шина даних у випадку мікропроцесора 8086 суміщені, а часове розділення адрес/даних здійснюється синхронно з керуючим сигналом ALE, то слід використовувати регістри-фіксатори із відповідним з'єднанням виводу ALE мікропроцесора до входів дозволу завантаження даних у регістри (рис. 11. 3). Зокрема, адреси записуються у лівий регістр, а дані – у правий.

Як видно з рис. 11. 3, виведення інформації з мікропроцесора відбувається через молодший байт даних (AD[0..7]), що відповідає парним портам введення-виведення, а введення – через старший (AD[8..15]), що відповідає непарним портам.

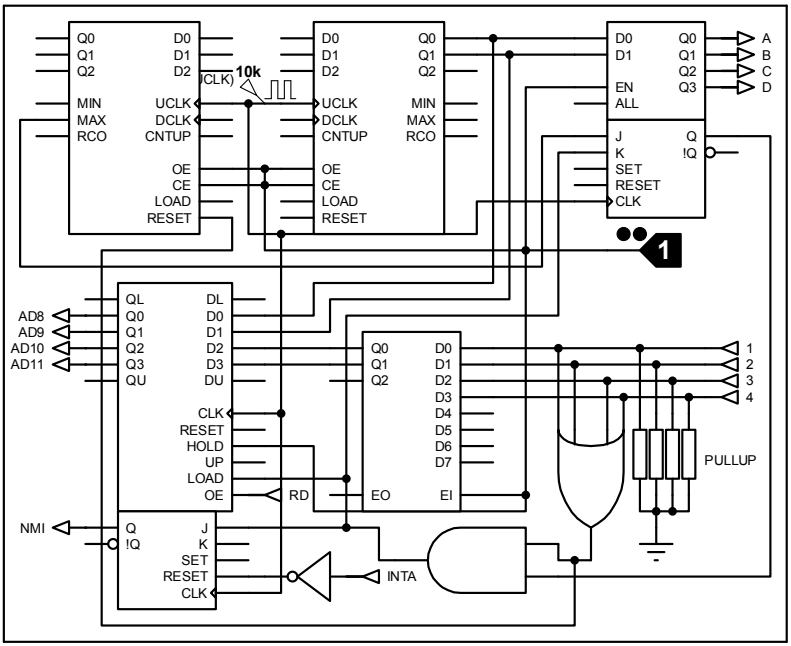


Рис. 11. 4. Вміст підсхеми SUB1 – схема опитування матричної клавіатури, формування коду натиснутої клавіші та сигналу переривання

Під час виконання команди OUT активується лінія запису WR мікропроцесора, а при виконанні команди IN – лінія читання RD.

У програмі, наведеній нижче, введення інформації відбувається при виконанні команди IN AX,1 – запис у регістр AX даних із порту 1. Оскільки номер порту – непарне число, то у молодшу частину регістра (AL) буде занесено старший байт 16-розрядної шини даних (AD[8..15]), тобто код натиснутої клавіші. Оскільки код – 4-розрядне число, то можна використати молодшу тетраду старшого байта, тобто лінії AD[8..11] (рис. 11.3, 11.4).

Виведення інформації відбувається при виконанні команди OUT DX, AX – запис у порт з адресою DX вмісту регістра AX. Адреса порту являє собою 7-сегментний код для висвітлення цифри і формується командами XLATB та XCHG DX,AX. Вміст регістра AX – код „біжучий нуль”: MOV AX,07FH; ROR AL,1.

Програма для мікропроцесора 8086

```
data segment
7_SEG DB 0FCH,060H,0DAH,0F2H,
   066H,0B6H,0BEH,0E0H,0FEH,0F6H
KEYPAD DB 7,4,1,10,8,5,2,0,9,6,3,11,
                        15,14,13,12
OPER_1 DB 9 DUP(0)
OPER_2 DB 9 DUP(0)
HELP_O DB 18 DUP(0)
DIGIT_FLAG DB 0
OPER_FLAGS DB 0
O_F DB 0
FIRST_OPER DB 0
DIV_ZERO DB 0
CMP_DIV DB 0
ends
stack segment
DW 128 dup(0)
ends
code segment
start:IN AX,1
XOR AX,AX
MOV ES,AX
MOV BX,8
MOV SI,OFFSET[READ_KEY]
MOV ES:[BX],SI
ADD BX,2
MOV AX,CS
MOV ES:[BX],AX
MOV AX,data
MOV DS,AX
MOV ES,AX
MOV AX,07FH
CIRCLE:MOV CX,8
MOV DIGIT_FLAG,0
C2:PUSH AX
MOV BX,CX
MOV AL,OPER_1[BX-1]
CMP AL,0
JE CHECK_DF
MOV DIGIT_FLAG,1
JMP SHOW_DIGIT
CHECK_DF:CMP DIGIT_FLAG,1
JE SHOW_DIGIT
CMP BX,1
JE SHOW_DIGIT
JMP NOT_SHOW_DIGIT
SHOW_DIGIT:LEA BX,7_SEG
XLATB
XCHG DX,AX
POP AX
OUT DX, AX
JMP NOT_POP
NOT_SHOW_DIGIT:POP AX
NOT_POP:ROR AL,1
LOOP C2
JMP CIRCLE
READ_KEY:PUSH AX
PUSH BX
PUSH CX
IN AX,1
LEA BX,KEYPAD
XLATB
CMP AL,10
JS NOT_OF
MOV O_F,1
NOT_OF:CMP AL,10
JNE N1
CALL ON_P
JMP FULL
N1:CMP AL,11
JNE N2
CALL EQU_P
JMP FULL
N2:CMP AL,12
JS N3
SUB AL,11
MOV OPER_FLAGS,AL
MOV FIRST_OPER,0
JMP FULL
N3:CMP O_F,0
JE READ_DIGIT
CALL CLEAR_OP_2
CALL O1_XCHG_O2
READ_DIGIT:MOV O_F,0
CMP OPER_1[7],0
JNE FULL
MOV CX,7
SHIFT:MOV BX,CX
MOV AL,OPER_1[BX-1]
MOV OPER_1[BX],AL
LOOP SHIFT
IN AX,1
LEA BX,KEYPAD
XLATB
MOV OPER_1[0],AL
FULL:POP CX
POP BX
POP AX
IRET
ON_P:PUSH CX
PUSH BX
MOV CX,9
C3:MOV BX,CX
MOV OPER_1[BX-1],0
MOV OPER_2[BX-1],0
LOOP C3
MOV OPER_FLAGS,0
MOV FIRST_OPER,0
MOV O_F,0
CALL CLEAR_HELP
POP BX
POP CX
RET
EQU_P:CMP OPER_FLAGS,0
JE E4
CMP FIRST_OPER,0
JNE NFO
CALL O1_XCHG_O2
MOV FIRST_OPER,1
NFO:CMP OPER_FLAGS,1
JNE E1
CALL ADD_P
E1:CMP OPER_FLAGS,2
JNE E2
CALL SUB_P
E2:CMP OPER_FLAGS,3
JNE E3
CALL MUL_P
E3:CMP OPER_FLAGS,4
JNE E4
CALL DIV_P
E4:RET
ADD_P:PUSH AX
PUSH BX
PUSH CX
MOV CX,8
XOR BX,BX
MOV AH,0
C5:MOV AL,OPER_1[BX]
ADD AL,AH
ADD AL,OPER_2[BX]
MOV AH,0
AAA
MOV OPER_1[BX],AL
INC BX
LOOP C5
POP CX
POP BX
POP AX
RET
SUB_P:PUSH AX
PUSH BX
PUSH CX
MOV CX,8
XOR BX,BX
MOV AH,0
C6:MOV AL,OPER_1[BX]
ADD AL,AH
SUB AL,OPER_2[BX]
MOV AH,0
AAS
MOV OPER_1[BX],AL
INC BX
LOOP C6
POP CX
POP BX
POP AX
RET
MUL_P:PUSH AX
PUSH BX
PUSH CX
PUSH DI
MOV CX,8
XOR DI,DI
CALL CLEAR_HELP
C_7:PUSH CX
MOV CX,8
XOR BX,BX
C7:MOV AL,OPER_1[BX]
MUL OPER_2[DI]
```

```
AAM
ADD AL,HELP_O[BX+DI]
AAA
MOV HELP_O[BX+DI],AL
ADD AH,HELP_O[BX+DI+1]
XCHG AH,AL
MOV AH,0
AAA
MOV HELP_O[BX+DI+1],AL
ADD HELP_O[BX+DI+2],AH
INC BX
LOOP C7
INC DI
POP CX
LOOP C_7
CALL MOV_HELP_TO_OPER1
POP DI
POP CX
POP BX
POP AX
RET
DIV_P:PUSH AX
PUSH BX
PUSH CX
PUSH DX
CALL CHECK_DIV_ZERO
CMP DIV_ZERO,0
JE END_DIV
CALL PREP_DIV
XOR DX,DX
NEXT_SHIFT_L:CALL CMP_OPER
CMP CMP_DIV,0
JE NOT_SHIFT_L
INC DX
MOV CX,8
C8:MOV BX,CX
MOV AL,OPER_2[BX-1]
MOV OPER_2[BX],AL
LOOP C8
MOV OPER_2[0],0
JMP NEXT_SHIFT_L
NOT_SHIFT_L:CMP DX,0
JE NOT_SHIFT_R
NEXT_SHIFT_R:DEC DX
MOV CX,8
XOR BX,BX
C_8:INC BX
MOV AL,OPER_2[BX]
MOV OPER_2[BX-1],AL
LOOP C_8
MOV OPER_2[8],0
NOT_SHIFT_R:CALL CMP_OPER
CMP CMP_DIV,0
JE NOT_SUB
CALL SUB_AT_DIV
MOV BX,DX
INC OPER_1[BX]
JMP NOT_SHIFT_R
NOT_SUB:CMP DX,0
JE NOT_ON
JMP NEXT_SHIFT_R
END_DIV:CALL ON_P
NOT_ON:POP DX
POP CX
POP BX
POP AX
RET
O1_XCHG_O2:PUSH CX
PUSH BX
PUSH AX
PUSH DX
MOV CX,8
C4:MOV BX,CX
MOV AL,OPER_1[BX-1]
MOV DL,OPER_2[BX-1]
MOV OPER_1[BX-1],DL
MOV OPER_2[BX-1],AL
LOOP C4
POP DX
POP AX
POP BX
POP CX
RET
CLEAR_HELP:PUSH CX
PUSH BX
MOV CX,18
XOR BX,BX
C9:MOV HELP_O[BX],0
INC BX
LOOP C9
POP BX
POP CX
RET
CLEAR_OP_2:PUSH CX
PUSH BX
MOV CX,9
XOR BX,BX
C_9:MOV OPER_2[BX],0
INC BX
LOOP C_9
POP BX
POP CX
RET
MOV_HELP_TO_OPER1:PUSH CX
PUSH BX
PUSH AX
MOV CX,8
XOR BX,BX
C10:MOV AL,HELP_O[BX]
MOV OPER_1[BX],AL
INC BX
LOOP C10
POP AX
POP BX
POP CX
RET
PREP_DIV:PUSH CX
PUSH BX
PUSH AX
MOV CX,8
XOR BX,BX
C11:MOV AL,OPER_1[BX]
MOV HELP_O[BX],AL
MOV OPER_1[BX],0
INC BX
LOOP C11
POP AX
POP BX
POP CX
RET
CHECK_DIV_ZERO:PUSH CX
PUSH BX
MOV CX,8
MOV DIV_ZERO,0
C12:MOV BX,CX
CMP OPER_2[BX-1],0
JE NOT_ZERO
MOV DIV_ZERO,1
NOT_ZERO:LOOP C12
POP BX
POP CX
RET
CMP_OPER:PUSH CX
PUSH BX
PUSH AX
MOV CX,9
MOV CMP_DIV,1
C13:MOV BX,CX
MOV AL,HELP_O[BX-1]
CMP AL,OPER_2[BX-1]
JE EQU_1_2
JNC END_CMP
MOV CMP_DIV,0
JMP END_CMP
EQU_1_2:LOOP C13
END_CMP:POP AX
POP BX
POP CX
RET
SUB_AT_DIV:PUSH AX
PUSH BX
PUSH CX
MOV CX,8
XOR BX,BX
MOV AH,0
C14:MOV AL,HELP_O[BX]
ADD AL,AH
SUB AL,OPER_2[BX]
MOV AH,0
AAS
MOV HELP_O[BX],AL
INC BX
LOOP C14
POP CX
POP BX
POP AX
RET
ends
end start
```

Програма для мікропроцесора 8086, окрім організації введення-виведення інформації, забезпечує виконання операцій додавання, віднімання, множення та ділення двох цілочислових додатних операндів, представлених у 8-позиційному розпакованому двійково-десятковому форматі. У зв'язку з цим, відповідні підпрограми виконання арифметичних операцій, окрім команд ADD, SUB, MUL, використовують команди двійково-десяткової корекції: AAA, AAS та AAM. Підпрограма ділення побудована на основі методу ділення з відновленням залишку, тобто на принципах пробних віднімань (CMP), віднімань із корекцією, інкременту позиції результату при черговому відніманні та зсувів BCD-чисел.

Контрольні запитання та завдання

1. Що таке порти введення-виведення і як вони організовані у мікропроцесорі 8086? Яку кількість портів введення-виведення підтримує мікропроцесор 8086?
2. Опишіть особливості функціонування мікропроцесора під час виконання команд IN та OUT.
3. Яке основне призначення інтерфейсів периферійних пристроїв? Опишіть найпростішу схему інтерфейсного пристрою.
4. Як відбувається відокремлення адрес та даних, що передаються по одній спільній шині у мікропроцесорі 8086?
5. Програма для мікропроцесора 8086 схеми мікрокалькулятора не передбачає сигналізації переповнення розрядної сітки (відображаються молодші вісім позицій результату) та ділення на нуль (відбувається скидання калькулятора). Модифікуйте програму в такий спосіб, щоб відбувалась сигналізація переповнення та ділення на нуль, наприклад висвітлювалось би повідомлення „Error" (Помилка).
6. Розроблена схема мікрокалькулятора використовує механізм апаратних переривань. Як слід модифікувати схему та програму, щоб уникнути використання переривань?
7. Модифікуйте схему мікрокалькулятора з метою уникнення використання команди XLATB та змініть програму.

Лабораторна робота № 12
Дослідження роботи інтерфейсних пристроїв на прикладі програмованого паралельного інтерфейсу 8255

Теоретична частина

Як зазначалося у попередній лабораторній роботі, програмна реалізація передачі інформації між мікропроцесором та периферійними пристроями ускладнює програмне забезпечення і, крім того, знижує продуктивність всієї системи. При наявності спеціальних апаратних засобів, які автоматично виконують прийом і передачу сигналів, програмне забезпечення спрощується, а продуктивність системи зростає. Такі апаратні засоби називають інтерфейсними пристроями.

Характеристики інтерфейсу з боку периферійного пристрою повинні бути узгоджені з характеристиками цього пристрою. Іноді виникає проблема, яка полягає в тому, що дуже часто однотипні периферійні пристрої мають індивідуальні відмінності, тому з метою уніфікації інтерфейсної схеми використовується метод визначення цих відмінностей програмним шляхом. Інтерфейс, в якому дана проблема розв'язується подібним способом, називається програмованим інтерфейсом. Відмінності визначаються з допомогою запису відповідних бітових комбінацій у передбачені для цього регістри інтерфейсної схеми.

Для приведення в дію складних периферійних пристроїв (таких, як накопичувач на гнучких дисках) зазвичай необхідне складне керування, при цьому бажано, щоб таке керування здійснювалось інтерфейсом автоматично. Для цього є декілька команд, одна з яких вказується мікропроцесором, який записує необхідну команду в регістр команд, що знаходиться в інтерфейсній схемі. Все інше виконується інтерфейсом автоматично. Інтерфейс такого роду називають контролером.

Контролер здійснює декілька функцій: він може безперервно керувати деякими периферійними пристроями одночасно з функціонуванням процесора, обробляти дані, які циркулюють між центральним процесором і периферійним пристроєм, і реагувати на команди, які надходять із центрального процесора,

виконуючи більш складні послідовності дій по керуванню периферійним пристроєм.

Периферійний пристрій може здійснювати підготовку даних для передачі їх у комп'ютер або вимагати видачу даних. Як приклад розглянемо взаємодію процесора і клавіатури. Під час роботи процесор не "знає", в який момент часу оператор натисне клавішу. Можна було б змусити процесор постійно опитувати стан клавіатури: після натискання якої-небудь клавіші процесор зчитував би відповідний їй код та виконав би деяку просту програму обробки цього коду. Однак під час виконання цієї програми може бути натиснута ще одна клавіша. Оскільки процесор зайнятий обробкою попереднього коду, він може пропустити натискання наступної клавіші. Щоб виключити подібні ситуації, між процесором і клавіатурою вводять контролер: останній сприйме код, повідомить про це процесор і продовжить зчитування кодів із клавіатури, поки процесор буде виконувати відповідну програму. Таким чином, використання контролера гарантує, що жодне натискання клавіші не буде пропущено.

Контролер може виконувати обробку даних, які циркулюють між пристроєм і процесором. Наприклад, деякі пристрої працюють із послідовно організованими даними. Процесор завжди розрахований на паралельну організацію даних. Контролер зчитує послідовні дані з пристрою і перед тим, як передати ці дані в процесор, перетворить їх в паралельні. До таких пристроїв належить, наприклад, накопичувач на дисках. Контролер накопичувача на дисках веде паралельний прийом даних від процесора, послідовно розміщує їх на однорозрядну шину та відправляє в накопичувач.

Програмований паралельний інтерфейс

Програмований паралельний інтерфейс (ППІ) 8255 призначений для введення-виведення паралельної інформації у 8-бітовому форматі, що дозволяє реалізувати більшість відомих протоколів обміну по паралельних каналах. Мікросхема ППІ може використовуватися для з'єднання мікропроцесора зі

стандартним периферійним устаткуванням (дисплеєм, телетайпом, накопичувачем тощо).

Структурну схему ППІ показано на рис. 12.1. Можна виділити такі блоки:

- двонапрямлений 8-розрядний буфер даних *Buffer of Data* (*BD*), що з'єднує лінії даних мікросхеми із системною шиною даних;
- блок керування зчитуванням/записом *Read/Write Control Unit* (*RWCU*), що забезпечує керування зовнішнім і внутрішнім передаванням даних і керуючих слів;
- три 8-розрядні порти введення-виведення (*Port A, Port B, Port C*) для обміну інформацією, причому порт *C* поділений на два 4-розрядні: *C'* (*PC*7 – *PC*4) і *C"* (*PC*3 – *PC*0). Порти *A* і *C'* об'єднані у групу *A*, порти *B* і *C"* – у групу *B*.

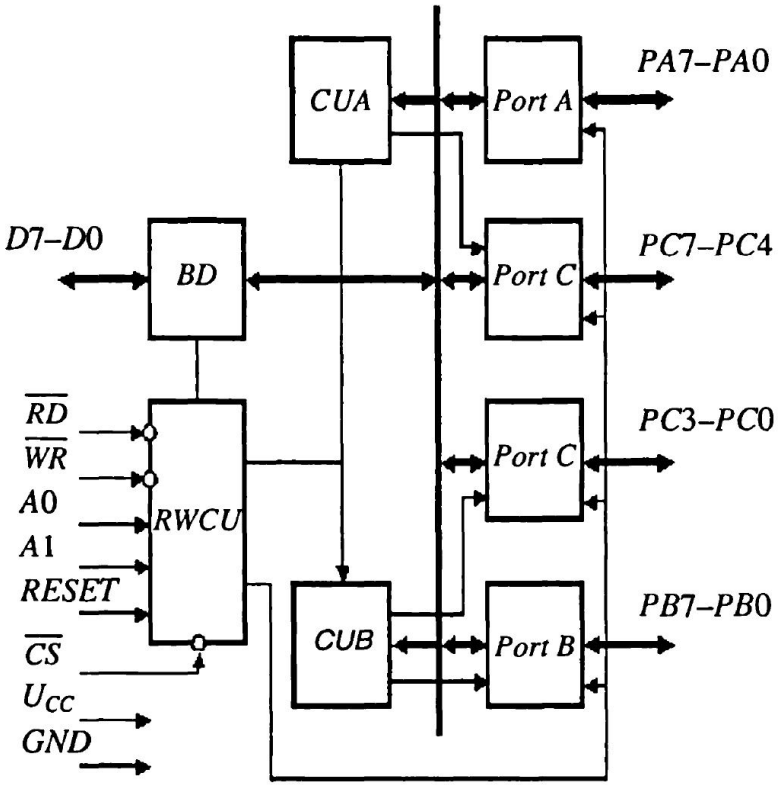


Рис. 12. 1. Структурна схема програмованого паралельного інтерфейсу

Схема ППІ містить також блоки керування групою *A Control Unit A* (*CUA*) та групою *B* (*CUB*), що формують сигнали керування для відповідних груп.

Блок *RWCU* (*Register of Control Word Unit*) містить регістр керуючого слова, який зберігає керуючі слова, що надходять від мікропроцесора.

Адресні розряди *A*1, *A*0 дозволяють обирати один із портів або регістр керуючого слова *RCW*. Зокрема, при *A*1=0, *A*0=0 обирається порт *A*, при *A*1=0, *A*0=1 – порт *B*, при *A*1=1, *A*0=0 – порт *C*, при *A*1=1, *A*0=1 – регістр керуючого слова *RCW*.

Сигнал керування третім станом шини даних *CS*, сигнал зчитування *RD*, сигнал запису *WR* та сигнал скидання *RESET* подаються на блок *RWCU* і разом з адресними сигналами *A*0, *A*1 задають вид операції, що виконується.

Програмування ППІ полягає у завантаженні керуючого слова режиму при *A*1=1, *A*0 = 1. Формат керуючого слова режиму показано на рис. 12.2. Керуюче слово визначає один із трьох режимів портів паралельного інтерфейсу: режим 0 – основний режим введення-виведення, режим 1 – режим введення-виведення за сигналом готовності, режим 2 – режим двонаправленої передачі інформації.

*D*7	*D*6	*D*5	*D*4	*D*3	*D*2	*D*1	*D*0
1	*M*1	*M*0	*IOA*	*IOC'*	*M*	*IOB*	*IOC"*

Рис. 12. 2. Формат керуючого слова режиму

На рис. 12.2 позначено:

- біти *M*1, *M*0 задають режим групи *A*. При *M*1*M*0 = 00 – режим 0, при 01 – режим 1, при 10 або 11 – режим 2;
- біт *IOA* задає режим уведення або виведення порту *A* (1 – введення інформації, 0 – виведення);
- біт *IOC'* задає режим уведення або виведення порту *C'* (1 – введення інформації, 0 – виведення);
- біт *M* задає режим групи *B* (0 – режим 0, 1 – режим 1);
- біт *IOB* задає режим уведення або виведення порту *B* (1 – введення інформації, 0 – виведення);

- біт *IOC"* задає режим уведення або виведення порту *C"* (1 – введення інформації, 0 – виведення).

Керуюче слово може встановлювати різні режими роботи для кожного з портів. Порт *A* може працювати в будь-якому з трьох режимів, порт *B* – у режимах 0 та 1. Порт *C* можна використовувати для передачі даних тільки в режимі 0, в інших режимах його застосовують для передачі керуючих сигналів, що супроводжують процес обміну по портах *A* і *B*.

Практична частина

Як приклад для дослідження роботи програмованого паралельного інтерфейсу 8255 спроектуємо схему введення-виведення деякої символьної інформації (рис. 12. 3). Інформація між периферійними пристроями введення-виведення (матрична клавіатура 4×4 і матриця світлодіодів 8×8) та мікропроцесором 8086 передається через мікросхему програмованого паралельного інтерфейсу 8255, яка працює в режимі 0.

Інформація з матричної клавіатури буде вводитись у мікропроцесор за допомогою порту *C* інтерфейсної мікросхеми. Ця інформація являє собою код „біжучий нуль", що подається на входи матричної клавіатури (молодша тетрада), та набір сигналів із виходу клавіатури (старша тетрада). При натисканні клавіші в момент часу, що відповідає появі нульового сигналу у відповідному рядку, відбудеться формування нульового сигналу у відповідному стовпці. Мікропроцесор періодично зчитує інформацію з порту *C* інтерфейсної схеми та аналізує її. У випадку натискання клавіші значення старшої тетради зчитаного байта даних буде відмінне від 0FH і мікропроцесор перейде до обчислення коду натиснутої клавіші (див. текст програми).

Код натиснутої клавіші, помножений на 8, відповідає адресі бітового образу відповідного символу. Набір бітових образів для висвітлення символів кнопок клавіатури асоціюється із символьним ім'ям „S". Використовуючи базово-індексну адресацію фрагменти символів по рядкам виводяться у порт *B*.

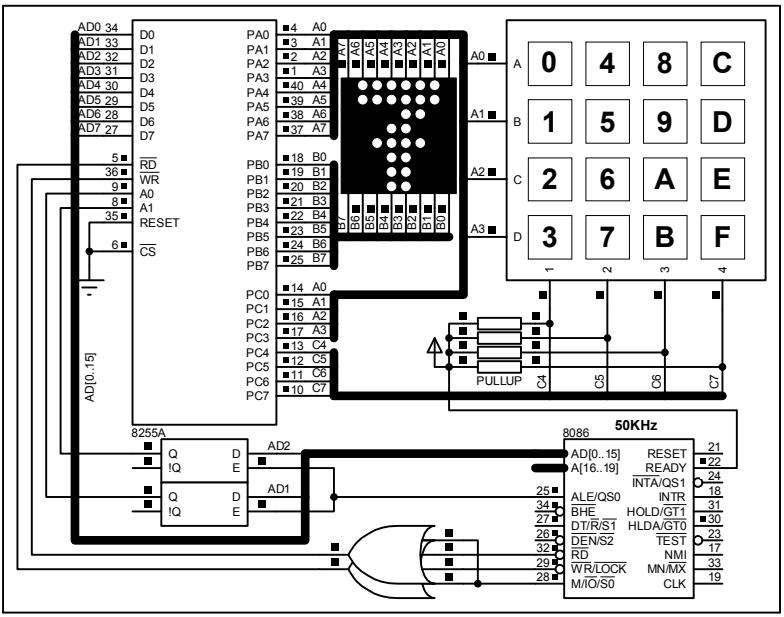


Рис. 12. 3. Вигляд схеми введення-виведення символьної інформації на основі МП 8086 та ППІ 8255

У порт *A* виводиться код „біжучий нуль” для забезпечення висвітлення символу рядок за рядком матрицею світлодіодів 8×8 мультиплексорного типу. Крім того, молодша тетрада байта, що посилається у порт *A*, подається на входи матричної клавіатури для сканування рядків.

Як було зазначено у попередній лабораторній роботі, адресна шина (молодші 16 розрядів) та шина даних у випадку мікропроцесора 8086 суміщені, а часове розділення адрес/даних здійснюється синхронно з керуючим сигналом ALE, тому слід використовувати регістри-фіксатори з відповідним з’єднанням виводу ALE мікропроцесора до входів дозволу завантаження даних у регістри. У нашому випадку для адресних розрядів *A*1, *A*0 мікросхеми 8255 використовуються два D-тригера (рис. 12. 3). Для блокування запису в інтерфейсну мікросхему під час адресації пам’яті використано елементи „АБО” при формуванні сигналів RD і WR.

Програма для мікропроцесора 8086

```
org 100h
JMP BEGIN
S DB 3CH,7EH,0E7H,0C3H,0C3H,0E7H,7EH,3CH
  DB 18H,38H,78H,18H,18H,18H,18H,3CH
  DB 3CH,7EH,66H,0CH,18H,30H,7EH,7EH
  DB 3CH,7EH,66H,0CH,0CH,66H,7EH,3CH
  DB 0CH,1CH,3CH,6CH,0FFH,0FFH,0CH,0CH
  DB 7EH,7EH,60H,7CH,7EH,06H,7EH,7CH
  DB 3CH,7EH,60H,7CH,7EH,66H,7EH,3CH
  DB 7EH,7EH,0CH,18H,3CH,18H,18H,18H
  DB 3CH,7EH,66H,3CH,7EH,66H,7EH,3CH
  DB 3CH,7EH,66H,7EH,3EH,06H,7EH,3CH
  DB 18H,3CH,66H,0C3H,0FFH,0FFH,0C3H,0C3H
  DB 7CH,7EH,66H,7CH,7CH,66H,7EH,7CH
  DB 3CH,7EH,0C3H,0C0H,0C0H,0C3H,7EH,3CH
  DB 7CH,7EH,67H,63H,63H,67H,7EH,7CH
  DB 7EH,7EH,60H,7EH,7EH,60H,7EH,7EH
  DB 7EH,7EH,60H,7EH,7EH,60H,60H,60H
BEGIN:MOV AL,7FH
PUSH AX
MOV CX,8
XOR BX,BX
C1:IN AL,4
MOV DH,AL
MOV DL,AL
SHR DH,4
CMP DH,15
JZ NOT_KEY
PUSH CX
MOV CX,4
XOR AH,AH
C2:TEST DH,1
JZ NOT_INC_H
INC AH
SHR DH,1
LOOP C2
NOT_INC_H:SHL AH,2
MOV CX,4
XOR AL,AL
C3:TEST DL,1
JZ NOT_INC_L
INC AL
SHR DL,1
LOOP C3
NOT_INC_L:ADD AL,AH
XOR AH,AH
MOV SI,AX
SHL SI,3
POP CX
NOT_KEY:MOV AL,0FFH
OUT 0,AL
MOV AL,S[BX+SI]
OUT 2,AL
POP AX
OUT 0,AL
ROR AL,1
PUSH AX
INC BX
LOOP C1
JMP BEGIN
END
```

Контрольні запитання та завдання

1. Обґрунтуйте необхідність використання інтерфейсних пристроїв. Що таке контролер?
2. Укажіть призначення та опишіть режими роботи мікросхеми програмованого паралельного інтерфейсу (ППІ) 8255.
3. Назвіть можливі комбінації ввімкнення портів ППІ 8255.
4. Які порти ППІ можуть працювати у всіх можливих режимах?
5. Запишіть керуюче слово для роботи паралельного інтерфейсу в режимі 0 при налагодженні портів *A* і *B* на виведення, порту *C* – на введення даних.
6. Організуйте введення даних за перериванням (рис. 12. 3).
7. Переробіть схему, спроектовану у практичні частині даної лабораторної роботи, та програму для мікропроцесора, з метою введення-виведення, наприклад, 16-ти символів українського алфавіту або будь-якого іншого набору з 16-ти символів (оберіть варіант, запропонований викладачем).

Список літератури

1. Таненбаум Э. Архитектура компьютера. – 4-е изд. – СПб.: Питер, 2003. – 697 с.
2. Поворознюк А. И. Архитектура компьютеров. Часть 1. Архитектура микропроцессорного ядра и системных устройств. – Харьков: Торнадо, 2004. – 355 с.
3. Поворознюк А. И. Архитектура компьютеров. Часть 2. Архитектура внешней памяти, видеосистемы и внешних интерфейсов. – Харьков: Торнадо, 2004. – 296 с.
4. Гук М. Ю. Аппаратные средства IBM PC. Энциклопедия. – 3-е изд. – СПб.: Питер, 2006. – 1072 с.
5. Кулаков В. Программирование на аппаратном уровне : специальный справочник. – СПб.: Питер, 2003. – 847 с.
6. Голубь Н. Г. Искусство программирования на Ассемблере. Лекции и упражнения. – 2-е изд. – СПб.: ООО «ДиаСофтЮП», 2002. – 656 с.
7. Ревич Ю.В. Занимательная электроника. – СПб.: БХВ-Петербург, 2005. – 672 с.
8. Эрл Д. Гейтс. Введение в электронику. Серия «Учебники и учебные пособия». – Ростов-на-Дону: «Феникс», 1998. – 640 с.
9. Хоровиц П., Хилл У. Искусство схемотехники. – 5-е изд. – М.: Мир, 1998. – 704 с.
10. Точчи Р., Уидмер Н. Цифровые системы. Теория и практика. – М.: Издательский дом «Вильямс», 2004. – 1024 с.
11. Бабич М.П., Жуков І.А. Комп’ютерна схемотехніка: Навчальний посібник. – К.: МК-Прес, 2004. – 412 с.
12. Якименко Ю. І., Терещенко Т. О. Мікропроцесорна техніка. – К.: Політехніка, 2004. – 440 с.
13. Титце У., Шенк К. Полупроводниковая схемотехника: Справочное руководство. – М.: Мир, 1982. – 512 с.
14. Хамахер К., Вранешич З., Заки С. Организация ЭВМ – 5-е изд. – СПб.: Питер, 2003. – 848 с.
15. Ковригин Б. Н. Проектирование процессора ЭВМ: учебно-методическое пособие. – М.: МИФИ, 2006. –72 с.

Навчальне видання

Архітектура комп'ютерів
Лабораторний практикум

Укладач *Жихаревич Володимир Вікторович*

Відповідальний за випуск *Остапов С.Е.*
Літературний редактор *Колодій О.В.*

Друкарня видавництва "Рута"
Чернівецького національного університету ім. Ю. Федьковича
58012, Чернівці, вул. Коцюбинського, 2