МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Державний університет інтелектуальних технологій і зв'язку

Кафедра комп'ютерно-інтегрованих технологічних процесів та виробництв

О.М. Воробйова, Ю.В. Флейта

ПРОМИСЛОВА ЕЛЕКТРОНІКА

Навчальний посібник

Частина ІІ

Одеса – 2021

УДК 681.32 План НМВ 2021 р.

Воробйова О.М. Промислова електроніка: навч. посіб. Ч. II / О.М. Воробйова, Ю.В. Флейта. – Одеса : ДУІТЗ, 2021. – 82 с.

У навчальному посібнику розглядаються основи теорії, принципи дії, технічні показники та сфери застосування напівпровідникових приладів, оптоелектронних елементів та систем, інтегральних мікросхем та мікропроцесорних комплектів. На основі цієї елементарної бази наведено принципи побудови, робота, розрахунок та вибір елементів при проєктуванні пристроїв промислової електроніки. Дано аналіз фізичних процесів у розглядуваних пристроях відносно розробок пристроїв промислової електроніки.

УХВАЛЕНО
на засіданні кафедри КІТ П і В
і рекомендовано до друку.
Протокол № від р.

ЗАТВЕРДЖЕНО
методичною радою ДУІТЗ.
Протокол № 1 від 11.03.2021 р.

ЗМІСТ

ВСТУП

Цифрова техніка стала потужним стимулом для швидкого розвитку мікросхемотехніки.

Одне з досягнень технології напівпровідників полягає *у* створенні великих інтегральних схем (ВІС), тобто схем, що дозволяють розміщувати значну кількість транзисторів, скажімо 10^6 або більше, на одній кремнієвій прокладці (на одному «кристалі»). Це називають *високим ступенем інтеграції.* Власне, висока ступінь інтеграції і стала поштовхом до появи до мікропроцесорів.

Мікропроцесори – це нові надзвичайні логічні пристрої, які помітно впливають на наше життя. Мікропроцесори можна виявити в кишенькових калькуляторах, касових апаратах магазинів, побутових і наукових приладах, в обладнанні контор, медичному обладнанні, телеіграх – і це далеко не повний перелік їх застосування. Більше того, кожного дня з’являються нові додатки і розробляються нові вироби на основі мікропроцесорів. їх потенційний вплив на наше життя майже не піддається уяві.

Мікропроцесор – це програмований логічний пристрій, виготовлений за ВІС-технологією. Як ми побачимо далі, у конструкцію мікропроцесора закладена велика гнучкість. Сам по собі він не може розв’язати ту чи іншу конкретну задачу, щоб це вирішити, його потрібно запрограмувати і з'єднати з іншими пристроями. До їх числа зазвичай входять пам'ять і пристрої введення/виведення. Взагалі, певна сукупність з'єднаних один з одним системних пристроїв, що включає й мікрокомпресор, пам'ять і пристрої введення/виведення, спрямована на виконання певної чітко визначеної функції, що називається *мікропроцесорним комплексом.*

Одним із самих необхідних вузлів мікропроцесорного комплексу є запам’ятовувальні пристрої. Ці мікросхеми (ВІС) випускаються промисловістю в окремих корпусах і безпосередньо в корпусі мікропроцесора.

1. ЗАПАМ’ЯТОВУВАЛЬНІ ПРИСТРОЇ

1.1. Класифікація запам’ятовувальних пристроїв

Запам’ятовувальний пристрій (ЗП), або *пам'ять*, – це пристрій, за допомогою якого інформація існує в часі, точніше, передається від одного моменту до іншого. Напрямок передачі, звичайно, збігається з напрямом руху реального часу, а саме з рухом вперед. Отже, ЗП можна вважати каналом, який дозволяє передати у майбутнє інформацію, що згенерувала сьогодні.

Усі послідовнісні схеми і комп'ютери у тому числі, мають властивість запам'ятовування, і це дозволяє їхнім виходам на даний момент залежати від входів у попередні моменти. Взагалі кажучи, цією властивістю схеми зобов'язані *запам'ятовувальним елементам*, наприклад тригерам. У цьому розділі в центрі нашої уваги будуть структури запам'ятовувальних пристроїв, що складаються з великої кількості запам'ятовувальних елементів з деякою регулярною структурою.

Організація запам'ятовувального пристрою визначає способи передавання інформації у пристрій і з нього. Зазвичай інформація передається порціями, що складаються з фіксованого числа бітів і званими *словами*. ЗП можна уявляти собі у вигляді деякого простору, що складається з безлічі ідентифікованих позицій для розміщення слів.

У деяких ЗП на кожну таку позицію відводяться свої фіксовані запам'ятовувальні елементи. У цьому випадку місце розташування запам'ятовувальних елементів однозначно визначає позицію слова, звану *осередком*. В інших ЗП слова переміщуються відносно множини запам'ятовувальних елементів, зберігаючи упорядкованість відносно один одного. У цьому випадку позиція слова ідентифікується як часом, так і місцем розташування запам'ятовувальних елементів. В усіх випадках, коли слово інформації передається у ЗП, воно поміщається в деяку конкретну позицію. Цей процес називається *записом у пам'ять*. З іншого боку, коли інформація передається з пам'яті, вона також вибирається з деякої конкретної позиції (зазвичай інформація в цій позиції зберігається). Цей процес називається *зчитуванням з пам'яті*.

Існують різні способи вибору тієї позиції, для якої провадиться операція запису або читання. Засоби вибору позиції і передачі інформації в позицію або з неї утворюють засоби доступу (або вибірки).

ЗП поділяються на два головних типи. ЗП з *довільним доступом* та ЗП з *послідовним доступом*. До першого типу відносять ЗП, в яких доступ до будь-якої позиції вимагає приблизно одного і того ж самого часу. Іншими словами, ми можемо навмання обрати позицію, і це не відіб'ється на часі, який витрачається на читання або запис. До другого типу відносять ЗП, доступ до яких можливий лише у певному порядку.

Пам'ять з довільним доступом – це такий ЗП, в якому елемент даних, який запам’ятований в осередку, може бути безпосередньо зчитаний. Час, необхідний для вибірки даного осередку, виявляється приблизно тим самим, що

і для будь-якого іншого осередку. Кожен осередок містить фіксоване число запам'ятовувальних елементів і має свій ідентифікуючий номер. Ідентифікуючий номер, що складається з фіксованого числа бітів, називається *адресою осередку*. Наявність адрес дозволяє розрізняти осередки при зверненні до них для виконання операцій запису і читання.

Причому вибір осередку для роботи не залежить від місце розташування цього осередку. Вибір здійснюється за заданою адресою.

Що ж стосується осередків ЗП з послідовнісним доступом, то час підключення осередку дуже залежить від його місцезнаходження. Зчитування завжди розпочинається з початку відліку першого осередку ЗП і послідовно місцезнаходження необхідного осередку відшукується шляхом зсуву пошуку у нарощуваному порядку.

У сучасній цифровій техніці, у тому числі і у мікропроцесорній, використовуються різноманітні пристрої пам’яті. Для зберігання одного біту даних треба мати один запам’ятовувальний елемент (ЗЕ) наприклад – тригер. Зберігання невеликих масивів кодових слів можна здійснювати за допомогою регістрів. Але для зберігання десятків кодових слів використання регістрів призводить до невиправданих апаратних витрат, тому для зберігання великих масивів слів будуються запам’ятовувальні пристрої (ЗП) із використанням спеціальних мікросхем, у кожній із яких може зберігатися тисячі біт.

Для зберігання багаторозрядного числа, яке має обсяг кілька біт треба мати комірку пам’яті (КП), з кількома запам’ятовувальними елементами. При наявності якогось числа комірок пам’яті у складі запам’ятовувального пристрою виникає необхідність встановлення порядку вибору комірки пам’яті, до якої проводиться звертання з метою запису або зчитування даних.

Існують два типи ЗП: динамічні та статичні.

Динамічні ЗП використовують у ролі запам’ятовувачів конденсатори. Ці ЗП мають той недолік, що через неминучі струми витікання конденсатора, останній після запису розряджається, а при зчитуванні зарядженого стану частина заряду розтікається по шині зчитування. Тому динамічну пам’ять час від часу треба поновлювати (регенерувати). Схема регенерації є досить складною. Через це застосування динамічних ЗП обмежене.

Статичні ЗП будуються на базі тригерів. Основною перевагою статичних ЗП є те, що записана інформація, тобто стан тригера, з часом експлуатації та при зчитуванні не руйнується. Тому статичні ЗП здобули найбільшого поширення.

ЗП будують, як правило, за матричним принципом. На відміну від ПЛМ, на перетинах шин включають запам’ятовувальні комірки або статичні, або динамічні, тому й ЗП носять назву або статичних, або динамічних. Найбільшого розповсюдження набули статичні ЗП. Тому динамічні ЗП далі не розглядаються.

За функціональною ознакою інтегральні ЗП розподіляють на три класи: оперативні та постійні та зовнішні.

Оперативні запам’ятовувальні пристрої (ОЗП) призначені для запису, тимчасового зберігання та зчитування інформації. ОЗП виконують операції

запису й зчитування за наявності напруги живлення. Щодо зберігання інформації, то в одних ОЗП воно здійснюється під дією напруги, а в інших – енергонезалежно.

Постійні запам'ятовувальні пристрої (ПЗП) призначені для тривалого зберігання записаної інформації та її зчитування. Зміст запису в ПЗП не змінюється під час експлуатації і за відсутності напруги живлення не руйнується. ПЗП бувають двох типів: одноразового запису та такі, що допускають поновлення занесеної інформації декілька разів. Цей процес реалізують за допомогою спеціальних пристроїв – програматорів.

Існують два типи ЗП: динамічні та статичні.

Динамічні ЗП використовують у ролі запам'ятовувачів конденсатори. Ці ЗП мають той недолік, що через неминучі струми витікання конденсатора, останній після запису розряджається, а при зчитуванні зарядженого стану частина заряду розтікається по шині зчитування. Тому динамічну пам'ять час від часу треба поновлювати (регенерувати). Схема регенерації є досить складною. Через це застосування динамічних ЗП обмежене.

Статичні ЗП будуються на базі тригерів. Основною перевагою статичних ЗП є те, що записана інформація, тобто стан тригера, з часом експлуатації та при зчитуванні не руйнується. Тому статичні ЗП здобули найбільшого поширення.

ЗП будують, як правило, за матричним принципом. На відміну від ПЛМ, на перетинах шин включають запам'ятовувальні комірки або статичні, або динамічні, тому й ЗП носять назву або статичних, або динамічних. Найбільшого розповсюдження набули статичні ЗП. Тому динамічні ЗП далі не розглядаються.

За функціональною ознакою інтегральні ЗП розподіляють на два класи: оперативні та постійні.

Оперативні запам'ятовувальні пристрої (ОЗП) призначені для запису, тимчасового зберігання та зчитування інформації. ОЗП виконують операції запису й зчитування за наявності напруги живлення. Щодо зберігання інформації, то в одних ОЗП воно здійснюється під дією напруги, а в інших – енергонезалежно.

Постійні запам'ятовувальні пристрої (ПЗП) призначені для тривалого зберігання записаної інформації та її зчитування. Зміст запису в ПЗП не змінюється під час експлуатації і за відсутності напруги живлення не руйнується. ПЗП бувають двох типів: одноразового запису та такі, що допускають поновлення занесеної інформації декілька разів. Цей процес реалізують за допомогою спеціальних пристроїв – програматорів.

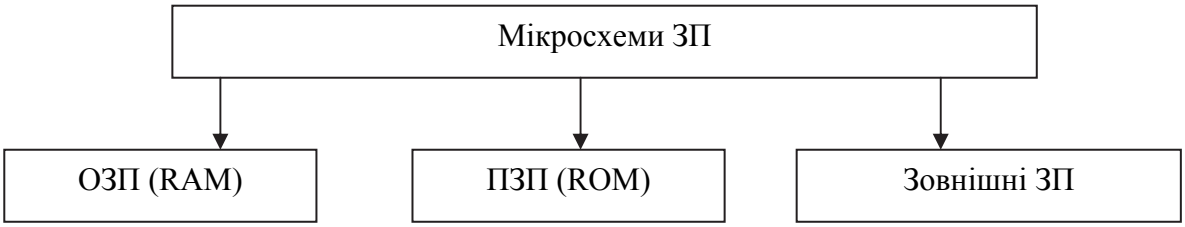


Рис. 1.1. Класифікація ЗП

Зовнішні запам'ятовувальні пристрої – для зберігання великих об'ємів даних і програм, організації архівів, бібліотек, банків даних тощо.

Запам'ятовувальні пристрої зовнішньої пам'яті, як правило реалізуються у вигляді окремих периферійних пристроїв, які підключають до мікропроцесорної системи через узгоджуючий пристрій з'єднання *(інтерфейс)*. Окрім цього, застосовуються велика кількість *буферних запам'ятовувальних пристроїв* для узгодження часових параметрів у процесі обміну інформацією між пристроями.

За способом звернення ЗП класифікують на адресні та безадресні.

Матриця безадресних ЗП має всього дві шини – запису і зчитування, тобто запис або зчитування може здійснюватись тільки для одного запам'ятовувача.

Щодо багатьох запам'ятовувачів, то для них організують адресну вибірку, згідно з якою шукану комірку знаходять за номером стовпчика і рядка, тобто за адресою, яка має вигляд комбінації *n*-розрядного двійкового числа. Такі ЗП є адресними.

Адресні ЗП мають спеціальні адресні входи і можуть бути з довільним або з послідовним зверненнями. У перших пошук інформації здійснюється шляхом безпосереднього звернення до перетину шин, адрес яких задається двійковим числом. У других – за допомогою послідовної вибірки при збільшенні або при зменшенні адресного числа.

У *безадресних ЗП* звернення виконується незалежно від координат запам'ятовувачів, тобто не за адресою, а за певними ознаками самої інформації, що міститься в запам'ятовувачі ЗП.

За технологією виконання ЗП поділяються на пристрої, де використовується схемотехніка ТТЛ, ЕЗЛ, МОН, КМОН та $І^2Л$.

1.2. Параметри ЗП

Основні параметри ЗП залежать від технології виготовлення і визначаються так само, як і для інших мікросхем. Це наступні параметри: навантажувальна здатність, споживана потужність, завадостійкість, числова величина логічних рівнів. До найважливіших параметрів ЗП належать місткість і швидкодія.

Кількість інформації, що може зберігатися у ЗП, визначає його *місткість*. ЗП складаються з декількох комірок пам'яті N, кожна з яких може зберігати слово з визначеним числом розрядів n. Отже, місткість задається або добутком числа запам'ятовувачів N на розрядність n слів: $M = N \cdot n$, або у розкритій формі M. Наприклад, для першого випадку позначення ЗП у формі 32 х 8 біт визначає, що ЗП здатний зберігати 32 слова по 8 розрядів, тобто 32 байти або 256 біт. Для другого випадку місткість ЗП визначається лише у вигляді 32 байти або 256 біт, не розкриваючи при цьому кількість слів та кількість розрядів.

Швидкодія ЗП характеризується часом повного циклу звернення і часом вибірки або часом доступу до пам'яті. Період звернення – це мінімально

припустимий час між двома черговими зверненнями до ЗП. Цей параметр залежить від характеристики та властивостей ЗП.

До швидкодії надоперативної і оперативної пам'яті, а також буферних запам'ятовувальних пристроїв пред'являються підвищені вимоги. Ці види пам'яті будуються на основі напівпровідникових інтегральних мікросхем на біполярних і МОН-транзисторах.

До швидкодії зовнішніх запам'ятовувальних пристроїв вимоги трохи нижчі, але вони повинні мати велику ємність і відносно низьку вартість на одиницю зберігання даних. Під час виготовлення таких ЗП широко використовуються магнітні запам'ятовувальні елементи різноманітних типів. На їх основі формуються накопичувачі, які дозволяють зберігати великі масиви даних.

Час вибірки – це інтервал часу між моментом подачі сигналу вибірки до появи інформації на виході ЗП. Для ВІС ЗП час вибірки наближається до пікосекундного діапазону.

Кожна мікросхема ЗП характеризується потужністю споживання, набором напруг живлення, типом корпусу. Мікросхеми ПЗП мають додаткові параметри: часову тривалість зберігання інформації, по закінченню якої інформація може самовільно змінитися, та максимальну кількість циклів перезапису, після чого мікросхема визначається нездатною до використання.

1.3. Оперативні запам'ятовувальні пристрої

Оперативні ЗП (ОЗП) – це невіддільні елементи цифрових приладів, які використовуються для тимчасового зберігання інформації і без яких не може бути жодної ЕОМ. У цифрових системах зв'язку на ОЗП зберігають дані, які потрібні, наприклад, для перетворення кодів та іншої різноманітної обробки сигналів. ОЗП поділяють на статичні й динамічні, в залежності від запам'ятовувальної комірки, що включається на перетинах шин. ОЗП виконують з використанням практично усіх відомих технологій: ТТЛ, ТТЛШ, ЕЗЛ, МОН, КМОН, $І^2Л$.

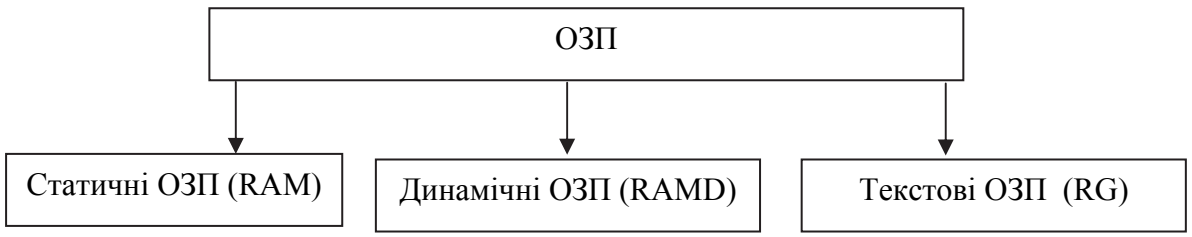


Рис. 1.2. Класифікація мікросхем пам'яті ОЗП

1.3.1. Статичні запамятовувачі ВІС ОЗП (RAM)

Роль *статичних запам'ятовувачів* у ВІС ОЗП відіграють двостанові комірки пам'яті – тригери. Фізичний стан тригерних структур під час звернення не руйнується. Вони крім зберігання одного біта інформації (1 або 0)

дозволяють здійснювати операцію запису *(WR)* або зчитування *(RD)*. Такі запам'ятовувачі дозволяють будувати ВІС ОЗП, які є найбільш розповсюдженими в цифровій та мікропроцесорній техніці і застосовуються як регістри оперативної та буферної пам'яті.

На рис. 1.3 наведена схема статичного запам'ятовувача на тригері *D3*.

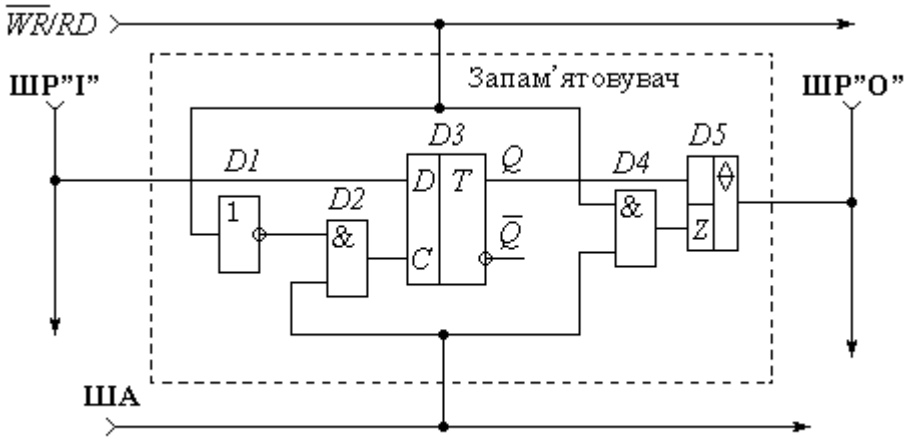


Рис. 1.3. Статичний запам'ятовувач

Тут ШР"І" – шина запису даних;

ШР"О" – шина зчитування даних;

$\overline{WR}/RD$ – шина запису-зчитування;

ША – шина адреси.

Запам'ятовувач працює в одному з трьох режимів: запису інформації, її зберігання та зчитування. Інвертор *D1* з ключем *D2* керують режимом запису. Ключ *D4* через драйвер *D5* керує режимом зчитування.

Ці режими здійснюються наступним чином.

Запам'ятовувач зберігає інформацію при ША = 0 за межами часу t_1 та t_2 (рис. 1.4), бо на виході ключа *D2*, тобто на вході синхронізації тригера $C = 0$, через що запис неможливий.

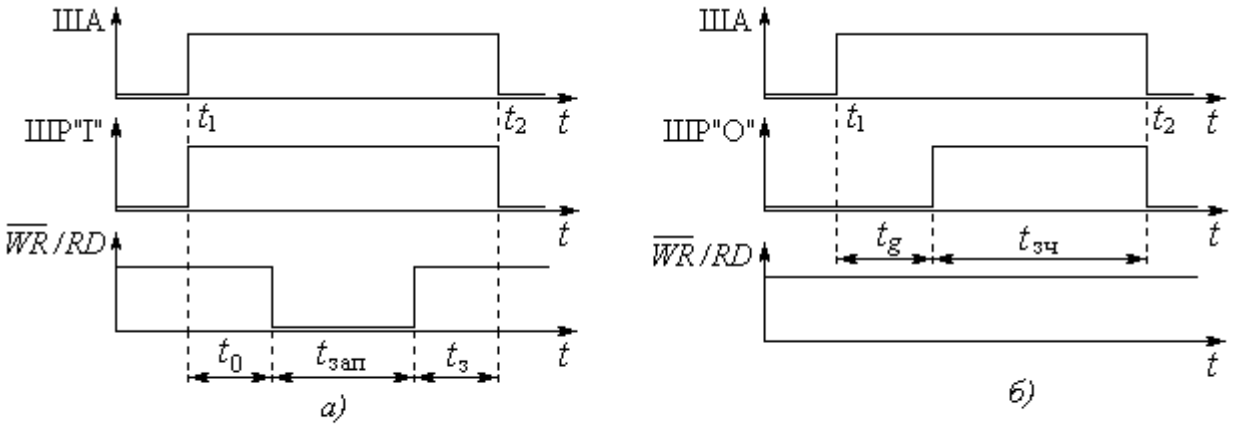


Рис. 1.4. Часова діаграма роботи запам'ятовувача: а) в режимі запису; б) в режимі зчитування

Зчитування при ША = 0 також неможливе, бо завдяки ключу *D4* вхід дозволу драйвера *D5* нульовий ($Z = 0$), через що на виході *D5* високий опір,

тобто розрядна шина вихідних даних ШР “О” від’єднується від прямого виходу *Q* тригера *D3*. Так здійснюється режим зберігання.

Режими запису та зчитування досягаються при одиничному стані шини адреси (ША = 1). При цьому через ключ *D2* дається дозвіл на запис, а через ключ *D4* – дозвіл на зчитування. Послідовність подачі сигналів у режимах запису та зчитування має бути такою, що наведена на рис. 1.4.

Запис досягається в активному стані запам’ятовувача (ША = 1) при $\overline{WR}/RD = 0$ протягом $t_{зап}$. За час t_0 до запису треба виставити дані ШР “І”. Протягом $t_{зап}$ обидва входи ключа *D2* одиничні. Тому $C = 1$ і саме тим стан розрядної шини вхідних даних ШР “І” через вхід тригера *D* записується в тригер. Запис слід припинити за час затримки $t_з$ до моменту t_2.

Зчитування досягається в активному стані запам’ятовувача (ША = 1) при $\overline{WR}/RD = 1$. Цей рівень треба установити за час t_g до зчитування. При цьому ключ *D2* запирається, припиняючи запис, а обидва входи ключа *D4* одиничні. Тому вхід дозволу $Z = 1$ драйвера *D5* здійснює з’єднання прямого виходу *Q* тригера *D3* з розрядною шиною вихідних даних ШР “О”.

1.3.2. Динамічні запам’ятовувачі ВІС ОЗП (RAMD)

У динамічних пристроях пам’яті зберігання даних здійснюється за рахунок заряду спеціально сформованих у структурі напівпровідника конденсаторів. Такі запам’ятовувальні елементи мають більш просту структуру в порівнянні із тригерними і займають меншу площу на напівпровідниковому кристалі, але у процесі зберігання даних потребують періодичного відновлення стану (регенерації). Іноді система регенерації вмонтована у запам’ятовувальний пристрій і працює автоматично.

1.3.3. Структура ВІС ОЗП

У структурі ВІС ОЗП запам’ятовувачі компонуються у прямокутну матрицю, яку називають накопичувачем ЗП. Залежно від способу звернення ($\overline{WR}/RD$) до запам’ятовувачів розрізняють два найбільш поширених типи організації накопичувачів:

– з однокоординатною або послівною вибіркою;

– з двокоординатною або паралельною вибіркою.

Структурна схема накопичувача з однокоординатною вибіркою наведена на рис. 1.5.

Кожний рядок накопичувача – це адресна шина ($ША_1$, $ША_2$, … $ША_N$), а стовпець – дві розрядні шини: шина запису ШР “І” та шина зчитування ШР “О”.

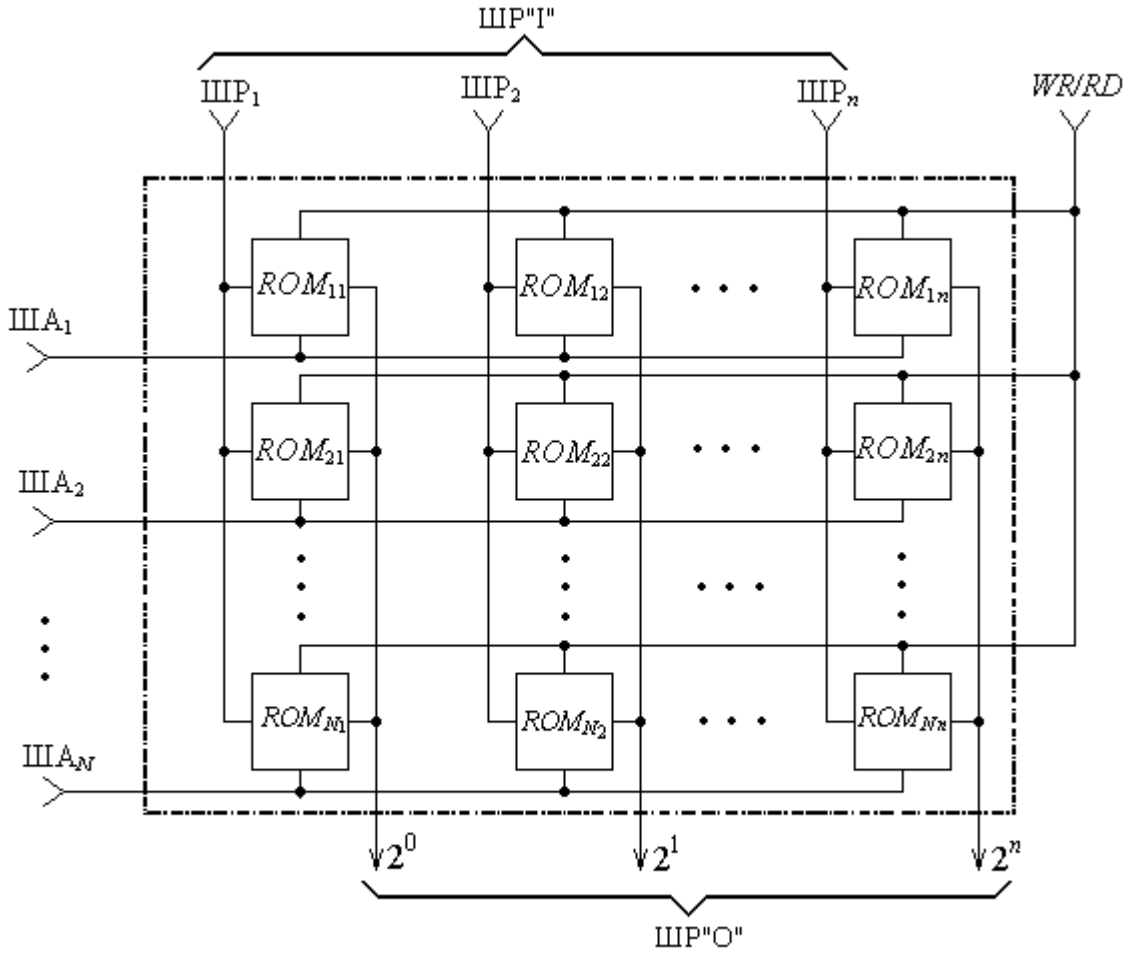


Рис. 1.5. Схема ОЗП з одно координатною вибіркою

У кожному перетині матриці ставиться запам’ятовувач, який може зберігати один біт інформації. Запам’ятовувачі, що згруповані по горизонталях, здатні запам’ятовувати *n* біт (розрядів) одного слова, бо з’єднані *однією* адресою. Кількість слів, яку може запам’ятовувати матриця, визна-чається числом горизонтальних рядків (адресних шин $ША_1$, $ША_2$, … $ША_N$). Кожне слово вибирається шляхом ініціації відповідної адресної шини.

Матриця *N* слів, кожне з яких має розрядність *n*, визначається основ-ним вузлом ЗП, який називають *накопичувачем інформації.* Накопичувач інформації з однокоординатною або послівною вибіркою характеризується тим, що має лише одну координату звернення до запам’ятовувачів, а саме – номер рядка накопичувача і тому має лише один дешифратор адреси.

Типова структурна схема ВІС ОЗП (накопичувача інформації) з *одно-координатною* вибіркою наведена на рис. 1.6.

З метою мінімізації кількості адресних входів ОЗП інформація щодо адреси надходить у вигляді двійкового числа і подається на дешифратор адреси *DC*, що входить до складу ОЗП. На виході дешифратора адреси ініціюється тільки одна шина, яка вибирає для запису або зчитування лише адреси один рядок, тобто одне слово.

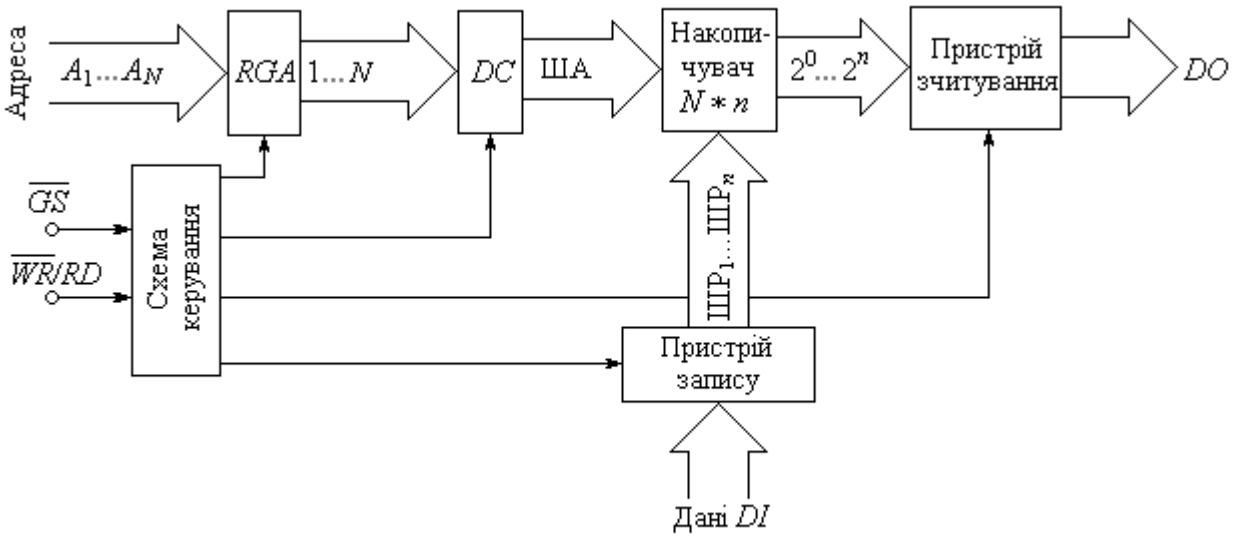


Рис. 1.6. Структурна схема ВІС ОЗП з одно координатною вибіркою

ОЗП має число $N \cdot n$ запам'ятовуючих елементів ROM_{Nn}, які розміщені по N рядках та n стовпцях. Кожний рядок утворює слово з номером N.

При однокоординатній вибірці пошук кожного слова з номером i здійснюється відповідною однією шиною адреси $ША_i$, бо ОЗП має шини адреси $ША_1$, $ША_2$, …, $ША_i$, …, $ША_N$, що з'єднані відповідно з кожним запам'ятовувачем ROM_{ij} однойменного i -го слова.

Розрядні вхідні шини $ШР_j$ з'єднують входи запису кожного розряду усіх N слів. Вихідними є шини 2^0, 2^1, …, 2^n, які з'єднують виходи зчитування кожного розряду усіх N слів.

Для всіх $N \cdot n$ запам'ятовувачів ROM_{ij} існує загальна шина звернення $\overline{WR}/RD$, сигнал на якій визначає режим запису або зчитування.

Схема працює наступним чином.

Сигнал вибірки адреси в кожний момент часу може з'явитися лише на одній з адресних шин $ША_i$. Запис або зчитування відбувається за наявності на вході $\overline{WR}/RD$ необхідного рівня (1 або 0). Зчитування RD (від англ. *Read* – читання) виконується при надходженні сигналу одиничного рівня, а запис WR (від англ. *Writ* – писати), коли на шині установлюється нульовий рівень.

Для запису слова в комірки запам'ятовувачів ROM_{ij} слід активізувати i-й рядок накопичувача, тобто адресну шину $ША_i$, а на шину $\overline{WR}/RD$ подати сигнал дозволу на запис інформації, яким є логічний 0, через що всі шини $ШР_1$, $ШР_2$, … , $ШР_n$ будуть підключені до входів запису i-го рядка.

Для зчитування слова з комірок запам'ятовувачів ROM_{ij} слід активізувати i-й рядок накопичувача, тобто адресну шину $ША_i$, а на шину $\overline{WR}/RD$ подати сигнал дозволу на зчитування інформації, яким є логічна одиниця 1, через що на всіх шинах 2^0, 2^1, …, 2^{n-1} установляться відповідні значення логічного нуля або логічної одиниці зчитаного N-го слова.

Розглянутий тип накопичувача інформації має лише одну координату звернення до запам'ятовувачів, а саме – номер рядка накопичувача і тому має лише один дешифратор адреси. Такий накопичувач з однокоординатною

вибіркою називають ще двовимірним (типу 2*D*), бо запам'ятовувачі в ньому розміщені на площині.

До складу ВІС ОЗП, крім основного вузла – накопичувача, входять регістр адреси *RGA*, дешифратор адресних шин *DC*, схема керування, пристрій запису даних *DI* і пристрій зчитування даних *DO*.

Інформаційні сигнали, які треба записувати, надходять по шині *DI*. Зчитування інформації відбувається за допомогою пристроїв, які комутуються до шин *DO*. Регістр адреси *RGA* може бути відсутнім. Тоді адресний вхід мікросхеми ВІС ОЗП безпосередньо зв'язаний з входом дешифратора *DC*.

Накопичувач з двокоординатною вибіркою (рис. 1.7) має дві адресні шини: горизонтальні по рядках (ША*X*) і вертикальні по стовпцях (ША*Y*).

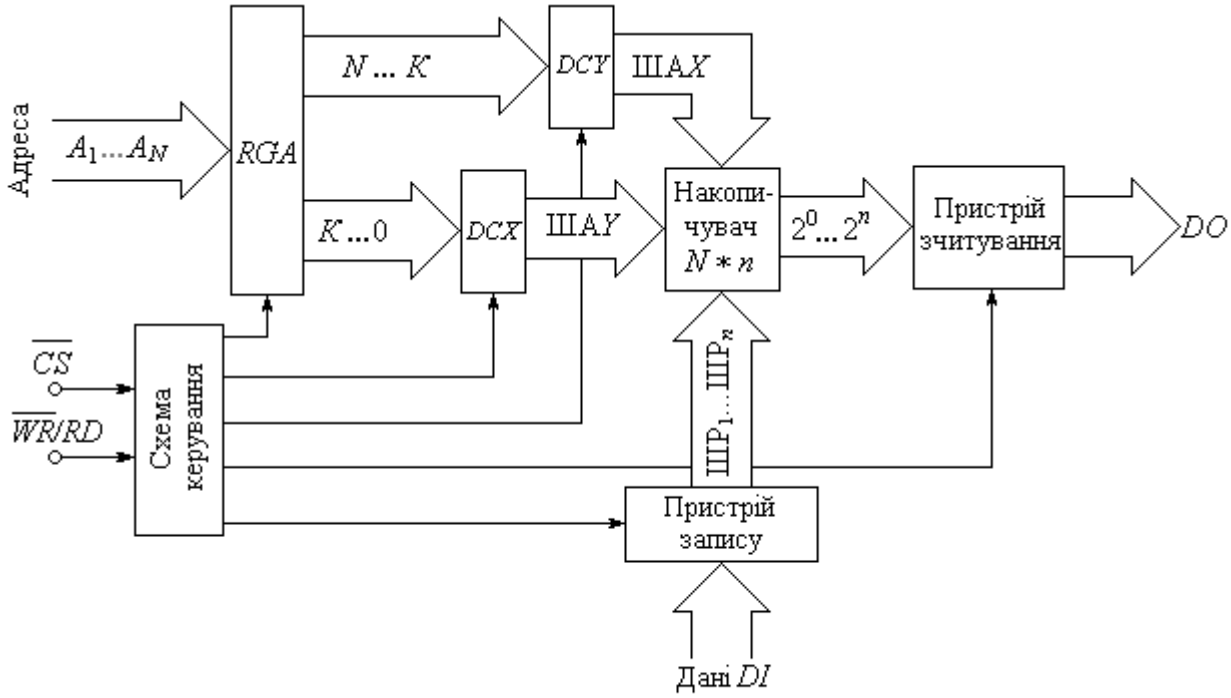


Рис. 1.7. Структурна схема ВІС ОЗП з двокоординатною вибіркою

Накопичувач з такою структурою має тривимірну будову і складається з *n* накопичувачів розрядів, кожний з яких містить *N* запам'ятовувачів. Тому кожний накопичувач відповідає тільки за один однойменний розряд усіх *N* слів, а кожне слово записується, зберігається і зчитується в усіх *n* накопичувачах за ідентичною двокоординатною адресою x_i, y_j.

Обидві структурні схеми відрізняються лише наявністю одного (рис. 1.6) дешифратора адреси, або двох (рис. 1.7), що носять назви: *DCX* – дешифратор адресних шин строк ША*X*; *DCY* – дешифратор адресних шин стовпців ША*Y*.

Для запису слова у запам'ятовувач за адресою x_i, y_j на вході адресних дешифраторів ША*Y* і ША*X* (рис. 1.7) подаються відповідні адреси.

Інформаційне двійкове слово надходить на спільні розрядні шини ШР через пристрій запису, а зчитування записаного слова здійснюється через пристрій зчитування.

Структурна схема має два керуючих входи: запис/зчитування $\overline{WR}/RD$ та додатковий – $\overline{CS}$.

Керуючий вхід $\overline{CS}$, що носить назву “вибір мікросхеми” – це вхід дозволу проходження сигналу.

Вхідні схеми ВІС ОЗП – це формувачі на логічних елементах, що забезпечують узгодження накопичувача з вхідними зовнішніми пристроями за струмом та напругою.

Вихідні каскади ВІС ОЗП – це логічні елементи з відкритим колектором або тристанові буфери, які забезпечують каскадування ВІС у системах з шинною організацією передавання даних.

На рис. 1.8 показані умовні позначення статичних ВІС ОЗП з паралельними входами й виходами (рис. 1.8,*а*) та послідовними входом і виходом (рис. 1.8,*б*).

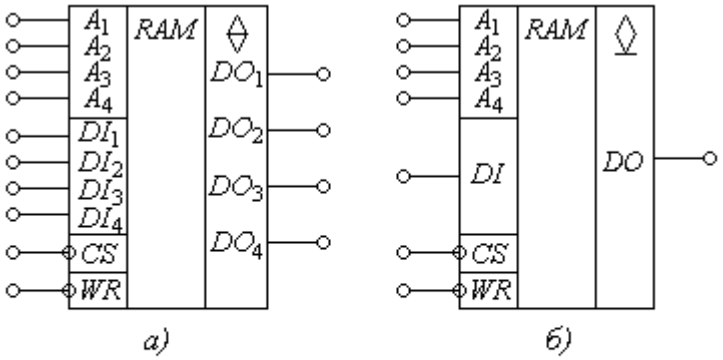


Рис. 1.8. Умовне позначення ВІС ОЗП:
а) з паралельними входами і виходами;
б) з послідовними входом і виходом

Мікросхема статичного чотирирозрядного ОЗП (рис. 1.8,*а*) має паралельні входи DI_j і виходи DO_j з тристановим буфером і з організацією пам’яті місткістю (16 × 4) біт, тобто 64 біт.

Мікросхема статичного чотирирозрядного ОЗП (рис. 1.8,*б*) має послідовні вхід *DI* і вихід *DO* даних з відкритим колектором.

1.3.4. Інформаційні та керуючі сигнали ВІС ОЗП

Обмін інформацією з ОЗП здійснюється по шинах або магістралях, що поділяються на три групи: шина даних, адресна шина, шина керуючих сигналів.

Шина даних – це набір сигнальних ліній, за допомогою яких передається паралельно двійкова інформація. Число ліній, що утворюють шину, визначають максимальну довжину числа, яке можна передавати в двійковій формі. Шини даних бувають 8-, 16- і 32-розрядними. Символами *DI* позначається шина запису даних, а *DO* – шина зчитування даних.

Шина даних може бути *двонапрямленою*, тобто для взаємного обміну даними різні пристрої підключаються до єдиного проводу – шини, як показа- но на рис. 1.9. По цій шині відбувається обмін інформацією між робочими блоками пристроїв.

Робочий блок – це будь-який вузол, до якого надходять дані EI_1 … EI_N, наприклад, ОЗП. Виходи робочих блоків через елементи & з відкритим колектором підключені до числової шини *D*. До других входів елементів & надходять керуючі сигнали (сигнали звернення) EO_1 … EO_N. При $EO_i = 0$

вихідний транзистор елемента & з відкритим колектором закритий і вихід блока від'єднується від шини.

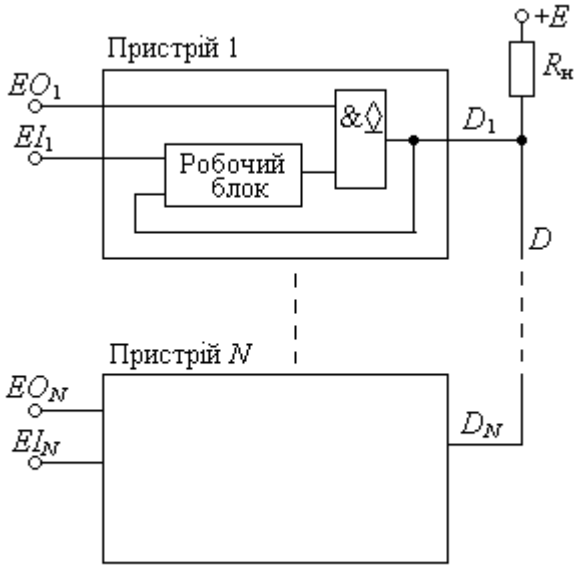


Рис. 1.9. Організація шини даних

Якщо на керуючому вході EO_i присутня логічна одиниця, то вихідний сигнал відповідного *i*-го робочого блоку пропускається до шини *D*.

У кожний відрізок часу одиничне значення сигналу $EO_i = 1$ має подаватися лише на один пристрій. Якщо одиничні рівні $EO_i = 1$ на керуючих входах змінювати почергово, то на шині *D* вихідні сигнали пристроїв будуть відбиватися також почергово.

Керуючі сигнали EI_1 … EI_N визначають режими приймання чи передавання. Завдяки цьому до шини *D* можна підключати не тільки виходи, а й входи робочих блоків з метою утворювання *двонаправленого* режиму роботи.

Якщо на деякий час подати $EO_i = 0$, а $EI_i = 1$, то робочий блок буде приймати сигнали з шини *D*. Якщо на один з пристроїв подавати сигнал $EO_i = 1$, а на інший $EI_j = 1$, то сигнал від *i*-го пристрою можна передавати на *j*-й пристрій, тобто виконувати за необхідності обмін даними між двома робочими блоками.

Одержану числову магістраль *D* на базі елементів з відкритим колектором називають *двонаправленою* шиною. Таке з'єднання елементів дозволяє зменшити кількість виводів пристроїв і кількість провідників магістралі.

Коли усі пристрої відключені від шини, тобто шина знаходиться у стані інформаційного нуля, то на ній утримується високий рівень напруги, завдяки джерелу *+E* та резистору $R_н$. Тоді обмін по шині з відкритим колектором здійснюється за правилами негативної логіки, коли активний рівень низький. Це забезпечує меншу споживану потужність у проміжках між операціями обміну.

Суттєвим недоліком таких шин є мала швидкодія та низька завадостійкість. Це пояснюється тим, що шини передавання інформації, як правило, довгі, мають значні паразитні ємності і тому, як наслідок, вони чутливі до завад. Крім того, активний вихід (навіть високоомний) незручний і досить шкідливий в тих випадках, коли потрібно вести обмін даними одночасно з кількома робочими блоками або вузлами, як це має місце у мікропроцесорній техніці.

Сумісну роботу декількох блоків на одній лінії інформаційної шини успішно забезпечує логічний елемент з трьома вихідними станами або тристановий драйвер, який був розроблений спеціально для використання в ролі вихідного буфера для підключення цифрових блоків до магістралі або шини. Буфери з трьома станами називають шинними драйверами.

Адресна шина – це група сигнальних ліній, за допомогою яких визначається прилад, в який чи з якого повинна бути передана або зчитана інформація. Кожне двійкове число, що подане на адресну шину, може визначити лише конкретну область пам'яті. Восьмирозрядні процесори мають 16 адресних ліній; 16-розрядні мають, як правило, 20-розрядні адресні шини. Адресні шини, звичайно, однонаправлені і позначаються символами A_i.

Шина керуючих сигналів – це набір сигнальних ліній. Склад таких сигналів може бути досить різноманітним в залежності від типу запам'ятовувача.

Нормальне функціонування ОЗП залежить від організації часових співвідношень між сигналами. Тому всі згадані сигнали мають бути розподіленими за часом.

Розглянемо процеси запису й зчитування інформації в мікросхемі, що наведена на рис. 1.8,*а*. Ці процеси ілюстровані часовими діаграмами роботи на рис. 1.10,*а* та 1.10,*б*.

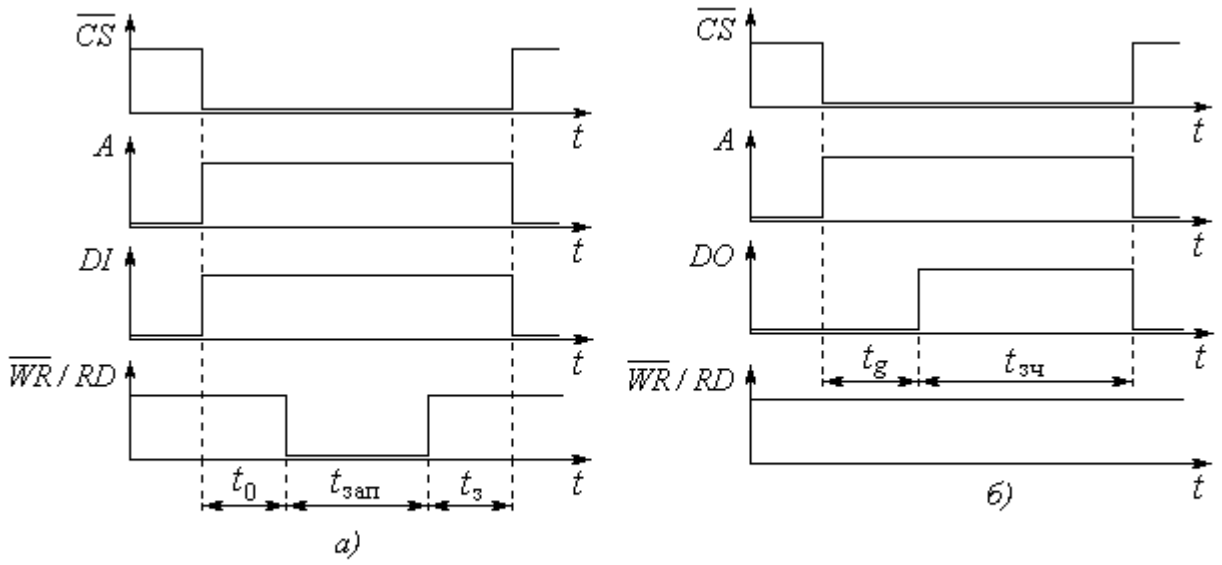


Рис. 1.10. Часові діаграми роботи ВІС ОЗП: а) у режимі запису; б) у режимі зчитування

Мікросхема має наступні шини керуючих сигналів: $\overline{CS}$ – вибір мікросхеми; $\overline{WR}/RD$ – керування режимами запису та зчитування.

Послідовність подачі сигналів у режимах запису та зчитування наведена на рис. 1.10, яку обов'язково треба виконувати для правильної роботи мікросхеми.

У режимах запису й зчитування тривалість імпульсу $\overline{CS}$ визначає час звернення до даної мікросхеми, за який потрібно закінчити процеси звернення.

У режимі запису інформації (рис. 1.10,*а*) дозвіл на запис $\overline{WR}/RD = 0$ подається (за часом) після установки адреси. У нашому випадку адреса задається сигналами одиничного рівня, тобто $A = 1$. Часова затримка приходу сигналу $\overline{WR}/RD$ за порівнянням з $A = 1$ на тривалість t_0 запобігає помилок, які пов'язані із забороною починати запис інформації до закінчення часу

дешифрування адреси. Для забезпечення надійного запису у вибрану комірку ОЗП сигнал дозволу на запис $\overline{WR}/RD$ знімається раніше за всі інші сигнали *DI*, *A* і $\overline{CS}$. Отже останні мають припинити свою дію лише з певною затримкою $t_з$. Сума часових інтервалів називається часом циклу запису

$$T_з = t_0 + t_{зап} + t_з . \quad (1.1)$$

У режимі зчитування інформації (рис. 1.10,*б*) після установлення активного рівня адреси *A* = 1 дані на виході *DO* з'являються тільки після певного проміжку часу t_g, який називають часом доступу при зчитуванні.

Сума часів називається часом циклу зчитування

$$T_{зч} = t_g + t_{зч}. \quad (1.2)$$

ВІС ОЗП допускають нарощування місткості пам'яті збільшенням розрядності та числа запам'ятовувачів.

1.4. Постійні запам'ятовувальні пристрої ПЗП

1.4.1. Класифікація ПЗП

Постійні запам'ятовувальні пристрої (ПЗП) – це функціональні вузли, які мають лише два режими роботи: зберігання та зчитування. Запис нових даних у ПЗП можна здійснювати тільки до включення ПЗП в роботу. Цей процес носить назву "Програмування ПЗП" і здійснюється як заводом-виготовувачем, так і користувачем-програмувачем.

Існує три типи ПЗП, що поділяються за способом програмування.

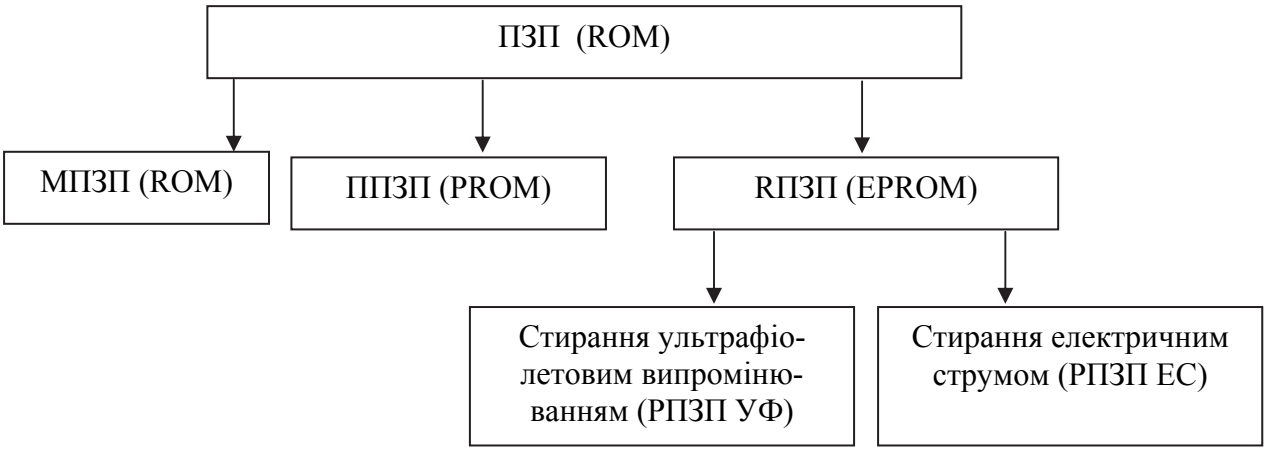


Рис. 1.11. Класифікація мікросхем пам'яті ПЗП

ПЗП з *масковим програмуванням* (МПЗП) – це пристрої, в які інформація записана раз і назавжди. Зміст пам'яті такого МПЗП залишається постійним на весь час його роботи. Маскові ПЗП виготовляються тільки на заводі, де за картою програмування, яка по суті є таблицею істинності, установлюються дільниці металізації, які потрібні для кодування тій чи іншої інформації. За такою технологією виготовляються ПЗП для перетворення, наприклад, двійкового коду в коди символів (російських, латинських літер, цифр тощо).

Програмовані ПЗП (ППЗП) на відміну від МПЗП мають напівпровідникові діоди або транзистори, які з‘єднані з усіма точками перетину шин матриці плавкими перемичками, тобто при виготовленні за всіма адресами ППЗП записується число {11 … 11}. Режим програмування складається з послідовної подачі адреси слів, після чого імпульсами струму руйнуються перемички в місцях, де вони непотрібні. Для перепалювання перемичок використовується спеціально призначений для цього пристрій – програматор.

Якщо при програмуванні ППЗП була допущена помилка, то виправити її вже не можна і ППЗП буде непрацездатним.

Перепрограмовані або *репрограмовані* ПЗП (РПЗП) дозволяють виконувати запис та стирання інформації. Організація РПЗП відрізняється від ППЗП тим, що між лініями рядків і стовпців включені не діоди чи біполярні транзистори з плавкими перемичками, а спеціальні МОН-транзистори з так званим плаваючим заслоном. Плаваючий заслін вбудований між металевим заслоном і підшарком МОН-транзистора. Він не має виводу і оточеній діелектриком. Під прискорювальним електричним полем він може накопичувати електрони, які відкривають транзистор на багато десятків років. Під полем протилежної полярності накопичення електронів руйнується, через що транзистор закивається. Так можна здійснювати перепрограмування РПЗП.

Стирання раніше записаних даних може виконуватися або ультрафіолетовим опромінюванням корпусу мікросхеми, або електричним сигналом.

Оскільки ПЗП існують тільки в інтегральному виконанні, то вони відомі під назвою великих інтегральних схем (ВІС).

ВІС (РПЗП) схожі на ОЗП, в яких цикл запису в декілька тисяч разів більше за цикл зчитування. В РПЗП можна повернути в початковий стан будь-який окремий МОН-транзистор.

Недоліком РПЗП та МПЗП порівняно з ППЗП є більший час вибірки: сотні секунд.

За способом зчитування ВІС ПЗП поділяють на асинхронні та синхронні.

Зчитування інформації з асинхронних ПЗП відбувається в будь-який час при зверненні до даного ПЗП, а із синхронних ПЗП – лише за наявності на спеціальному вході ВІС ПЗП синхроімпульсу заданої тривалості.

За технологією виготовлення ВІС ПЗП розрізняють за типом запам‘ятовувачів: діодні, біполярні та польові.

1.4.2. Структура ВІС ПЗП

Структурою ПЗП може бути дворівнева ПЛМ (див. рис. 1.12). Програмується лише матриця М2 (диз’юнкцій), а матриця М1 (кон’юнкцій) налагоджена на реалізацію функцій повного дешифратора всіх $q = 2^n$ вихідних від n вхідних адресних кодових комбінацій. Реалізація ПЗП зумовлює те, що схему повного дешифратора (матрицю М1) програмувати неможливо, через що параметр $q = 2^n$ зафіксований.

На рис. 1.12 наведена узагальнена структурна схема ВІС ПЗП.

Основою структурної схеми є накопичувач інформації (надалі – накопичувач), який з'єднаний через ША з дешифратором адреси *DC*, а через ШР – з блоком мультиплексорів *MUX*. Паралельний код адреси *A* подається на формувач адрес, протифазні сигнали якого надходять на *DC* і блок *MUX*, які активізують одну з горизонтальних (адресних) шин ША.

Зчитування записаної у запам'ятовувачах інформації відбувається по всіх вертикальних шинах ШР через блок мультиплексорів і пристрій вводу-виводу.

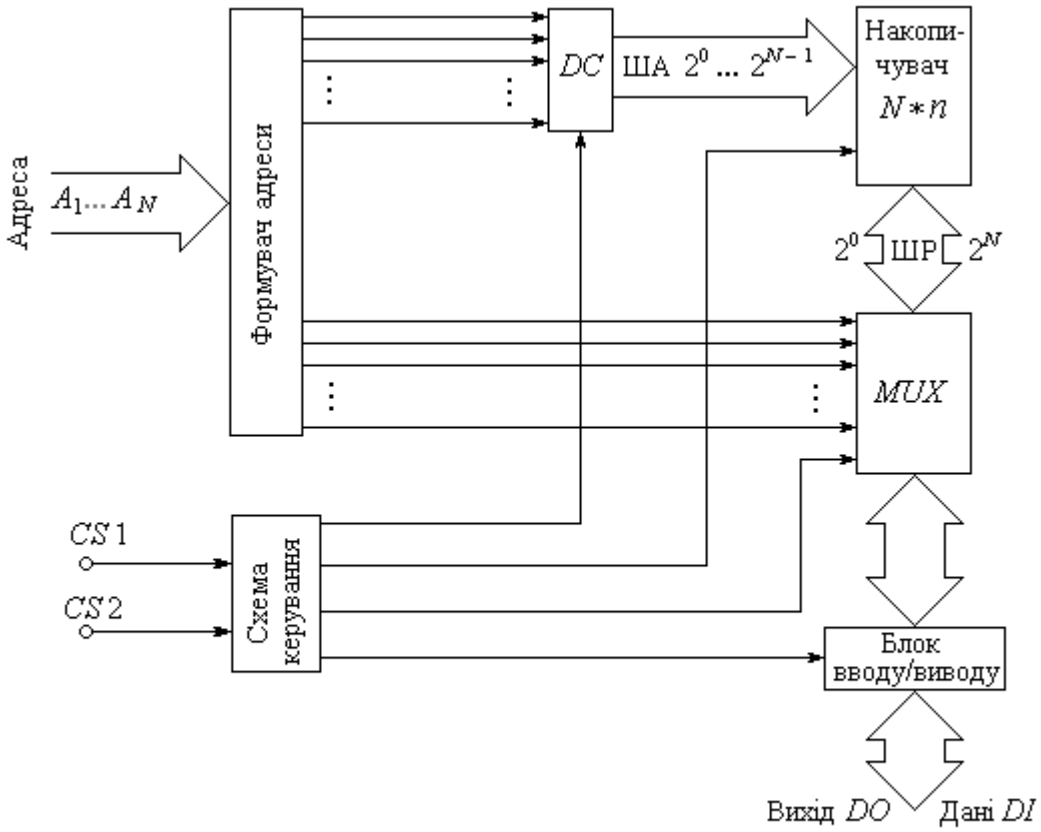


Рис. 1.12. Узагальнена структурна схема ВІС ПЗП

Схема керування синхронізує роботу *DC* і блока *MUX*. Пристрій вводу-виводу виконується або за схемами з відкритим колектором, або з використанням тристанових драйверів. Схема керування визначає режими запису, зчитування та стирання інформації. На відміну від ОЗП при зчитуванні накопичувача ПЗП видається вміст цілого рядка запам'ятовувачів, причому такий рядок може містити навіть декілька слів. З вибраного дешифратором рядка за допомогою мультиплексорів *MUX* виділяється і передається на вихід потрібне слово.

Щодо програмування, то воно починається зі складання таблиці істинності, де кожній комбінації повного набору адреси A_1, …, A_n відповідає кодова комбінація даних адреси D_0, …, $D_{(m-1)}$. Для наочного прикладу наведемо таблицю істинності для станів запам'ятовувача ПЗП (табл. 1.1).

Таблиця 1.1

Стани запам'ятовувача ПЗП

Слово ША	Дані ПЗП							
	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8
A_1	0	1	1	0	0	1	1	1
A_2	1	0	1	0	0	0	1	0
A_3	1	1	0	0	1	0	0	0
A_4	0	1	1	1	0	0	0	0

Таблиця істинності описує стани матриці накопичувача для побудови ПЗП, яка має місткість 32 біти, що розбиті на чотири слова (A_1 … A_4) по вісім розрядів (D_8 … D_1) у кожному з них. Такий ПЗП здатний запам'ятовувати чотири однобайтових слова (4 × 8 = 32 біти).

Діодна матриця накопичувача ПЗП, яка відповідає табл. 1.1, наведена на рис. 1.13.

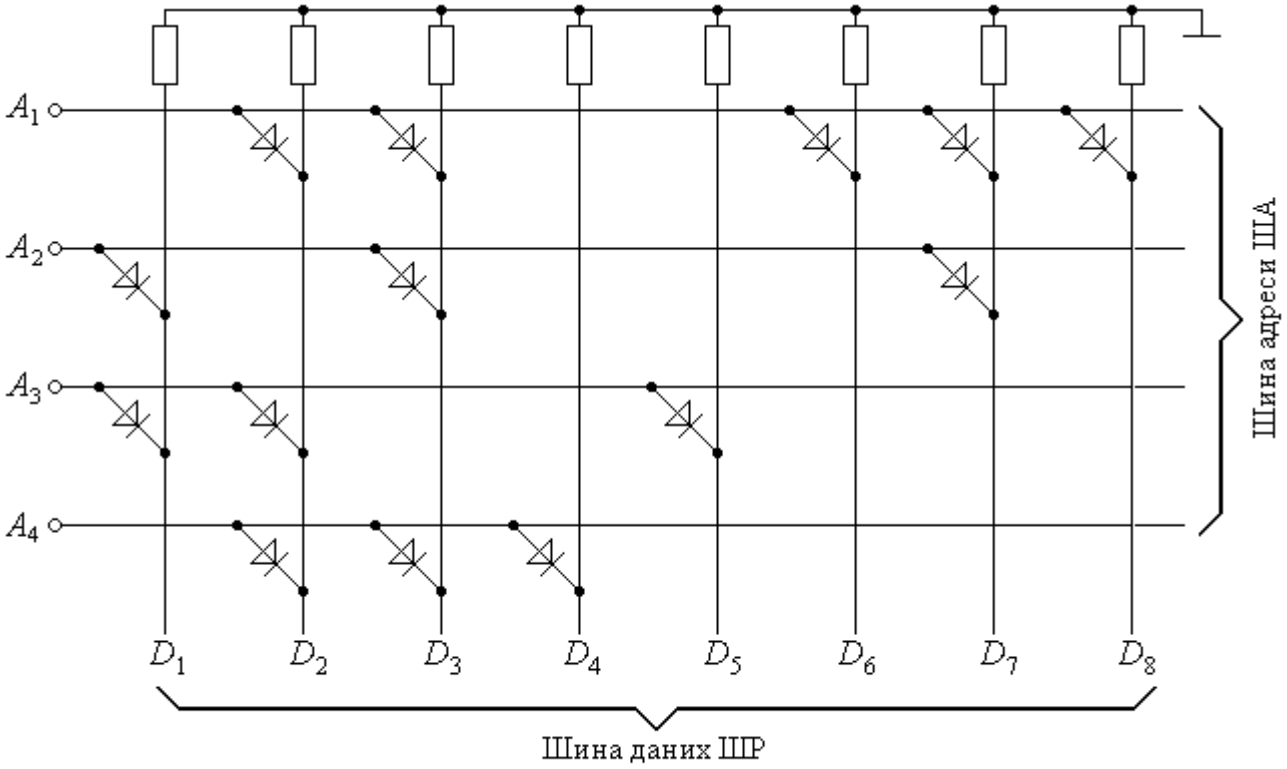


Рис. 1.13. Діодна матриця накопичувача ПЗП

Для вибору потрібного слова на одну з чотирьох адресних шин подається одиничний рівень напруги. Тоді на тих шинах даних ШР, на перетинах яких з обраною шиною адреси ША присутній діод, з'являється одиничний рівень, а якщо діод відсутній – нульовий. Наприклад, при активізації шини адреси A_1 кодова комбінація даних на шині ШР є словом: 11100110 (якщо вважати D_1 молодшим розрядом).

Основним недоліком ВІС ПЗП з діодним накопичувачем є відносно велике струмоспоживання по входах $A_1 – A_4$, що навантажує джерело сигналу.

Цей недолік усунений в ПЗП з транзисторним накопичувачем.

Схема біполярної матриці накопичувача, що відповідає табл. 11.1, наведена на рис. 1.14. Активним для матриці адресним потенціалом є рівень логічного нуля. В координатному полі накопичувача транзистори розміщують у точках перетину, де повинні зберігатися біти, що мають значення логічного нуля. При подачі на одну з адресних шин рівня логічного нуля, відкриваються транзистори, які підключені до цієї адресної шини. Тоді на тих шинах даних ШР, на перетинах яких з обраною ША присутній транзистор, формується нуль, а на решті ШР через резистори надходить рівень логічної одиниці від джерела $+E$.

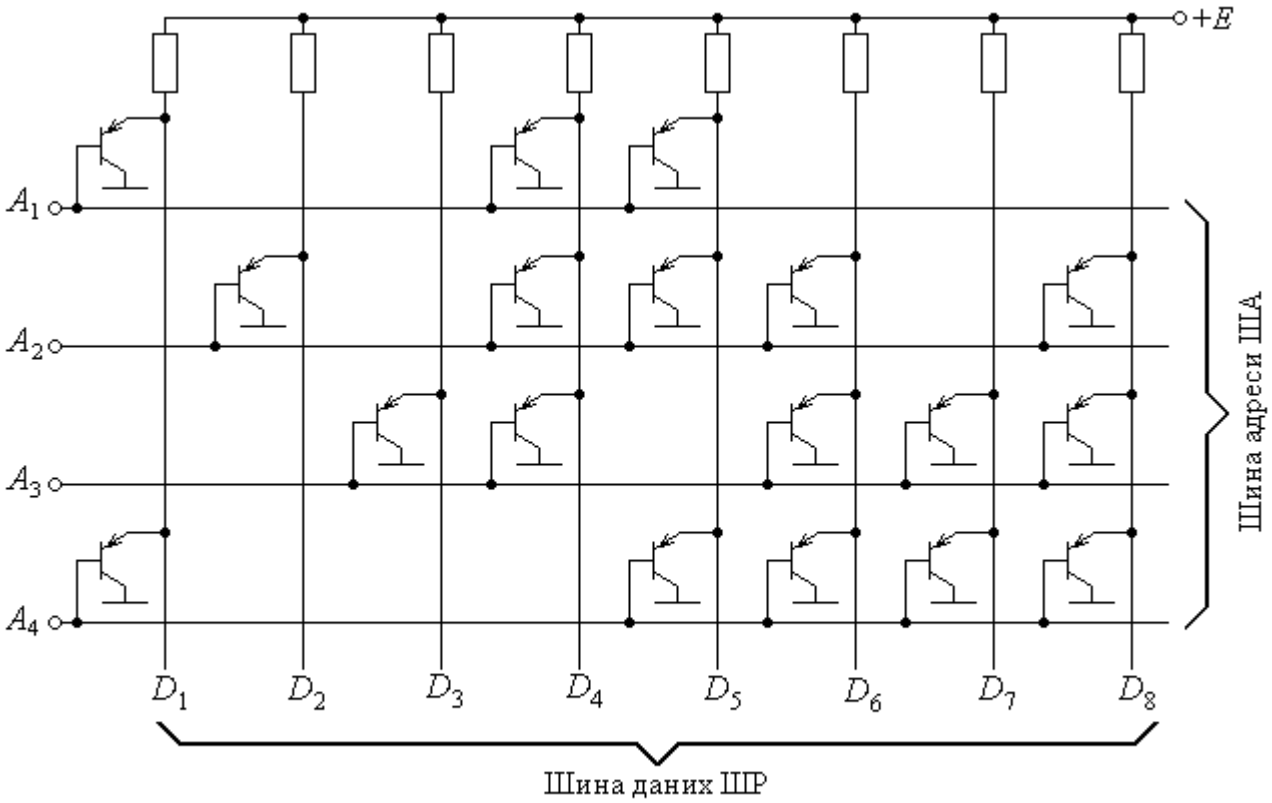


Рис. 1.14. Біполярна матриця накопичувача ПЗП

Щодо МОН-накопичувачів, то вони мають нижчу швидкодію, ніж біполярні, але потужність розсіювання їх значно нижча. З метою реалізації польової матриці-накопичувача на перетинах адресних шин і шин даних за відповідною таблицею істинності ПЗП включаються польові транзистори. Так само, як і у біполярного накопичувача, наявність або відсутність транзистора у точці ортогональних ліній відповідає стану логічної одиниці, або логічного нуля запам'ятовувача.

Структура програмованих ПЗП (ППЗП) аналогічна структурі маскових ПЗП. Різниця, по-перше, в схемі запам'ятовувача, а по-друге, в кількості напівпровідникових елементів, які перепалюються при програмуванні.

Запам'ятовувачі таких ППЗП бувають різні. Найчастіше зустрічаються запам'ятовувачі, що виготовляються з плавкими перемичками (рис. 1.15,*а*), або зі зворотно включеними діодами (рис. 1.15,*б*).

У процесі програмування перемички перепалюються електричним струмом. Якщо перемичка перепалюється, запам'ятовувач від'єднується від точки перетину.

Другий тип запам'ятовувача, що використовується у програмованих ПЗП, має зворотно включені діоди *VD* (рис. 1.15,*б*). За такої схеми у початковому стані до програмування ПЗП жодна точка перетину не має запам'ятовувачів.

У процесі програмування зворотно включені діоди *VD1* (*VD*) закорочуються, тобто залишається або один діод *VD2*, або транзистор *VT*, що відповідає присутності запам'ятовувача у точці перетину.

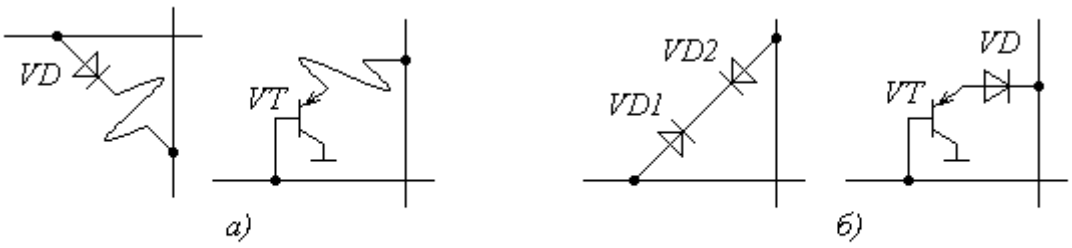


Рис. 1.15. Типи запамятовувачів програмованих ПЗП:
а) з плавкими перемичками; б) зі зворотно включеними діодами

Програмування ПЗП із запам'ятовувачами першого типу виконується таким чином, щоб перепалити перемички в тих точках перетину шин, де за таблицею істинності запам'ятовувач повинен бути відсутнім.

Програмування ПЗП із запам'ятовувачами другого типу, навпаки, зводиться до закорочування зворотних діодів у точках, де треба поставити запам'ятовувач. Режими програмування в різних мікросхемах програмованих ПЗП різні, бо визначаються не тільки технологією виготовлення даної ВІС, але й типом мікросхеми ПЗП.

Структура репрограмованих РПЗП аналогічна масковим та програмованим ПЗП, але відрізняється тим, що в них можна багаторазово стирати записану інформацію та записувати нову. Накопичувачі репрограмованих РПЗП виконуються за спеціальними технологіями з використанням спеціальних типів транзисторних структур.

Наприклад, на основі МОН-структур будують, так звані транзистори з плаваючим ізольованим заслоном. Ці транзистори змінюють під час програмування свої характеристики. Такі зміни є ознакою стану інформації, що зберігається в ПЗП. У процесі стирання всі транзистори під дією, наприклад, ультрафіолетового випромінювання закриваються, що відповідає запису в усіх запам'ятовувачах логічної одиниці. Потім за допомогою шин адреси і даних вибираються ті транзистори, в які потрібно занести логічний нуль, і відбувається запис нової інформації.

На рис. 1.16 наведені умовні позначення трьох типів ПЗП: маскові МПЗП (рис. 1.16,*а*), програмовані ППЗП (рис. 1.16,*б*) та репрограмовані РПЗП (рис. 1.16,*в*).

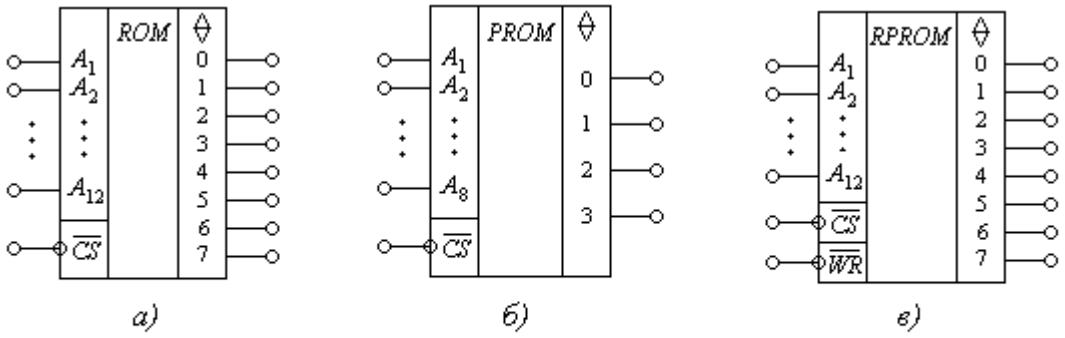


Рис. 1.16. Умовне позначення ПЗП: *а*) маскові (МПЗП); *б*) програмовані (ППЗП); *в*) ре програмовані (РПЗП)

Маскові ПЗП більш придатні для зберігання констант, стандартних таблиць, символів, підпрограм і нагадують “книжку для читання”.

Програмовані та репрограмовані ПЗП дозволяють ще реалізувати широкий клас функціональних вузлів і нагадують “блокнот з чистими сторінками”. ПЗП можуть бути використані для реалізації бульових функцій, побудови цифрових автоматів, перетворювачів кодів, арифметичних пристроїв та пристроїв оброблення інформації.

У ряді випадків застосування ВІС ПЗП можна одержати значний виграш у швидкодії, габаритних розмірах та вартості проектованих пристроїв.

1.5. Стекова пам'ять

Як відомо, за допомогою команд переходів можна відходити від строго послідовного порядку виконання команд. Особливий тип команди переходу, що трохи відрізняється від розглянутих вище, дозволяє багаторазово використовувати деяку спеціально виділену групу команд, звану *підпрограмою*. У процесі виконуваних обчислень у головній програмі може знадобитися виконати деяке «підрахування», оформлене у вигляді підпрограми. Комп'ютер повинен передати управління з головної програми у підпрограму, виконати відповідне «підрахування» і потім здійснити повернення до головної програми для продовження роботи. Щоб здійснити таке повернення, необхідно зберегти значення програмного лічильника, яке було до переходу до підпрограми. Це і робиться в команді *переходу на підпрограму*, а саме запам'ятовується вміст програмного лічильника і потім передається керування. Для того, щоб відновити програмний лічильник, використовуючи запам’ятоване значення, і забезпечити повернення до головної програми, вживається спеціальна команда *повернення з підпрограми*. Місце, де запам'ятовується вміст програмного лічильника, зазвичай являє собою *стек*.

У загальному випадку стек – це сукупність регістрів, які приймають і видають інформацію відповідно до правила *останнім увійшов, першим вийшов* (LIFO – last-in first-out). Це означає, що тільки вершина стека, де знаходиться

останній його елемент, безпосередньо доступна ззовні. Залежно від конструкції комп'ютера стек може перебувати у пристрої управління або бути частиною пам'яті.

Якщо стек може зберігати кілька адрес (значень програмного лічильника), ми отримуємо можливість виконувати вкладені підпрограми. Таким чином, з підпрограми можна зробити перехід на іншу підпрограму або навіть на саму себе – це називається *рекурсією*. Очевидно, що глибина вкладень підпрограм залежить від ємності стека.

Стекова пам'ять утворює одномірниій масив. Як правило, вона може бути організована у вигляді паралельних або зсувних регістрів. Кількість розрядів у регістрах визначає розрядність слова, яке модна записати у запам’ятовувальний пристрій.

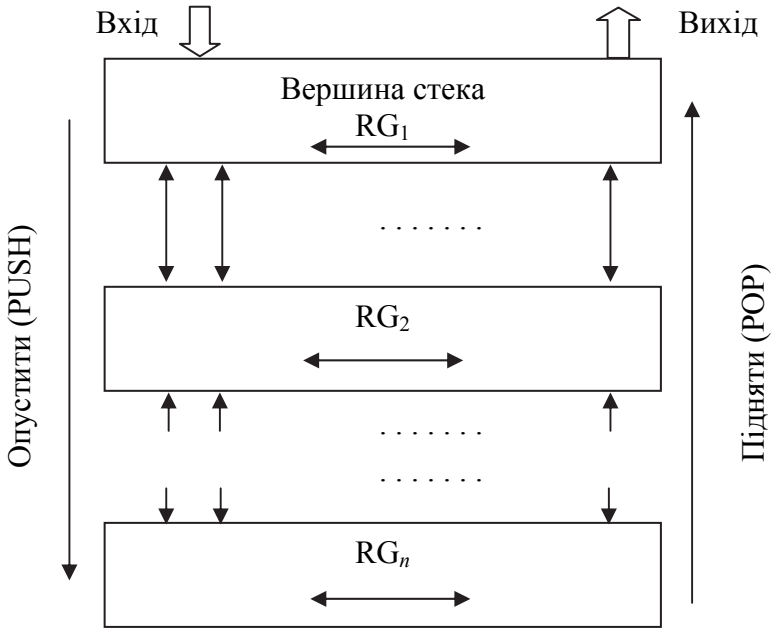


Рис. 1.17. Схема роботи стека

Доступ здійснюється завжди до верхньої комірки – вершини стека. Коли слово записує у вершину стека, слово, яке займало її, і усе слова, що йдуть нижче, зсуваються вниз на одну клітинку, а вміст нижньої комірки втрачається. При цьому кажуть, що стек опускається. Існує зворотна операція, за якої стек піднімається. Коли стек піднімається, то слово на вершині стека видаляється і його місце займає і тим самим стає доступним слово з другої зверху комірки.

З точки зору внутрішньої структури стек можна уявити групою працюючих «в унісон» регістрів зі зсувом в обох напрямах.

У багатьох додатках застосування стека виявляється дуже зручним та природним. Для таких додатків характерне використання пам'яті з вибіркою не за довільними адресами слів, а на основі взаємної їх впорядкованості. Наприклад, замість необхідних у процесі перерахування операнди можна було б завантажити у стек у порядку, зворотному їх використання. Підйом слів з стека буде тоді поставляти операнди у потрібному порядку.

Стек грає важливу роль у комп'ютерах як засіб збереження адрес повернення і стану даних для підпрограм. Його використання призводить до суттєвого спрощення, коли одна підпрограма викликає іншу, яка, своєю чергою, може викликати третю, і т. д. У таких випадках при кожному виклику адреса повернення з поточної підпрограми та інша необхідна інформація завантажується у стек. При поверненнях інформація у потрібному порядку вибирається з стека.

У багатьох мікропроцесорах пам'ять зі стековою організацією реалізується не за допомогою зсувних регістрів, а моделюється на пам'яті з довільним доступом. При цьому в якості стека зазвичай використовується просто частина оперативної пам'яті. Це дає додаткову гнучкість, оскільки ємність стека може змінюватися за необхідності, і максимальна ємність стека виявляється обмеженою лише розміром оперативної пам'яті. Крім того, перенесення стека у пам'ять дає економію апаратури.

Для моделювання стека у пам'яті з довільним доступом використовується підсумовувальний/віднімальний лічильник. Цей лічильник називається *покажчиком стека*. Передбачається, що на лічильнику увесь час знаходиться адреса комірки пам'яті, яка відповідає вершині стека. Спочатку на лічильник – покажчик стека – заноситься деяка початкова адреса. Коли слово потрібно помістити у стек, воно записується за адресою, яка записана в покажчику стека, і покажчик зменшується на 1. Коли слово беруть зі стека, покажчик збільшується на 1, і потім за отриманою адресою читається слово. Таким чином, стек розширюється при заповненні і стискається при вибірці. Покажчик стека містить адресу наступної доступної для запису комірки. Варіації описаної схеми роботи, які зустрічаються на практиці, в основному стосуються того, як просувається покажчик стека.

При роботі з модельованим стеком покажчик стека входить до складу мікропроцесора.

Контрольні питання

1.1. Що таке ОЗП?

1.2. Що таке запам’ятовувач?

1.3. Які вузли входять до типової структурної схеми ОЗП і які функції виконує кожен з них?

1.4. Як відбуваються запис та зчитування у накопичувач ОЗП?

1.5. Які часові співвідношення між вхідними сигналами ВІС ОЗП?

1.6. Що таке ПЗП? Які функції вони виконують?

1.7. Яка існує класифікація ПЗП за способом програмування?

1.8. Які принципи побудови ВІС ПЗП?

1.9. Які типи запам’ятовувачів використовуються в ВІС ПЗП?

1.10. Яка різниця між МЗП, ППЗП та РПЗП?

1.11. Що таке стек?

1.12. Які елементи цифрової схемотехніки най часті використовують при організації стекової пам’яті?

1.13. Що таке моделюючий стек?

1.14. Розрядність якого елемента у стековій пам’яті визначає розрядність слова, яке можна записати до ЗУ?

2. МІКРОПРОЦЕСОРИ

2.1. Вступ

У 1951 році було створено перший промисловий комп'ютер (Univac 1), який мав назву ЕОМ (електронна обчислювальна машина).

ЕОМ – комплекс технічних засобів, призначених для автоматизованого оброблення інформації процесі розв'язання обчислювальних та інформаційних задач. ОЕМ істотно вплинули на наше суспільство та спосіб життя. Виникла нова галузь промисловості. Терміни «цифрові обчислення», «логічне проектування», «програмування» стали науковими та інженерними поняттями. Проте, широта цих понять призводила до розходження інтересів. Наприклад, інтереси однієї групи фахівців у використанні ЕОМ та програмуванні (область програмного забезпечення). Інша група спеціалізувалася на створенні ЕОМ (область створення апаратури). Такий розподіл був виправданим лише у відношенні великих обчислювальних машин.

З 1965 року, з моменту появи мікросхем та мікроконтролерів, прикладні програмісти та конструктори машин стали тісно переплітатися та представляти одну цілісну команду.

Мінікомп'ютери вже не були однозначно призначені для оброблення даних та вирішення задач; нерідко вони входили як складові частини у системи, які потребували швидкого прийняття рішення, – *системи реального часу.*

З появою у 1971 році мікропроцесорів розходження інтересів ще більше зменшилося. Розпочалась ера програмованої логіки. У цю еру поняття програмування та принципи проектування логічних схем зблизилися настільки, що їх взаємопроникнення вимагало і від вчених, і від інженерів повного розуміння як принципів програмування, так і апаратури. Лише за цієї умови можна було успішно використовувати усі закладені у мікропроцесори можливості.

2.2. Основні положення

Одним із досягнень технології напівпровідників полягало у створенні великих (ВІС) та надвеликих (НВІС) інтегральних схем, де в одному тілі кристалу напівпровідника розміщували 10^4 або більше транзисторів. Поява елементів з високим ступенем інтеграції призвела до розробки мікропроцесорів.

Мікропроцесор – це програмований логічний пристрій, виготовлене за ВІС-технологією. У конструкцію мікропроцесора закладено велику гнучкість. Мікропроцесори бувають кількох типів.

Універсальні мікропроцесори призначені для застосування в обчислювальних системах та клерувальних персональних комп'ютерах, робочих станціях, серверах, у суперкомп'ютерах. Основною характеристикою універсальних мікропроцесорів є наявність розвинених пристроїв для ефективної реалізації операцій з плаваючою точкою над 64-розрядними і більш довгими операндами та можливість відімкнення розвиненої системи

введення-виведення інформації, яка забезпечує різноманітні види зовнішніх пристроїв. Вони призначені, в основному, для розв'язанні науково-технічних, економічних, математичних, інформаційних та інших задач, які характеризуються складністю алгоритмів та великим обсягом даних, що обробляється.

Процесори цифрового оброблення сигналів (сигнальні процесори) – розраховані на оброблення у реальному часі цифрових потоків. Сучасні сигнальні процесори здатні виконувати цілочислові операції та операції з плаваючою точкою 32-40-розрядними операндами.

Сигнальні процесори апаратно підтримують фільтрацію та згортання сигналів, обчислення кореляційної функції двох сигналів, пряме та обернене Фур'є-перетворення сигналу тощо.

Медійні та мультимедійні мікропроцесори забезпечують апаратну підтримку оброблення аудіо сигналів, графічної інформації, відео зображень тощо. Вони застосовуються у мультимедіакомп'ютерах, приставках для ігор, побутовій техніці тощо.

Сам по собі мікропроцесор не може вирішити ту або іншу конкретну задачу. Щоб розв'язати задачу, його потрібно запрограмувати та з'єднати з іншими пристроями. До їх числа зазвичай входять пам'ять та пристрої введення/виведення. Взагалі кажучи, деяка сукупність з'єднаних один з одним системних пристроїв, включаючи мікропроцесор, пам'ять та пристрої введення/виведення, яка націлена на виконання деякої чітко визначеної функції, називається *мікрокомп'ютером* або *мікропроцесорною системою.*

Хоча мікрокомп'ютери володіють усіма властивостями звичайних ЕОМ, їхня особливість полягає у відносно низькій вартості та малих розмірах.

Мікропроцесорні системи діляться на *керувальні, обчислювальні, контрольно-вимірювальні, збирання даних.*

Кожного дня відкриваються нові області застосування для мікропроцесорів. Нині вони використовуються у контрольно-вимірювальних пристроях, у касових апаратах магазинів, в інтелектуальних терміналах та кишенькових калькуляторах. Вони виявляються життєво важливим елементом в управлінні станками, хімічними процесами, периферійними пристроями великих комп'ютерів, в управлінні вуличним рухом, складними побутовими електроприладами та автомобілями. Кінець-кінцем, в області розваг та дозвілля завдяки цим пристроям виникло нове хобі – будувати свої власні комп'ютери.

Комп'ютер – це також закінчена мікропроцесорна система. Найбільш широке поняття має мікропроцесорний комплект.

Мікропроцесорний комплект – це сукупність інтегральних мікросхем різного ступеня інтеграції: ВІС мікропроцесорів, ВІС пристроїв пам'яті, ВІС пристроїв введення/виведення, ВІС контролерів зовнішніх пристроїв, службових інтегральних схем – тактовий генератор, модулі, з яких складається інтерфейс з пам'яттю, контролери шин, арбітри шин тощо, які є сумісні за своїми електричними, інформаційними та конструктивними параметрами. Мікропроцесорні комплекти призначені, в основному, для побудови мікропроцесорних систем.

У засобах автоматизації знаходять широкого застосування мікроконтролери.

Мікроконтролер є інтегрована на одному кристалі ВІС мікропроцесорна система, яка складається з одного або кількох мікропроцесорів, іноді з різною архітектурою та призначенням, підсистеми пам'яті та набору периферійних пристроїв. Мікроконтролер, у принципі, працює у пристроях – контролерах.

Контролером називається пристрій, часто вбудований, який призначений для керування технологічним пристроєм або процесом. Контролер будується на основі одного або кількох процесорів або мікроконтролера.

Сьогодні неможливо уявити сферу діяльності людини, де б не використовувалися мікропроцесори та мікроконтролери.

Більшість сфер людського життя при використанні мікропроцесорної техніки змінилися у кращий бік. Переваги мікропроцесорних пристроїв (малі габарити, економічність енергоспоживання, велика швидкодія) дозволили прискорити наукові дослідження, автоматизувати виробничі процеси, збільшити продуктивність праці. Для мікропроцесорних систем характерні висока гібридність та можливість перенастроювання для розв'язання поточних завдань при застосуванні алгоритму роботи. Універсальність мікропроцесорних систем і порівняна невелика вартість забезпечує їх доступність широкому загалу користувачів.

На основі мікропроцесорів (МП) створюються сучасні комп'ютери та складні радіотехнічні системи, пристрої зв'язку та «розумна» побутова техніка. Сучасний комп'ютер може обробляти числову, текстову, графічну та звукову інформацію. Інформація для оброблення повинна бути подана у вигляді, що сприймається комп'ютером. Структурна схема перетворення інформації в мікропроцесорній системі зображена на рис. 2.1.

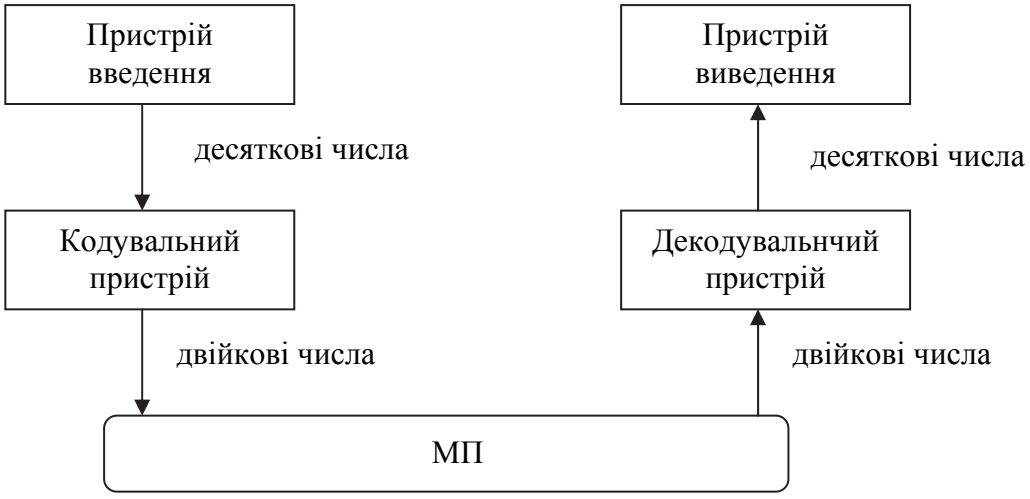


Рис. 2.1. Структурна схема перетворення інформації в процесорній системі

Інформація вводиться, звичайно, з низькочастотних датчиків та пристроїв керування, з клавіатури комп'ютера, що різко знижує загальну швидкодію системи.

Комп’ютер обробляє інформацію, подану у вигляді двійкових чисел. Інформація може вводитися в комп’ютер у більшості випадків з клавіатури датчиків і т. ін. у десятковому коді. Тому інформація від джерела має бути перетворена у двійковий код. В якості такого перетворювача може бути використаний шифратор – перетворювач десяткового коду у двійковий.

Процесор – електронний блок (або інтегральна схема – мікропроцесор), який виконує машинні коди програми, є головною частиною апаратного забезпечення комп’ютера або програмованого логічного контролера.

Після оброблення інформація з процесора надходить на перетворювач двійкового коду у десятковий. Таким перетворювачем слугує дешифратор. Далі інформація надходить на пристрій виведення, наприклад, на екран монітора, на принтер, апарат передачі даних і т.ін.

2.3. Структура мікропроцесорної системи

Принцип дії мікропроцесорної системи аналогічний дії будь-якої електронної обчислювальної машини (ЕОМ), яка виконує обчислення на основі алгоритмів. Алгоритм – це програма роботи МП, яка задає обчислювальний процес, що починається з довільних вихідних даних та спрямований на отримання повністю визначеного цими вихідними даними результату.

Кожний алгоритм характеризують:

- сукупність можливих вихідних даних;
- сукупність можливих проміжних та кінцевих результатів;
- правилами початку, перероблення, закінчення та виведення результату.

Важливими властивостями алгоритмів, які найбільш суттєво впливають на організацію МС, є:

- дискретність інформації, якою оперують алгоритми;
- кінцевість та елементарність набору операцій, які виконуються при реалізації алгоритмів;
- детермінованість обчислювальних процесів, що породжуються алгоритмами.

Елемент даних, які беруть участь в операціях як деякі величини, називають *операндом*.

На МС можна автоматично розв’язати будь-яку задачу, якщо вона надана у вигляді певних дій над операндами, що описуються системою формальних правил та умов, називаних *алгоритмами розв’язання задач*.

Алгоритм, поданий сукупністю команд, називається *програмою*. Фактично – це запис машинною мовою послідовності виконання операцій (команд), необхідних для розв’язання задачі. Мова, якою людина формулює послідовність операцій для розв’язання задачі, а МС сприймає її, називається *машинною мовою*.

Вперше спосіб реалізації програмного керування був запропонований Дж. фон Нейманом у 1945 р. Цей спосіб називається нейманівським принципом

програмного керування і використовується нині як основний принцип побудови МС.

Для цього принципа характерне наступне:

– різнотипні слова інформації різняться за способом використання, але не способами кодування;
– слова інформації розміщуються у комірках пам’яті, які нумеруються. Номера комірок називаються *адресою слова*.

Найпростіша нумерація комірок – лінійна. Це нумерація послідовними номерами. Будь-які інші структури, якими оперує програміст, повинні перетворюватися лише з цієї структури.

Машинна мова надана словом у двійковому коді. Для усіх способів надання даних до МС характерна неподільність слів інформації: слово оброблюється повністю як самостійний об’єкт – машинний елемент інформації.

Керуючі слова – команди – також являють собою двійковий код, який складається з кількох частин (полів) – коду операції та адрес операндів. Код операції задає машинну операцію, наприклад, операцію множення. МС реалізує набір функцій, який повинен володіти властивістю повноти, тобто бути достатній для описання алгоритму.

Дані та команди записані до комірок пам’яті.

Запам'ятовування великих обсягів інформації відбувається у пам'яті, або, точніше, у пристрої, що запам'ятовує (ЗП). Цей функціональний блок комп'ютера поділяється на підблоки, звані регістрами, кожен з яких здатний зберігати одне машинне слово. Кожен такий регістр, або елемент пам'яті, має свою адресу. Адреса – це просто ціле число, яке однозначно ідентифікує осередок. Слово, яке зберігається в осередку, називають вмістом цього осередку.

Отже, як дані, так і програма зберігаються в пам'яті. Це важлива обставина призводить до двох основних концепцій проектування комп'ютерів. Перша полягає у тому, що комп'ютер має два окремих і чітко розрізнюваних види пам'яті. Програма знаходиться завжди в одній пам'яті, а дані – в іншій. Машини, спроектовані відповідно до концепції поділу пам'яті на два види, називають машинами *гарвардського* типу.

Відповідно до другої концепції відмінність між програмною пам'яттю і пам'яттю для даних не проводиться, і відповідні комп'ютери називають машинами *фон-Неймановского* або *прінстонського* типу. У них програма може розміщуватися у будь-якому місці загальної пам'яті, і завдання програміста – стежити за тим, щоб дані і програма оброблялися по-різному. Перевага другої концепції – у можливості трактувати програму як дані, що дозволяє комп'ютеру змінювати свої власні команди. Існують мікропроцесори, спроектовані відповідно як до першої, так і до другої концепції. У цій книзі ми будемо розглядати машини фон-Неймановського типу.

Завдяки низькій вартості мікропроцесори часто застосовуються для вирішення однієї конкретної задачі. Великі універсальні ЕОМ постійно перепрограмовуються і тому можуть вирішувати завдання широкого спектра. Мікропроцесорам, спеціалізованим для одного конкретного додатка, така

гнучкість не потрібна. Одного разу написана і налагоджена програма надалі зазвичай не змінюється. Тому такі мікрокомп'ютери нерідко мають два види пам'яті: пам'ять, з якою можливе лише зчитування (ROM – read-only memory), або постійна пам'ять, і пам'ять зі зчитуванням та записом (RWM – read/write memory).

Стало загальноприйнятим пам'ять зі зчитуванням і записом називати пам'яттю з довільною вибіркою (RAM – random-access memory), незважаючи па те, що і пам'ять лише зі зчитуванням також володіє довільністю вибірки. Термін «довільна вибірка», або «довільний доступ», відповідає тому факту, що звернення до будь-якого осередку виконується за один і той самий час.

Як російські еквіваленти скорочень RAM, RWM і ROM часто використовуються ЗПДВ – запам'ятовуючиий пристрій з довільною вибіркою, ОЗУ – оперативне запам'ятовуючий пристрій і ПЗУ – постійний запам'ятовуючий пристрій.

Змінити інформацію, одного разу записану у постійну пам'ять, складно, якщо взагалі можливо. Пам'ять цього типу завдяки своїй низькій вартості використовується для зберігання програм і постійних даних; змінювана інформація зберігається у пам'яті зі зчитуванням і записом.

Місця даних та команд у пам'яті МС позначаються за допомогою адрес, що являють собою номер комірки. Читання та запис інформації при використанні пам'яті виконується з обов'язковим передаванням до пам'яті адреси комірки. При читанні інформації з комірки її зміст не руйнується і може бути прочитаний багато разів. При запису інформації до комірки попередній стан змінюється на новий.

Така операція заміни значення називається *операцією присвоєння* та позначається як =.

У команді мають бути зазначені адреси операндів. Використання у команді адрес замість самих операндів надає програмі універсальність формули: одна й та сама програма буде працювати при запису до комірки різних даних. Кількість адрес у команді залежить від кількості її операндів.

Для бінарних операцій (додавання, віднімання) необхідно два операнди, а у команді треба мати три адреси: дві – для зазначення операндів; одна – для адреси комірки, куди повинен бути розміщений результат.

Всі ці дані розташовані в команді у певному (для кожної МС) порядку, що задається форматом команди, тобто схемою розташування двійкових цифр у команді, що дозволяє розрізняти її складові частини та визначати їх функції.

Розглянемо спрощену структуру МС, що призначена для оброблення даних або керування деяким процесом, яка показана на рис. 2.2.

Структура МС – це сукупність елементів і зв'язків між ними. Усі МС мають наступні функціональні блоки:

- центральний процесор (ЦП), що складається з арифметико-логічного пристрою (АЛП) та пристрою керування (ПК);
- пам'ять (внутрішню та зовнішню);
- пристрій введення та пристрій виведення інформації.

Об’єднання функціональних вузлів та блоків МС здійснюється за допомогою шин.

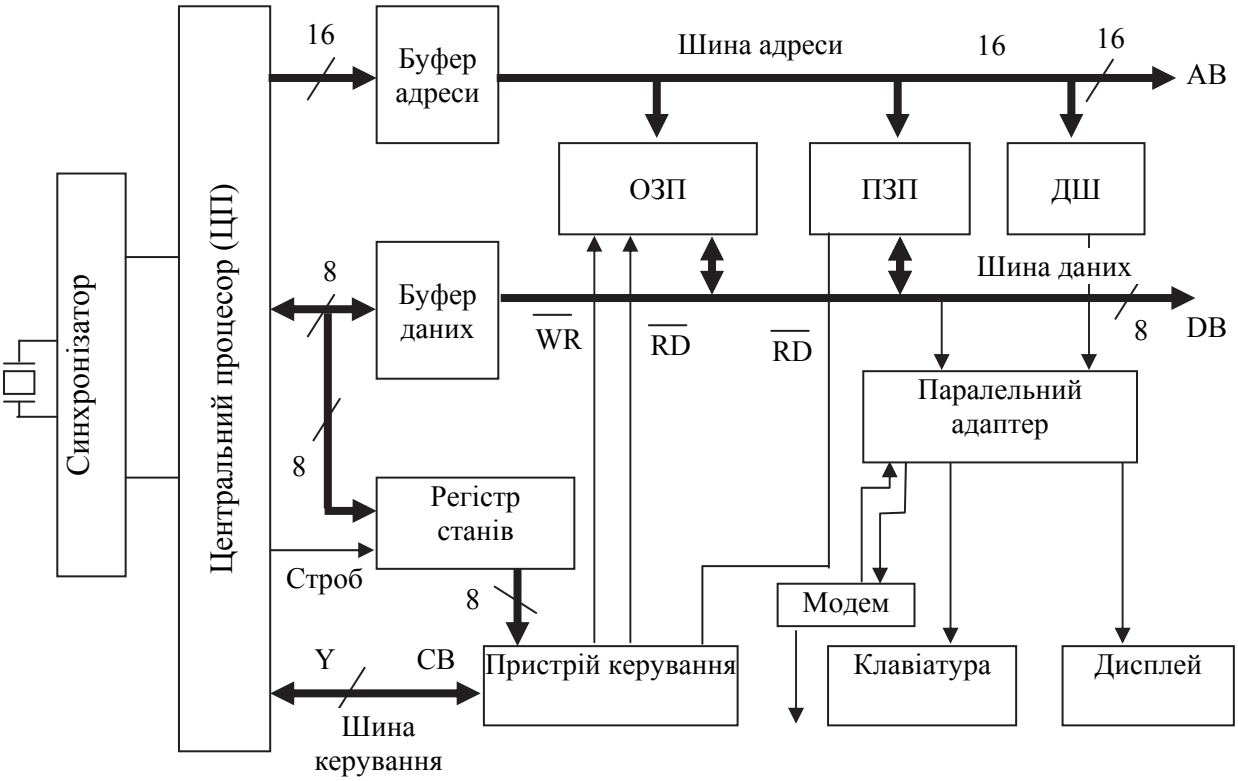


Рис. 2.2. Структура МС

Шина являє собою сукупність ліній, якими передається інформація від будь-якого з декількох джерел до будь-якого з декількох приймачів. Одна з поширених у комп'ютерах структур та шин представлена на рис. 2.3.

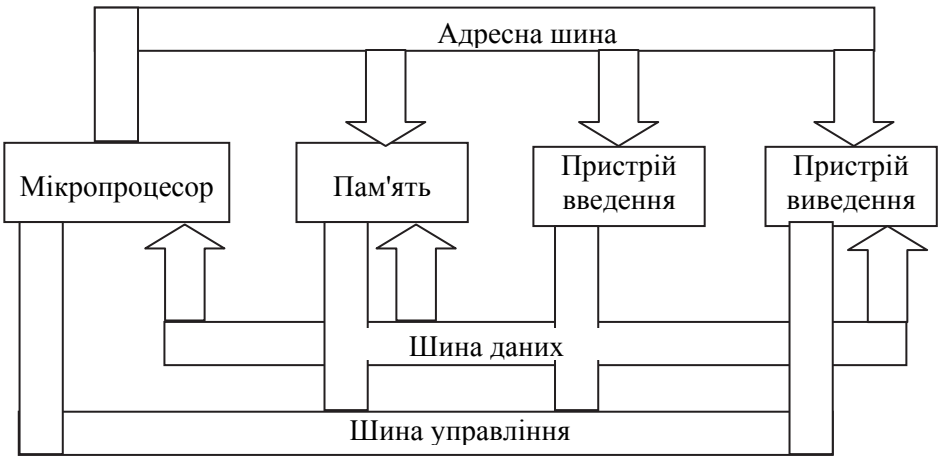


Рис. 2.3. Шини типового мікрокомпютера

На рис. 2.3 надані шини трьох типів. *Адресна шина* однонаправлена, тобто інформація по ній передається тільки в одному напрямі. Ця шина служить для передавання адреси від мікропроцесора до пам'яті, вхідного або вихідного пристрою. *Шина даних* двонаправлена, тобто інформація по ній може

передаватися в обох напрямах, і вона служить для передавання даних. Нарешті, *шина управління* складається з ліній, якими передаються тактові, що синхронізуючі сигнали, а також інформація про стан (статус) пристроїв. Частина ліній в керуючій шині однонаправлені, частина двонаправлені. Тому на рисунку спрямованість цієї шини ніяк не позначена.

Протоколи (порядок обміну даними, структура шин та швидкодія) при обміні можуть бути оптимальними.

У цій системі центральне місце займає процесор, який виконує арифметичні та логічні операції над даними, здійснює програмне керування пристроєм оброблення інформації, організує взаємодію усіх пристроїв, що входять до системи. Робота процесора здійснюється під дією сигналів схеми синхронізації й таймера, який дуже часто виконується у вигляді окремого кристалу.

Структурна (рис. 2.2) схема відображає магістрально-модульний принцип організації мікропроцесорних пристроїв і систем. Окремі блоки є функціонально закінченими модулями з вбудованими схемами керування, виконаними у вигляді одного або кількох кристалів ВІС, укладених у корпуси з відповідною кількістю виводів.

Міжмодульний обмін інформацією здійснюється за допомогою колективних шин (магістралей), до яких мають доступ всі основні модулі системи. У будь-який момент часу можливий обмін лише між двома модулями системи. Таким чином, обмін інформацією здійснюється шляхом розподілу у часі модулями системи шин (магістралей). Це є магістральний принцип побудови поєднання модулів (інтерфейсу) мікропроцесорної системи МС. Цей принцип припускає наявність інформаційно-логічної сумісності модулів, яка реалізується шляхом використання єдиних способів надання інформації, алгоритму керування обміном, форматів команд керування обміном та способу синхронізації.

2.4. Узагальнена структура мікропроцесора (МП)

Процесор (ЦПП) – це функціональний блок обчислювального пристрою, призначений для реалізації оброблення цифрових даних та керування ходом цього оброблення. Його іноді називають *мікропроцесором* або просто *процесором*.

Головними характеристиками ЦПП є тактова частота, продуктивність, енергоспоживання, норми метографічного процесу, використовуваного при виробництві, та архітектура.

В узагальненому вигляді функціонування процесора може бути подано, як циклічне чергування двох етапів: 1) вибірка команд з пам’яті та їх дешифрування; 2) виконання команд.

Вибірка команд – це автоматичний процес, який здійснюється під впливом імпульсів від генератора тактових імпульсів (ГТІ). Механізм реалізації залежить лише від апаратної структури процесора.

При дешифруванні команд формується послідовність керуючих сигналів для усіх вузлів процесора та інших блоків на основі коду, що міститься у команді.

Дії, які виконуються у відповідності з командою, можуть являти собою арифметичне або логічне оброблення, пересилання даних, формування адреси наступної команди або зміну режиму роботи МП. Ці дії задає програміст у межах тих команд, які властиві даному МП.

Після виконання команди, процесор автоматично переходить до вибірки наступної команди з пам’яті. Сучасні мікропроцесори суттєво різняться набором функціональних блоків та зв’язками між ними. Тим не менш, у структурі будь-якого процесора можна виокремити основні елементи, що визначають специфіку процесора як клерувального центра обчислювача. Перш за все, мова йде про два блоки: пристрій керування та операційний пристрій. Узагальнена структура мікропроцесора показана на рис. 2.4.

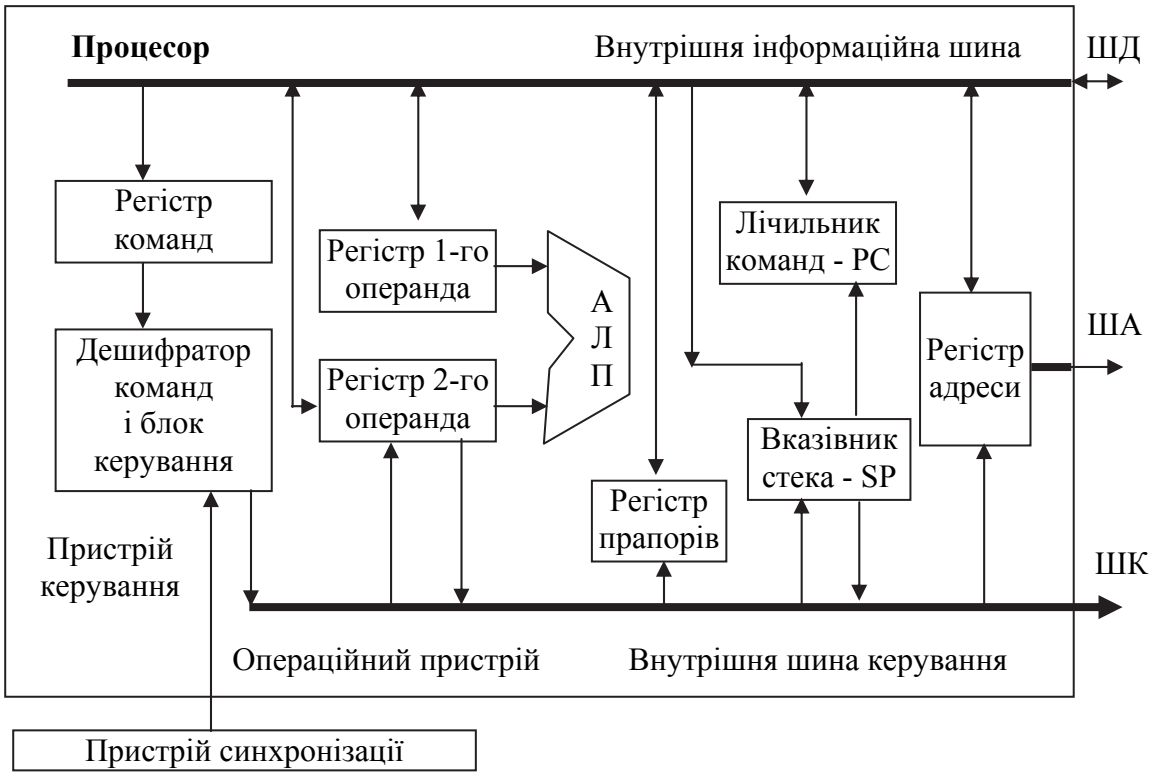


Рис. 2.4. Узагальнена структура мікропроцесора

Пристрій управління керує роботою комп'ютера. Він автоматично, послідовно по одній, отримує команди з пам'яті, декодує кожну з них і генерує необхідні для її виконання сигнали. Для того, щоб отримати команду з пам'яті, пристрій управління перш за все повинен знати її адресу. Зазвичай команди обираються з послідовних комірок пам'яті, і їх адреси вказуються *програмним лічильником*, що знаходяться у пристрої управління. Далі, щоб мати можливість декодувати і виконати поточну команду, її потрібно десь запам'ятати. Цій меті в пристрої управління слугує *регістр команди*.

Для того, щоб бути правильно інтерпретованою пристроєм управління, команда повинна мати певну структуру, яку називають *форматом команди.* У

мікропроцесорів різних типів формати команд різні. Однак є інформація, яка має бути присутня у команді у будь-якому випадку. Найбільш важливе значення має код операції і у деяких командах адреса. Код операції – це сукупність двійкових цифр, які однозначно визначають операцію, виконувану у процесі інтерпретації команди. Адресна частина команди (якщо вона є) вказує на осередки (наприклад, в пам'яті), до яких потрібно звернутися, виконуючи команду. Наприклад, якщо виконується операція додавання, адресна частина команди може вказувати на осередок, де знаходиться другий доданок.

Дуже важливо, щоб студент повністю розумів різні вживання слова «адреса» і розрізняв адресу комірки та її вміст. У командах певного формату існує адресна частина. Це числовий покажчик осередку, пов'язаної з операндом. Однак команда сама знаходиться у пам'яті і, отже, має пов'язану з нею адресу. Як правило, ця адреса не збігається з адресною частиною у самій команді.

Наступна функція пристрою управління – це синхронізація роботи окремих блоків комп'ютера. Вона здійснюється за допомогою *генератора тактових імпульсів*, або *тактового генератора*. Оброблення команди займає кілька періодів тактового генератора. Взагалі кажучи, команду потрібно вибрати з пам'яті, декодувати і потім виконувати. Вибірка, декодування і виконання розпадаються на кілька часових інтервалів. Кожен з цих інтервалів, що містить один або більшу кількість періодів тактового генератора, являє собою так званий *машинний цикл*. Сукупний час, необхідний для вибірки, декодування і виконання команди, утворює *командний цикл*, або *цикл виконання команди*.

Операційний пристрій (ОП) слугує для оброблення цифрової інформації (арифметичні або логічні дії, зміщення, аналіз чисел і т.ін.).

Основним елементом для зберігання інформації всередині процесора є регістри (РГ1, РГ2), які виконують функцію надоперативного ОЗП з мінімальним часом записування й зчитування.

Регістр команд (РгК) (англ. isnructionregister) використовується для фіксації кода команди після зчитування її з пам’яті. У цьому регістрі фіксується лише код операції (КОП) – частина коду команди, що визначає виконувану дію та спосіб адресації операндів.

Регістри операндів слугують для зберігання даних у процесі їх оброблення та дозволяють уникати постійних звертань до пам’яті. У сучасних процесорах кількість регістрів операндів може досягати 10…15. По суті, вони утворюють внутрішню оперативну пам'ять процесора. В однокристальних мікроконтролерах кількість регістрів операндів доведена до кількох десятків. Тому їх іноді називають *регістровими файлами*. Деякі регістри використовуються для зберігання або формування адрес інших операндів, тобто на їх основі реалізується механізм непрямої адресації.

Лічильник команд (англ. PC-programmingcounter) – регістр, в якому при вибірці або виконанні поточної команди формується адреса наступної команди. Модифікації змісту регістра PC – це засіб керування послідовністю вибірки команд з пам’яті і, отже, керування ходом обчислювального процесу.

Вказівник стека (англ. SP-stackpointer) – регістр, в якому при виконанні програми зберігається адреса межі тієї області пам’яті, для якої програміст використовує принцип послідовного доступу до даних (так званий протокол роботи зі стеком).

Регістр адреси – регістр, в якому формується адреса будь-якого пристрою, зовнішнього по відношенню до МП (комірка пам’яті або порту введення/виведення), перед звертанням до цього пристрою. Даний регістр необхідний, тому що джерелами адресної інформації можуть бути різні регістри процесора. Регістр адреси відіграє роль накопичувального буфера, з якого адресна інформація видається на зовнішню шину адреси.

Результат операції, що виконується в АЛП, заноситься до одного з регістрів або пересилається до пам’яті (залежно від команди). В регістрі ознак автоматично формуються ознаки, що характеризують цей результат.

Регістр ознак (англ. F-flags) – це елемент внутрішньої пам’яті, в якому у вигляді окремих біт фіксуються ознаки, що характеризують результат операції, виконуваної в АЛП (нульовий результат, переповнення стека і т.ін.).

Арифметично-логічний пристрій (АЛП) – функціональний блок МП, призначений для реалізації дій з оброблення даних.

2.5. Пристрої введення/виведення

Пристрої введення,виведення являють собою периферійні пристрої (МС) системи. Через ці пристрої здійснюється контакт МС із зовнішнім світом. Вони є буферними для перетворення інформації з тих мов і тих швидкостей, на яких працює мікропроцесорна система МС, до тих, які сприймає людина або інша система, пов’язана з даною. Пристрій введення отримує із зовнішнього світу дані та команди, які надходять у пам'ять. Пристрій виведення отримує обчислені результати і передає їх людині-оператору або іншій системі.

Як приклад пристрою виведення можна назвати друкарські пристрої, які під’єднані до МС.

Точки контакту між пристроями введення/виведення та мікропроцесором називають портами введення/виведення.

Порти введення/виведення мають свої адреси, тому до одного мікропроцесора може бути під’єднано кілька пристроїв введення/виведення.

Характерною особливістю МС є те, що вся інформація зберігається та обробляється у дискретному вигляді, тобто у вигляді кінцевих чисел. Нерідко виникає необхідність узгодження комп’ютера з іншої системою, яка не здатна обробляти дискретну інформацію. Недискретна інформація називається *аналоговою* або *безперервною*. У таких випадках доводиться здійснювати перетворення із цифрової форми у неперервну і навпаки. Пристрої введення/виведення, які здійснюють відповідні перетворення, називають *аналого-цифровими* та *цифро-аналоговими перетворювачами.*

Способів підімкнення портів введення/виведення кілька. Їх можна під’єднувати безпосередньо до арифметичного пристрою. Нерідко для досягнення більшої продуктивності системи бажано дозволити пристроям

введення/виведення звертатися до пам’яті безпосередньо, а не через арифметичний пристрій. У такому випадку кажуть *про прямий доступ до пам’яті* (DAM – direct memory access).

2.6. Функціонування мікропроцесорної системи або мікрокомп’ютера

У цьому розділі ми зупинимося на взаємодії блоків та на динаміці інформаційних потоків.

Обробка даних здійснюється головним чином в арифметичному пристрої. Ця обробка включає як арифметичні, так і логічні операції Вбудовані операції надзвичайно елементарні. Більш складні математичні дії повинні виконуватися за допомогою програм, що користуються вбудованими операціями.

Зазвичай головний регістр в арифметичному пристрої називається *акумулятором*. У ньому, як правило, знаходиться один з операндів перед виконанням операції, і в нього ж поміщається її результат. Арифметичний пристрій часто містить ще кілька допоміжних регістрів, званих робочими; вони спрощують складання програми.

Арифметичний пристрій містить також ознакові біти, або *прапорці*. Ці біти містять інформацію, що характеризує стан мікропроцесора, який важливий для вибору подальшого шляху обчислень. Наприклад, може існувати прапорець, який вказує на нульовий результат операції. Програміст може скористатися перевіркою цього прапорця для прийняття рішення: якщо певна операція дала нульовий результат, то буде виконана одна послідовність команд, в іншому випадку – інша. Прапорцеві біти, що характеризують результати операцій або будь-яких перевірок, часто розміщуються разом з іншою важливою інформацією про стан машини у спеціальному реєстрі, званому *словом стану програми* (PSW – program status word).

Пристрій управління у процесі функціонування проходить через три фази: вибірка, декодування і виконання. Після того, як програма і дані надійшли у пам'ять, адреса першої виконаної команди розміщується у програмний лічильник, і у пристрої управління встановлюється фаза вибірки. При цьому вміст програмного лічильника надходить на адресну шину, і тим самим забезпечується можливість вибірки відповідної команди і пам'яті.

Команда, що зберігається в осередку з адресою, заданою на програмному лічильнику, надсилається шинами даних у регістр команди у пристрої управління. Оскільки команди в пам'яті розташовуються у послідовних комірках, програмний лічильник збільшується на 1 і на ньому з'являється адреса наступного слова в програмі. Потім пристрій управління декодує код операції тільки що отриманої команди. Якщо код операції показує, що команда складається з більш ніж одного слова, фаза вибірки повторюється потрібну кількість разів, щоб вибрати команду повністю. При цьому кожного разу збільшується вміст програмного лічильника.

Після вибірки і декодування всієї команди пристрій управління переходить у фазу виконання. Він генерує керуючі сигнали, і відповідні схеми виконують задану в команді операцію. Якщо в команді задано адресу операнда,

пристрій управління переходить до пересилання адресованої інформації між зазначеним осередком і відповідним блоком машини, наприклад арифметичним пристроєм або пристроєм виведення.

Для здійснення пересилання адресна частина команди передається на адресну шину, готуючи наступну появу адресованої інформації на шині даних. В кінці-кінців пристрій управління забезпечує фактичне виконання заданої операції і після її завершення знову повертається до фази вибірки, щоб отримати із пам'яті наступну команду, адресу якої містить програмний лічильник. Цей процес повторюється до тих пір, поки комп'ютер не отримає вказівку зупинитися.

Загалом в описаному вище випадку команди виконуються послідовно у порядку їх розташування в пам'яті. Однак в деяких випадках цей порядок бажано змінити, наприклад залежно від стану прапорцевих бітів. У цьому випадку адреса наступної команди може не бути адресою комірки, наступної по порядку за коміркою, звідки була взята поточна виконувана команда. Команди, в яких відбуваються такі зміни порядку вибірки команд, називаються *командами переходів*, *передач управління* або *розгалужень*.

У цих командах адресна частина містить адресу наступної команди, яку має обрати пристрій управління, якщо послідовність вибірки змінюється. Тому, після того, як пристрій управління декодує команду переходу і встановить, що умови зміни порядку виконані, воно помістить адресну частину виконуваної команди, яка містить адресу наступної команди, у програмний лічильник. Таким чином, після того як пристрій управління перейде у фазу вибірки, він автоматично отримає потрібну йому наступну команду.

У багатьох додатках мікропроцесорів виникає необхідність переривати процес обчислень для обслуговування зовнішнього пристрою. Переривання може відбуватися при надходженні у процесор сигналу від зовнішнього пристрою, що вимагає уваги. У відповідь на це процесор повинен призупинити виконання поточної програми, запам'ятати стан, в якому вона була перервана (вміст різних регістрів, включаючи програмний лічильник), і потім обслужити запит зовнішнього пристрою. Завершивши обслуговування, мікропроцесор повертається до призупиненого процесу обчислення, скориставшись раніше запам’ятованою інформацією про стан перерваної програми.

Поняття переривання можна узагальнити для випадку декількох зовнішніх пристроїв. У цьому випадку будь-який з пристроїв може надіслати свій запит на увагу. Оброблення запитів може або слідувати правилу обслуговування «першим зайшов – першим вийшов», або деякою пріоритетною схемою у відповідності з пріоритетом запитів.

За усіх умов здатність обробляти переривання може виявитися дуже потужним засобом у багатьох додатках мікропроцесорів. Оскільки при перериванні потрібно зберігати інформацію про поточний стан процесора, можна знову скористатися стеком, тобто зберегти у ньому внутрішній стан мікропроцесора на момент надходження запиту, включаючи програмний лічильник. У певному сенсі переривання подібне переходу на підпрограму, але цей перехід проініційований зовнішнім сигналом, а не командою в програмі.

2.7. Пристрій мікроконтролерів AVR

Мікроконтролер AVR містить швидкий RISC-процесор, два типи енергозалежної пам’яті (Flash – пам'ять програм і пам'ять даних EEPROM), оперативну пам'ять RAM, порти введення/виведення та різноманітні інтерфейсні схеми.

Процесор

Серце мікроконтролера AVR – це 8-бітне мікропроцесорне ядро, побудоване на принципах RISC-архітектури. Основою цього блока є арифметико-логічний пристрій (АЛП). За системним тактовим сигналом з пам’яті програм відповідно змісту лічильника команд (Program Counter – PC) обирається чергова команда та виконується в АЛП. Під час вибору команди з пам’яті програм виконується попередньо обрана команда, що дозволяє досягти швидкодії 1 MIPS на 1 МГц.

Структурну модель мікроконтролера AVR показано на рис. 5.4.

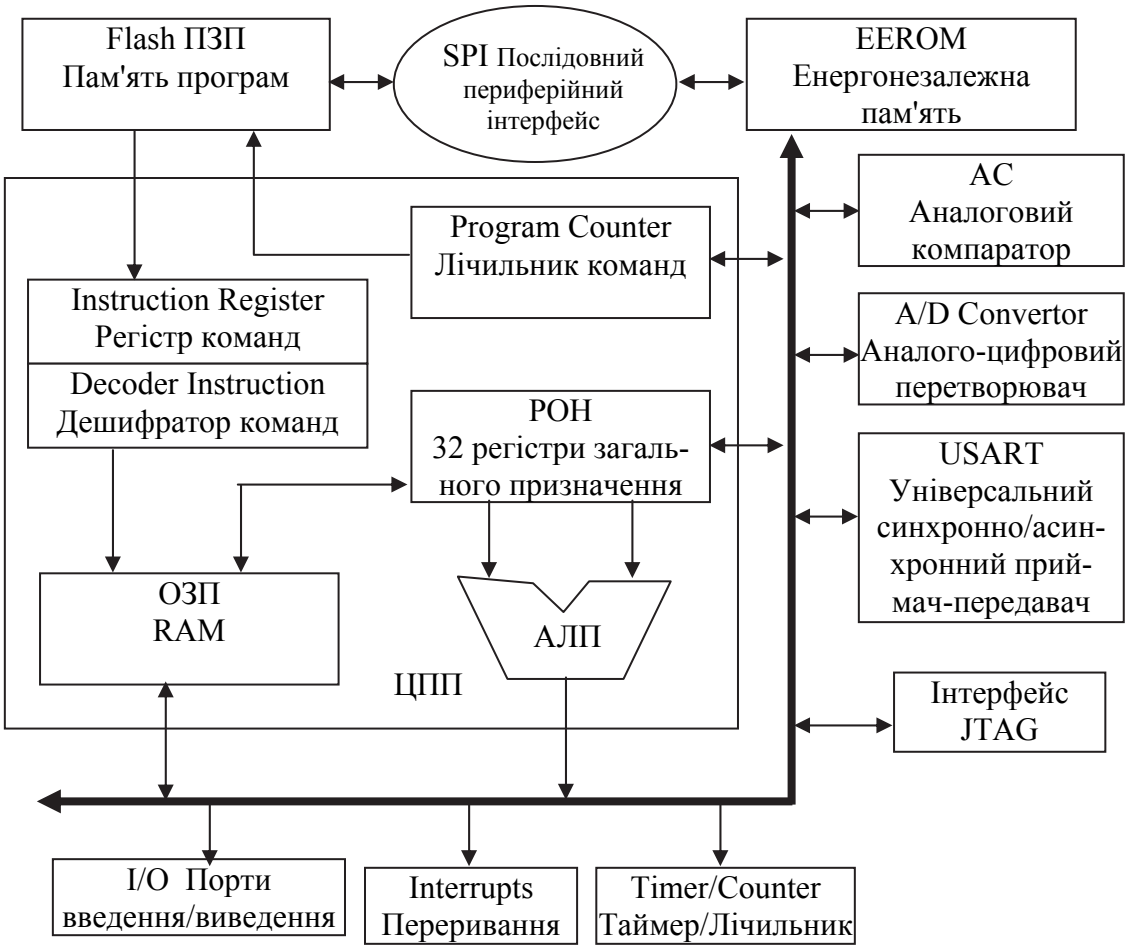


Рис. 5.4. Структурна модель мікроконтролера AVR

АЛП підімкнений до регістрів загального призначення РОН (General Purpose Registers – GPR). Регістрів загального призначення всього 32, вони мають байтів формат, тобто кожен з них складається з восьми біт. РОН

знаходиться на початку адресного простору оперативної пам'яті, але фізично не є її частиною. Тому до них можна звертатися двома способами (як до регістрів, так і до пам'яті). Таке рішення є особливістю AVR та підвищує ефективність роботи і продуктивність мікроконтролера.

Відмінність між регістрами та оперативною пам'яттю полягає у тому, що з регістрами можна виконувати будь-які операції (арифметичні, логічні, бітові), а до оперативної пам'яті можна лише записувати дані з регістрів.

Пам'ять

У мікроконтролерах AVR реалізована Гарвардська архітектура (Harvard architecture), відповідно з якою розділені не лише простори пам'яті програм та пам'яті даних, а й шини доступу до них. Кожна з областей пам'яті даних (оперативна пам'ять та EEPROM) також розташовані у своєму адресному просторі.

Пам'ять програм (Flash ROM або Flash ПЗП)

Пам'ять програм призначена для зберігання послідовності команд, клерувальних функціонуванням мікроконтролера, й має 16-бітну організацію. Усі AVR мають Flash-пам'ять програм, яка може мати розмір від 1 до 256 кбайт. Її головна перевага полягає у тому, що вона побудована за принципом електричного перепрограмування, тобто припускає багаторазове записування та стирання інформації. Програма заноситься до Flash-пам'яті AVR або за допомогою звичайного програматора, або за допомогою SPJ-інтерфейсу, у тому числі безпосередньо на власній платі. Усі мікропроцесори сімейства Mega мають можливість самопрограмування, тобто самостійно змінювати зміст своєї пам'яті програм. Ця особливість дозволяє створювати на їх основі досить гнучкі системи, алгоритм роботи яких буде змінюватися самим мікроконтролером залежно від якихось внутрішніх умов або зовнішніх подій.

Гарантована кількість циклів Flash-пам'яті у мікроконтролерів AVR другого покоління складає не менше 10 тисяч циклів при типовому значенні 100 тисяч циклів.

Пам'ять даних

Пам'ять даних розподілена на три частини: регістрова пам'ять, оперативна пам'ять (ОЗП) та енергонезалежна пам'ять (ЕСППЗП або EEPROM).

Регістрова пам'ять (РОН та РВВ)

Регістрова пам'ять містить 32 регістри загального призначення (РОН або GPR), об'єднаних у файл, та службові регістри введення/виведення (РВВ). І одні, й інші розташовані в адресному просторі ОЗП, але не є його частиною.

В області регістра введення/виведення розташовані службові регістри (регістр керування мікроконтролером, регістри станів і т.ін.), а також регістри керування периферійними пристроями, що входять до складу мікроконтролера. По суті, керування мікроконтролером полягає в керуванні цими регістрами.

Енергонезалежна пам'ять даних (EEPROM)

Для довготривалого зберігання різноманітної інформації, яка може змінюватися в процесі функціонування мікроконтролерної системи,

використовується EEPROM-пам’ять. Усі AVR мають блок енергонезалежної електрично перезаписуваної пам’яті даних EEPROM від 64 байт до 4 кбайт. Цей тип пам’яті доступний програмі мікроконтролера безпосередньо у ході її виконання, зручний для зберігання проміжних даних, різних констант, коефіцієнтів і т.п. EEPROM може бути завантажена як ззовні через SPJ інтерфейси, так і за допомогою звичайного програматора. Кількість циклів записування/стирання не менше 100 тисяч.

Оперативна пам’ять (ОЗП або RAM)

Внутрішня оперативна статична пам’ять Static RAM (SRAM) має байтів формат та використовується для оперативного зберігання даних.

Розмір оперативної пам’яті може варіюватися у різних чипів від 64 байт до 4 кбайт. Кількість циклів читання та записування до RAM необмежена, але при відключенні напруги живлення уся інформація втрачається.

Для деяких мікроконтролерів можлива організація підключення зовнішнього статичного ОЗП об’ємом до 64 К.

Периферія

Периферія мікроконтролерів AVR містить:

- порти (від 3 до 48 ліній введення та виведення);
- підтримку зовнішніх переривань;
- таймери-лічильники;
- очікуваний таймер;
- аналогові компаратори;
- 10-розрядний 8-канальний АЦП;
- інтерфейси UART, JTAG та SPL;
- пристрої скидання при зниженому живленні;
- широтно-імпульсні модулятори.

Переривання (INTERRUPTS)

Система переривань – одна з важливих частин мікроконтролера. Усі мікроконтролери AVR мають багаторівневу систему переривань. Переривання припиняє нормальний хід програми для виконання пріоритетної задачі, що визначається внутрішньою або зовнішньою подією.

Для кожної такої події розробляється окрема програма, яку називають *підпрограмою оброблення запиту на переривання* та розміщують у пам’яті програм.

При виникненні події, що викликає переривання, мікроконтролер зберігає зміст лічильника команд, перериває виконання центральним процесором поточної програми й переходить до виконання підпрограми оброблення переривання.

Після виконання підпрограми переривання здійснюється відновлення попередньо збереженого лічильника команд та процесор повертається до виконання перерваної програми.

Для кожної події може бути встановлений пріоритет. Поняття пріоритет означає, що виконувана підпрограма переривання може бути перервана іншою

подією лише за умови, що вона має більш високий пріоритет, ніж поточна. В іншому випадку центральний процесор перейде до оброблення нової події лише після закінчення оброблення попередньої.

Таймери-лічильники (TIMER-COUNTERS)

Мікроконтролери AVR мають у своєму складі від 1 до 4 таймерів-лічильників з розрядністю 8 або 16 біт, які можуть працювати і як таймери від внутрішнього джерела тактової частоти, і як лічильники зовнішніх подій.

Їх можна використовувати для точного формування часових інтервалів, підрахунку імпульсів на виводах мікроконтролера, формування послідовності імпульсів, тактування приймально-передавального послідовного каналу зв'язку. У режимі ШІМ таймер-лічильник може являти собою широтно-імпульсний модулятор та використовуватися для генерування сигналу з програмованою частотою та щілинністю. Таймери-лічильники здатні виробляти запити переривань, перемикаючи процесор на їх обслуговування за подіями та звільняючи його від необхідності періодичного запиту стану таймерів. Оскільки основне застосування мікроконтролери знаходять у системах реального часу, таймери-лічильники є одним із найбільш вагомих елементів.

Аналого-цифровий перетворювач (A/D convertor)

Аналого-цифровий перетворювач (АЦП) слугує для отримання числового значення напруги, поданої на його вхід. Цей результат зберігається у регістрі даних АЦП. Який з виводів мікроконтролера є входом АЦП, визначається числом, занесеним у відповідний регістр.

Універсальний послідовний приймач-передавач (UART або USART)

Універсальний асинхронний або універсальний синхронний приймач-передавач – зручний та простий послідовний інтерфейс для організації інформаційного каналу обміну мікроконтролера із зовнішнім світом. Здатен працювати у дуплексному режимі (одночасне передавання та приймання даних). Він підтримує протокол стандарту RS-232, що забезпечує можливість організації зв'язку з персональним комп'ютером. Для стикування мікроконтролера та комп'ютера обов'язково необхідна схема поєднання рівнів сигналу. Для цього існують спеціальні мікросхеми, наприклад, MAX232.

Інтерфейс JTAG

Інтерфейс JTAG був розроблений групою провідних спеціалістів з проблем тестування електронних компонентів та був зареєстрований як промисловий стандарт (IEEE). Чотиридротовий інтерфейс JTAG використовується для тестування друкованих плат, внутрішньосхемного налагодження, програмування мікроконтролерів.

Багато мікроконтрелерів сімейства MEGA мають сумісний з IEEE інтерфейс JTAG або debug WIRE для вбудованого налагодження. Крім того, всі мікроконтролери MEGA з флеш-пам'яттю ємністю 16 кбайт і більше можуть програмуватися через інтерфейс JTAG.

Стек

У комірках оперативної пам’яті організується системний стек, який використовується автоматично для зберігання адрес повернення при виконанні підпрограм, а також може використовуватися програмістом для тимчасового зберігання змісту оперативних регістрів (команди PUSH і POP). Мікроконтролери, що не мають SRAM, містять трирівневий апаратний стек.

Слід мати на увазі, що якщо стек розміщений на зовнішній SRAM, то виклики підпрограм та повернення із них вимагають двох додаткових циклів, якщо біт SRW не встановлений, та чотирьох, якщо встановлений.

Розмір стека, що організується в оперативній пам’яті, обмежений лише розмірами цієї пам’яті. Якщо мікроконтролер містить на кристалі 28 байт внутрішньої SRAM та не має можливості відімкнення зовнішньої SRAM, то в якості вказівника вершини стека використовується регістр введення/виведення SPL. Якщо є можливість відімкнення зовнішньої пам’яті або внутрішня пам’ять має розміри 256 байт і більше, то вказівник стека складається з двох регістрів введення/виведення SPL і SPH.

При занесенні числа до стека автоматично виконуються наступні дії:

– число записується до комірки пам’яті за адресою, що зберігається у вказівнику стека. При цьому (SPH : SPL) < число;
– вміст вказівника стека зменшується на одиницю: SPH : SPL = = SPH : SPL – 1.

Обернені дії виконуються при витягненні числа зі стека:

– вміст вказівника збільшується на одиницю: SPH : SPL = SPH : SPL + 1;
– число витягується з комірки пам’яті з адресою, що зберігається у вказівнику стека: (SPH : SPL) > число.

Таким чином, стек зростає від старших адрес до молодших, тому, враховуючи, що початкове значення вказівника стека після скидання дорівнює нулю, програміст AVR обов’язково повинен в ініціалізованій частині програми потурбуватися про встановлення вказівника стека, якщо він припускає використати хоча б одну підпрограму.

2.8. Система команд

2.8.1. Загальні відомості

Зазвичай мікропроцесор виконує команди з деякого фіксованого набору, який називається системою команд. Різні мікропроцесори мають різні системи команд. Для кожної команди наводиться її символічне позначення, формат, а також символічний та словесний опис виконуваної дії. Ніяких інших команд обраний мікропроцесор виконувати не може. Отже, розв’язання будь-якої задачі має бути подано у вигляді послідовності команд даного мікропроцесора.

Формат кожної команди повністю визначає подання команди до пам’яті комп’ютера. Таким чином, команди у пам’яті зберігаються як послідовність одиниць або нулів. З іншого боку, для людини зручніше символічне зображення. Тому при написанні програм можна користуватися символічної

формою, перетворюючи її у машинний двійковий код безпосередньо перед введенням у машину.

Час, необхідний для того, щоб прочитати з пам'яті команду, декодувати її та виконати, називається командним циклом.

Завдяки тому, що команди можуть складатися з одного, двох або трьох байтів та те, що час їх виконання може бути різним, командний цикл не є постійною величиною.

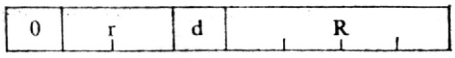
Для знайомства використаємо систему команд ілюстрованого мікропроцесора.

2.8.2. Команди пересилання

Команда: MOVE (пересилання)

Символічна форма: MOV r to R або MOV r from R

Формат:



Описання:

$(r) \xrightarrow{d=0} R$ або $(R) \xrightarrow{d=1} r$

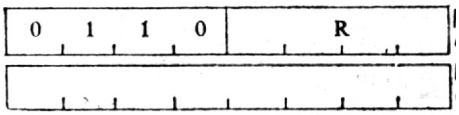
Якщо d = 0, то вміст загального регістра r передається у загальний регістр R.

Якщо d = 1, то вміст загального регістра R передається у загальний регістр r.

2.8.3. Команди з безпосередньою адресацією

Команда: LOAD REGISTER IMMEDIATE (завантаження регістра безпосереднє)

Формат:



Перший байт

Другий байт

Описання: $(B_r) \longrightarrow R$

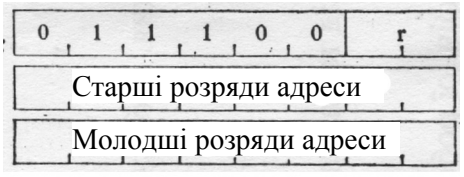
Другий байт команди передається у загальний регістр R.

2.8.4. Команди звернення до пам'яті

Команда: LOAD REGISTER (завантаження регістра)

Символічна форма: LDR r

Формат:



Перший байт

Другий байт

Третій байт

Описання: $(M[\langle B_2 \rangle \langle B_3 \rangle) \longrightarrow r$

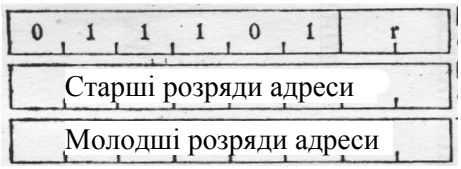
Вміст комірки пам'яті передається у загальний регістр r.

Старші 8 розрядів адреси комірки беруться з другого байта команди, молодші 8 розрядів – з третього байта.

Команда: STORE REGISTER (запам'ятовування регістра)

Символічна форма: STR r

Формат:



Перший байт
Другий байт
Третій байт

Описанyz: $r \to M[\langle B_2 \rangle \langle B_3 \rangle$

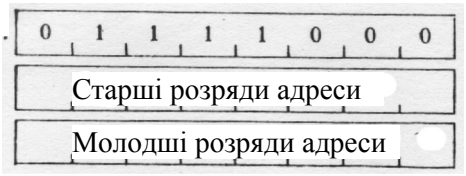
Вміст загального регістра r передається у комірку у головній пам'яті. Старші 8 розрядів адреси комірки зазначені у другому байті команди, а молодші 8 розрядів – у третьому байті.

2.8.5. Команди передавання управління

Команда: JUMP ON CARRY NOT-ZERO (перехід за ненульового перенесення)

Символічна форма: JCN

Формат:



Перший байт
Другий байт
Третій байт

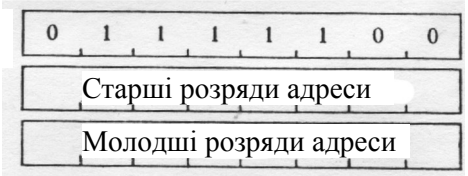
Описання: якщо (С) = 1, то $\langle B_2 \rangle \langle B_3 \rangle \to PC$

Якщо на тригері перенесення 1, то вміст лічильника команд заміщається другим та третім байтами команди JCN, причому другий байт заміщує старші 8 розрядів лічильника команд, а третій байт – молодші 8 розрядів , що викликає передавання управління у задану комірку. Інакше, коли (С) = 0, другий та третій байт команди ігнорується, і виконується наступна за порядком команда.

Команда: JUMP ON CARRY ZERO (перехід за нульового перенесення)

Символічна форма: JCZ

Формат:



Перший байт
Другий байт
Третій байт

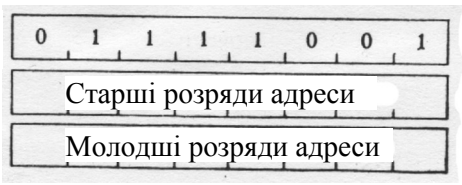
Описання: якщо (С) = 0, то $\langle B_2 \rangle \langle B_3 \rangle \to PC$

Якщо на тригері перенесення 0, то вміст лічильника команд заміщається другим та третім байтами команди JCZ, причому другий байт заміщує старші 8 розрядів лічильника команд, а третій байт – молодші 8 розрядів , що викликає передавання управління у задану комірку. Коли (С) = 1, другий та третій байт команди ігнорується, і виконується наступна за порядком команда.

Команда: JUMP ON ACCUMULATOR ZERO (перехід за нульового акумулятора)

Символічна форма: JAZ

Формат:



Перший байт
Другий байт
Третій байт

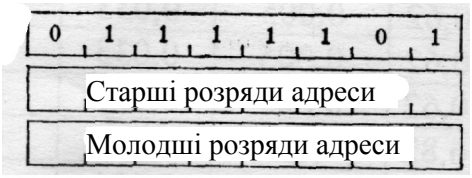
Описання: якщо (Acc) = 0000 0000, то $\langle B_2 \rangle \langle B_3 \rangle \rightarrow PC$

Якщо вміст акумулятора дорівнює нулю, то вміст лічильника команд замещується другим та третім байтами команди JAZ, причому другий байт заміщує старші 8 розрядів лічильника команд, а третій байт – молодші 8 розрядів, що викликає передавання управління у задану комірку. Інакше, тобто коли (Acc) ≠ 0, другий та третій байти команди ігноруються, і виконується наступна за порядком команда.

Команда: JUMP ON ACCUMULATOR NOT ZERO (перехід за ненульового акумулятора)

Символічна форма: JAN

Формат:



Перший байт
Другий байт
Третій байт

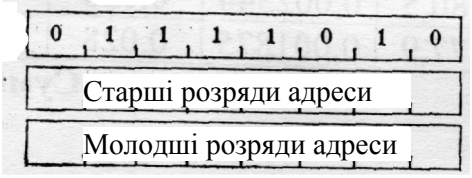
Описання: Якщо (Acc) ≠ 0000 0000, то $\langle B_2 \rangle \langle B_3 \rangle \rightarrow PC$

Якщо вміст акумулятора не дорівнює нулю, то вміст лічильника команд замещується другим та третім байтами команди JAN, причому другий байт заміщує старші 8 розрядів лічильника команд, а третій байт – молодші 8 розрядів, що викликає передавання управління у задану комірку. Коли вміст акумулятора дорівнює 0, другий та третій байти команди ігноруються, і виконується наступна за порядком команда.

Команда: JUMP ON ACCUMULATOR POSITIVE (перехід за додатного акумулятора)

Символічна форма: JAP

Формат:



Перший байт
Другий байт
Третій байт

Описання: Якщо (Acc_1) = 0, то $\langle B_2 \rangle \langle B_3 \rangle \rightarrow PC$

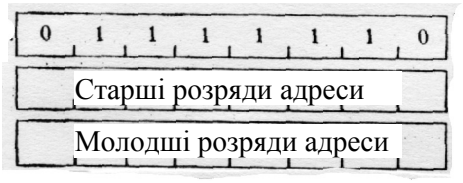
Якщо у старшому розряді акумулятора 0, то вміст лічильника команд замещується другим та третім байтами команди JAP, причому другий байт заміщує старші 8 розрядів лічильника команд, а третій байт – молодші 8

розрядів, що викликає передавання управління у задану комірку. Інакше, другий та третій байти команди ігноруються, і виконується наступна за порядком команда.

Команда: JUMP ON ACCUMULATOR MINUS (перехід за від'ємного акумулятора)

Символічна форма: JAM

Формат:



Перший байт
Другий байт
Третій байт

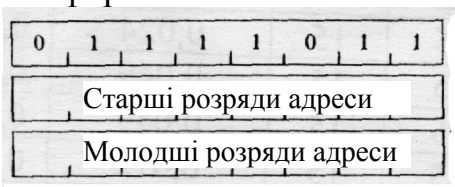
Описання: Якщо $(\text{Acc}_1) = 1$, то $\langle B_2 \rangle \langle B_3 \rangle \rightarrow PC$

Якщо у старшому розряді акумулятора 1, то вміст лічильника команд заміщується другим та третім байтами команди JAM, причому другий байт заміщує старші 8 розрядів лічильника команд, а третій байт – молодші 8 розрядів, що викликає передавання управління у задану комірку. Якщо $(\text{Acc}_1) = 0$, то другий та третій байти команди ігноруються, і виконується наступна за порядком команда.

Команда: JUMP UNCONDITIONALLY (перехід безумовний)

Символічна форма: JMP

Формат:



Перший байт
Другий байт
Третій байт

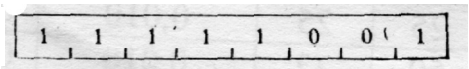
Описання: $\langle B_2 \rangle \langle B_3 \rangle \rightarrow PC$

Вміст лічильника команд заміщується другим та третім байтами команди JMP, причому другий байт заміщує старші 8 розрядів лічильника команд, а третій байт – молодші 8 розрядів, що викликає передавання управління у задану комірку.

Команда: JUMP INDIRECT (перехід за непрямою адресою)

Символічна форма: JHL

Формат:



Описання: (H)(L) → PC.

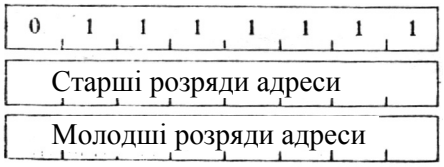
Вміст лічильника команд заміщується вмістом загальних регістрів H та L, причому вміст H заміщує старші 8 розрядів лічильника команд, а вміст L – молодші 8 розрядів, що викликає передавання управління у комірку з адресою, заданою на регістрах H та L.

Команди звернення до підпрограм

Команда: JUMP TO SUBROUTINE (перехід на підпрограму)

Символічна форма: JMS

Формат:



Описання: $(\text{Stack}_i) \rightarrow \text{Stack}_{i+1}$ $i = 1, 2, \ldots, 63_{10}$

$(\text{PC}) \rightarrow \text{Stack}_i$

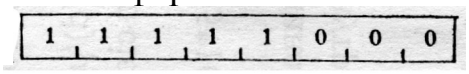
$\langle B_2 \rangle \langle B_3 \rangle \rightarrow PC$

Стек опускається і вміст лічильника команд поміщується у вершину стека. Вміст останнього регістру стека (Stack_{64}) губиться. Вміст лічильника команд заміщується другим і третім байтами команди JMS, причому другий байт заміщує старші 8 розрядів лічильника команд, а третій байт – молодші 8 розрядів, щі викликає передачу управління у задану комірку.

Команда: RETURN FROM SUBROUTINE (повернення з підпрограми)

Символічна форма: RET

Формат:



Описання: $(\text{Stack}_i) \rightarrow \text{PC}$

$(\text{Stack}_i) \rightarrow \text{Stack}_{i-1}$ $i = 2, 3, \ldots, 64_{10}$

$(\text{Stack}_{64}) \rightarrow \text{Stack}_{01}$

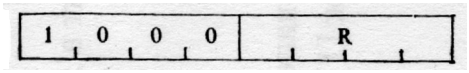
Вміст вершини стека передається у лічильник команд і стек піднімається. Вміст останнього регістра стекв (Stack_{64}) змінюється.

2.8.6. Арифметичні та логічні команди

Команда: ADD REGISTER (додавання з регістром)

Символічна форма: ADD R

Формат:



Описання: $(\text{Acc}) + (\text{R}) \rightarrow \text{Acc}$

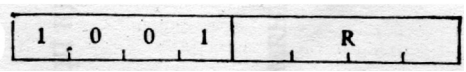
Перенесення $\rightarrow$ C

Вміст спільного регістра R додається до вмісту акумулятора. Результат додавання стає новим вмістом акумулятора, а перенесення зі старшого розряду стає новим вмістом тригера перенесення C. Усі числа вважаються цілими без знаків.

Команда: ADD REGISTER WITH CARRY (додавання з регістром та перенесенням)

Символічна форма: ADC R

Формат:

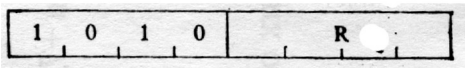


Описання: $(\text{Acc}) + (\text{R}) + (\text{C}) \rightarrow \text{Acc}$

Вміст спільного регістра R і тригер перенесення C додаються до вмісту акумулятора. Результат додавання стає новим вмістом акумулятора, а перенесення зі старшого розряду стає новим вмістом тригера перенесення C. Усі числа вважаються цілими без знаків.

Команда: SUBSTRACT REGISTER (віднімання регістру)

Символічна форма: SUB R

Формат: 

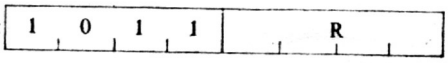
Описання: (Acc) – (R) → Acc

Позика → Acc

Вміст спільного регістра R віднімається від вмісту акумулятора. Результат віднімання стає новим вмістом акумулятора, а позика до старшого розряду стає новим вмістом тригера перенесення. Усі числа вважаються цілими без знаків.

Команда: SUBSTRACT REGISER WITH CARRY (віднімання регістра та переносу)

Символічна форма: SBC R

Формат: 

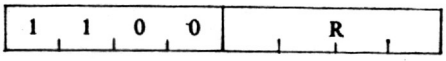
Описання: (ACC) – (R) – (C) → Acc

Позика → Acc

Вміст спільного регістра R та тригера перенесення C віднімаються від вмісту акумулятора. Результат віднімання стає новим вмістом акумулятора, а позика до старшого розряду стає новим вмістом тригера перенесення. Усі числа вважаються цілими без знаків.

Команда: LOGICAL AND (логічне І)

Символічна форма: AND R

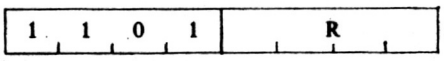
Формат: 

Описання: (Acc) $\wedge$ (R) → Acc

Над вмістом спільного регістра R та акумулятора порозрядно виконується операція логічного І. Результат стає новим вмістом акумулятора. Стан тригера перенесення не змінюється.

Команда: LOGICAL OR (логічне АБО)

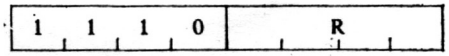
Символічна форма: OR R

Формат: 

Описання: (Acc) $\vee$ (R) → Acc

Над вмістом спільного регістра R та акумулятора порозрядно виконується операція логічного АБО. Результат стає новим вмістом акумулятора. Стан тригера перенесення не змінюється.

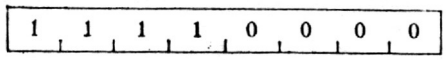
Команда: LOGICAL EXCLUSIVE-OR (логічне ВИКЛЮЧАЮЧЕ АБО)
Символічна форма: XOR R

Формат: 

Описання: (Acc) $\oplus$ (R) $\rightarrow$ Acc

Над вмістом спільного регістра R та акумулятора порозрядно виконується операція логічного ВИКЛЮЧАЮЧОГО АБО. Результат стає новим вмістом акумулятора. Стан тригера перенесення не змінюється.

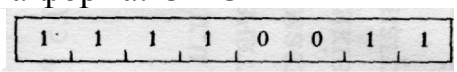
Команда: COMPLEMENT ACCUMULATOR (перетворення акумулятора)
Символічна форма: CMA

Формат: 

Описання: $(\overline{\text{Acc}}) \rightarrow$ Acc

Нулі у акумуляторі замінюються на одиниці,а одиниці – на нулі (тобто відбувається перетворення коду в акумуляторі)

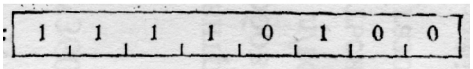
Команда: COMPLEMENT CARRY (перетворення переносу)
Символічна форма: CMC

Формат: 

Описання: $(\overline{\text{C}}) \rightarrow$ C

Якщо стан тригера перенесення C рівний 0, він змінюється на 1. Або навпаки скидається у 0.

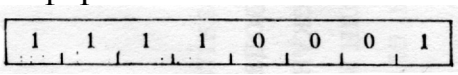
Команда: RESET CARRY (скидання переносу)
Символічна форма: RSC

Формат: 

Описання: 0 $\rightarrow$ C

Тригер переносу скидається.

Команда: ROTATE ACCUMULATOR AND CARRY LEFT (циклічний зсув акумулятора та перенесення вліво)
Символічна форма: RTL

Формат: 

Описання: (C) $\rightarrow$ Acc_0

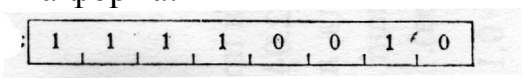
$(\text{Acc}_i) \rightarrow \text{Acc}_{i+1}$ $i = 0, 1, \ldots 6$

$(\text{Acc}_7) \rightarrow$ C

Вміст акумулятора та тригера перенесення C як єдине ціле зміщуються циклічно вліво. Вміст розряду Acc_7 потрапляє на тригер C, а вміст тригера C – у розряд Acc_0.

Команда: ROTATE ACCUMULATOR AND CARRY RIGHT (циклічний зсув акумулятора та переносу вправо)

Символічна форма: RTR

Формат: 

Описання: (C) → Acc_7

$(Acc_i) \rightarrow Acc_{i-1}$ $i = 0, 1, \ldots 7$

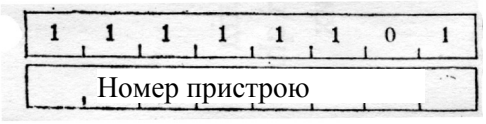
$(Acc_0) \rightarrow C$

Вміст акумулятора та тригера перенесення С як єдине ціле зміщуються циклічно вправо. Вміст розряду Acc_0 потрапляє на тригер С, а вміст тригера С – у розряд Acc_7.

2.8.7. Команди введення/виведення

Команда: INPUT (ввід)

Символічна форма: INP

Формат: 

Перший байт

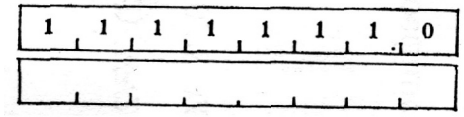
Другий байт

Описання: Пристрій введення $[\langle B_2 \rangle] \rightarrow$ Acc

Байт даних передається в акумулятор від пристрою введення, номер якого заданий другим байтом команди INP.

Команда: OUTPUT (виведення)

Символічна форма: OUT

Формат: 

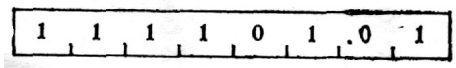
Описання: Acc → Пристрій виведення $[\langle B_2 \rangle]$

Вміст акумулятора передається на пристрій виведення, номер якого заданий другим байтом команди OUT.

2.8.8. Спеціальні команди

Команда: INCREMENT REGISTER PAIR H AND L (збільшення на 1 вмісту пари регістрів H та L)

Символічна форма: IHL

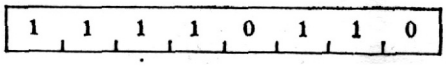
Формат: 

Описання: (H) (L) + 1 → HL

16-розрядне число, що міститься у регістрах H та L, збільшується на 1 за модулем 2^{16}.

Команда: DECREMENT REGISTER PAIR H AND L (зменшення на 1 вмісту пари регістрів H та L)

Символічна форма: DHL

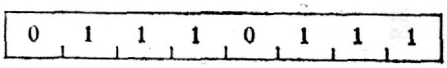
Формат: 

Описання: (H) (L) – 1 → HL

16-розрядне число, що міститься у регістрах H та L, зменшується на 1 за модулем 2^{16}.

Команда: PUSH DATA ONTO STACK (помістити дані у стек)

Символічна форма: PUSH

Формат: 

Описання: $(Stack_i) \rightarrow Stack_{i+2}$ $i = 2, 3, \ldots, 62_{10}$

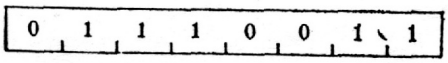
$0000000(C)(Acc) \rightarrow Stack_2$

$(H)(L) \rightarrow Stack_1$

Стек опускається двічі. У результаті у другому регістрі стеку виявляється значення акумулятора з перенесенням, а у першому(вершина стеку) вміст пари регістрів H та L.

Команда: POP DATA FROM STACK (вилучити дані зі стека)

Символічна форма: POP

Формат: 

Описання: $(Stack_1) \rightarrow HL$

$(Stack\ L_2) \rightarrow Acc$

$(Stack\ HO_2) \rightarrow C$

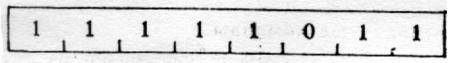
$(Stack_i) \rightarrow Stack_{i-2}$ $i = 3, 4, \ldots, 64_{10}$

$(Stack_j) \rightarrow Stack_j$ j =63, 64

Вміст вершини стеку передається парі регістрів H та L. Вміст молодшої половини другого регістру стеку передається на акумулятор, а молодший біт старшої половини цього ж регістру передається тригеру перенесення C. Стек піднімається двічі, причому стан двох нижніх регістрів залишається без змін.

Команда: ENABLE INTERRUPT (дозволити переривання)

Символічна форма: EIT

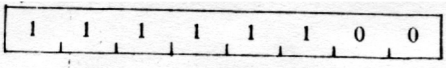
Формат: 

Описання: 1 → прапорець переривань

На тригері, який називається прапорцем переривань, встановлюється 1 після виконання команди, яка слідує за даною командою.

Команда: DISABLE INTERRUPT (заборонити переривання)

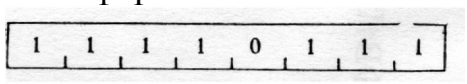
Символічна форма: DIT

Формат: 

Описання: 0 → прапорець переривань

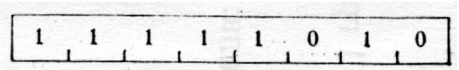
Тригер, який називається прапорцем переривань, скидається.

Команда: NO OPERATION (пуста команда)
Символічна форма: NOP

Формат: 

Описання: При виконанні цієї команди ніяких дій не виконується.

Команда: HALT(зупин)
Символічна форма: HLT

Формат: 

Описання: Виконання команд зупиняється до приходу імпульсу на лінію «пуск».

Позначення, які використовуються при описанні команд, наданих у табл. 2.1.

Позначення	Коментар
(X)	Вміст X
r	Код з 2 бітов, що означає один із загальних регістрів 0000, 0001 і 0010 (тобто Acc, H і L) двобітовими комбінаціями 00, 01 і 10 відповідно
R	Номер будь-якого загального регістра від 0000 до 1111
d	Вказівник напряму передавання. 0 відповідає «to», 1 – «from» (тобто «в» або «від»)
→	Передається на
Acc	Аккмулятор, тобто загальний регістр 0000
Acc_1	Розряд акумулятора з номером *i*. Розряді нумеруються від молодших до старшим, t = 0, 1, … ,7
C	Тригер перенесення
H	Загальний регістр 0001
L	Загальний регістр 0010
HL	Пара регістрів H та L
M	Комірка головної пам’яті з адресою (H) (L); посилання на неї задається як посилання на загальний регістр 1111
$<B_2>$	Другий байт команди
$<B_3>$	Третій байт команди
M[X]	Комірка головної пам’яті з адресою X
PC	Програмний лічильник
$Stack_i$	Регістр стека з номером *i*
Stack L_2	Младшая половина второго регистра стека
Stack HO_2	Молодший розряд старшої половины другого регістра стека
+	Арифметичее додавання
	Логічне множення (І)
	Логічне додавання (АБО)
(+)	Логічне, що ВИКЛЮЧАЄ АБО

2.9. Фаза вибірки та дешифрування

Першим етапом виконання кожної команди є прочитання цієї команди з пам’яті. Читання кожного байта команди займає один машинний цикл. На рис. 2.2 показано основні інформаційні потоки у фазі вибірки-дешифрування командного циклу. 16-бітова адреса першого байта команди передається у пам'ять шиною адреси із програмного лічильника. Пристрій управління формує сингал «читання», завдяки якому зміст адресованої комірки видається з пам’яті на шину даних, потім приймається регістром команд. Цей перший байт, який мстить код операції, дешифрується ДК, при цьому прояснюється, скільки байтів у команді.

Одночасно збільшується програмний лічильник.

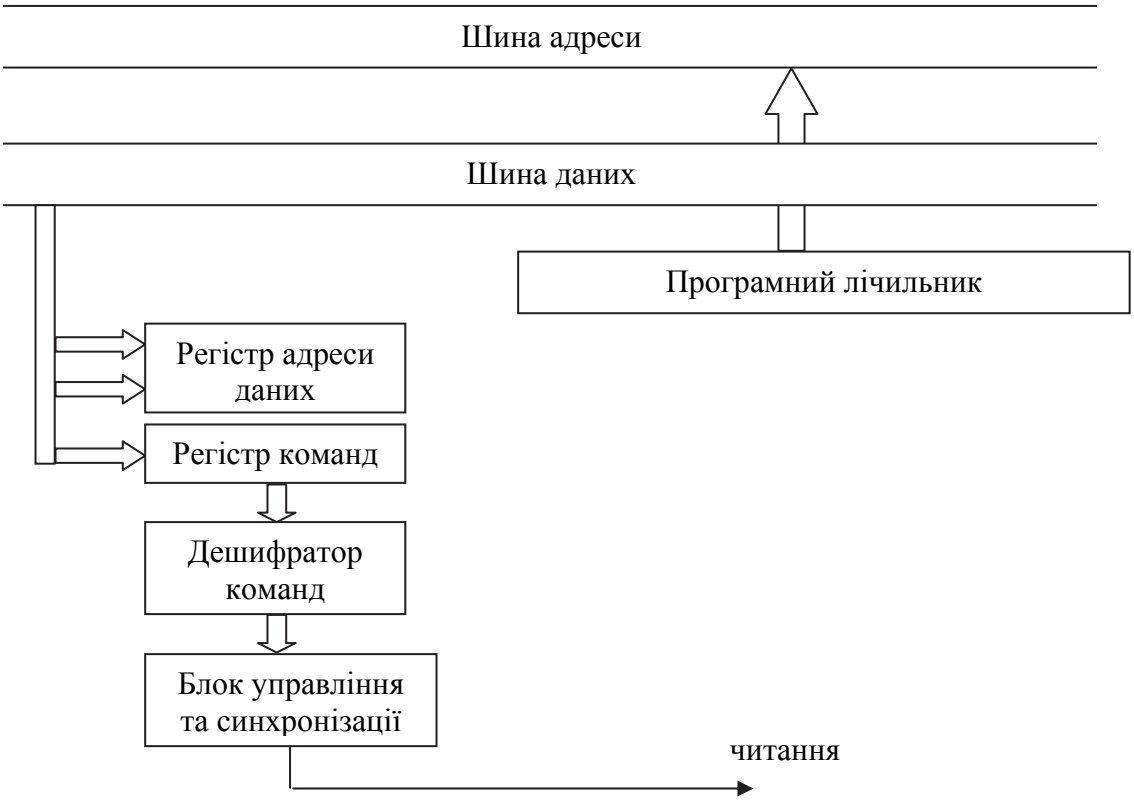


Рис. 2.6. Інформаційні потоки у фазі вибірки-дешифрування командного циклу

Якщо з’ясовується, що у команді є ще один або два байти, то задіюються ще один або два машинних цикли на те, щоб аналогічним чином вибрати наступний байт із пам’яті.

Приймає ці байти вже не регістр команд, а або регістр адреси даних, або один із загальних регістрів.

Фаза вибірки-дешифрування завершується після другого синхроімпульсу останнього машинного циклу.

2.10. Фаза виконання

Наступний етап після вибірки – це виконання команд.

Для виконання деяких команд достатньо третього залишеного синхроімпульсу останнього машинного циклу, для інших команд вимагаються додатково машинні цикли.

Кількість додаткових циклів визначається у блоці управління та синхронізації залежно від виконуваної команди.

2.11. Деякі інші команди, специфічні для мікропроцесорів

У випусканих мікропроцесорах нерідко можна зустріти команди, яких немає в нашому ілюстративному процесорі. Одні з них є просто різновидами команд нашого мікропроцесора, інші більш зручні з точки зору програмування, треті розширюють обчислювальну потужність мікропроцесора.

Арифметичні та логічні команди з операндами у головній пам’яті

В ілюстративному процесорі усі арифметичні та логічні команди працювали з операндами, які або знаходяться на регістрах мікропроцесора, або адресуються непрямо за допомогою регістрів H та L. У багатьох мікропроцесорах зустрічаються арифметичні та логічні команди з прямою адресацією операнда в головній пам'яті. За такою схемою команда додавання, наприклад, додає вміст комірки пам'яті до вмісту внутрішнього регістра, що служить акумулятором, і поміщає результат у внутрішній регістр. При заданні у команді прямої адреси операнда у головній пам'яті довжина команди, природно, зростає.

У деяких мікропроцесорах можливості розширюються ще більше та передбачається запис результату безпосередньо у головну пам'ять. Прикладами можуть слугувати команди, які інвертують вміст комірки пам’яті або збільшують його на одиницю.

2.11.1. Команди умовних переходів

У нашому мікропроцесорі вибір напряму обчислень виконується за допомогою шести команд умовних переходів, в яких аналізуються три умови: чи дорівнює нулю вміст акумулятора, чи додатній він та чи дорівнює нулю біт тригера перенесення. У багатьох мікропроцесорах, окрім перерахованих, аналізуються ще такі умови:

1. Переповнення при арифметичних операціях.
2. Парність (або непарність) числа одиниць у вмісті регістру.
3. Сувора позитивність вмісту регістра (позитивне і не дорівнює нулю).
4. Різні зовнішні умови, що задаються сигналами на спеціальних вхідних лініях.

У багатьох мікропроцесорах команди умовних переходів реагують на стан окремих прапорцевих тригерів. Значення цих тригерів встановлюються у момент виконання певних операцій.

2.11.2. Команди обробки даних

Операції, що виконуються в ілюстративному процесорі за допомогою арифметичних і логічних команд, є базисними і зустрічаються у більшості мікропроцесорів. У деяких мікропроцесорах, крім того, передбачені команди десяткової арифметики, двійкового множення, двійкового ділення та інших операцій.

2.12. Мікропроцесорні комплекти

Завдяки програмному управлінню та гнучкості інтерфейсів препроцесори, очевидно, є пристроями з дуже широким колом застосувань. Це коло ще більш розширюється завдяки низькій вартості та малим розмірам мікропроцесорів.

Проте при створенні кожної конкретної прикладної мікропроцесорної системи важливу роль відіграють питання визначення необхідних додаткових компонент і їх взаємозв'язків.

Щоб полегшити проектування і комплектацію систем, фірми-виробники мікропроцесорних комплектів включають до їх складу різні функціональні підсистеми, оформлені у вигляді великих інтегральних схем. Як правило, такі підсистеми виконують основні інтерфейсні функції і можуть налаштовуватися на застосування у тих чи інших конкретних умовах. Наприклад, випускаються різні універсальні асинхронні реверсивні трансмітери (UART) для управління послідовним введенням/виведенням з широким набором можливостей.

Взагалі, якщо функціональна підсистема може налаштовуватися на ті чи інші можливості, її називають *програмованою*. Програмування може здійснюватися подачею на відповідні керуючі лінії рівнів напруг логічного 0 або логічної 1; або ж завантаженням з боку мікропроцесора керуючих слів у регістри, що знижує число необхідних керуючих ліній. До складу мікропроцесорних комплектів зазвичай входять наступні підсистеми:

1. Програмовані порти введення/виведення, що містять, як правило, схеми декількох портів для даних, разом з необхідними портами управління і стану. Програмовані можливості зазвичай пов'язані з адресацією портів і вибором методу організацій введення / виведення. Часто включаються також схеми, що формують запити на переривання за інформацією про стан портів.

2. Програмовані інтервальні таймери, що складаються з одного або декількох реверсивних лічильників, що підключаються до мікропроцесора як порти введення і виведення і здатних працювати у декількох режимах. В одному з режимів лічильник інформує мікропроцесор про закінчення заданого інтервалу часу. У цьому випадку початкове значення на лічильник завантажується мікропроцесором, і кожен імпульс від зовнішнього генератора зменшує лічильник на одиницю. Коли лічильник досягає нуля, або встановлюється прапорець стану, або видається запит на переривання. В інших режимах лічильник може служити в якості годинника реального часу, лічильника подій або джерела імпульсів зі змінною частотою.

3. Контролери векторних систем переривання зі схемами вирішення конфліктів при одночасному надходженні запитів на переривання і схемами видачі потрібної інформації у мікропроцесор.

4. Контролери прямого доступу до пам'яті зі схемами, які керують передачею блоків слів між пам'яттю і пристроями введення / виведення.

5. Контролери для регенерації динамічної пам'яті, що забезпечують періодичну регенерацію даних у модулях ЗП, підключених до мікропроцесора.

6. Генератори тактових імпульсів, що видають серії синхроімпульсів із заданою частотою і потрібними фазовими співвідношеннями, а також інші спеціальні часові сигнали.

7. Пристрої зв'язку із зовнішніми об'єктами, що містять один або кілька ЦА-перетворювачів і/або один або кілька АЦ-перетворювачів, що підключаються до відповідних портів введення/виведення.

8. Розсширювачі арифметичних пристроїв типу помножувачів, подільників та генераторів тригонометричних функцій, що працюють через що порти введення / виведення, які входять до їх складу.

У принципі подібні підсистеми-модулі проектуються таким чином, щоб звести до мінімуму число зв'язків з мікропроцесором. За електричними та часовими параметрами сигналів вони узгоджуються з відповідними мікропроцесорами. Таким чином, проблема проектування багатьох мікропроцесорних систем зводиться лише до з'єднання таких підсистем-модулів з мікропроцесором. Хоча, звичайно, можуть виникати й інші специфічні для конкретного додатка проблеми, які потребують проектування спеціальних схем і компонентів.

Щоб ще більше спростити компонування мікропроцесорних систем, розроблені й випускаються так звані *однокристальні мікрокомп'ютери*. Крім мікропроцесора, однокристальний мікрокомп'ютер зазвичай містить ОЗП і ПЗП невеликого об’єму, кілька портів введення/виведення і низку інших базових підсистем. При цьому суттєво знижується кількість з'єднань у таких мікрокомп'ютерних системах, де вимоги до обсягу пам'яті і кількості пристроїв введення/виведення невеликі. У цьому випадку значна кількість виводів до шин зникає, тим самим звільняється місце для зовнішніх зв'язків з портами введення/виведення. У деяких системах одні й ті самі лінії програмуються або для виконання функцій шин, або для зв'язків з портами введення / виведення.

Велика увага при створенні мікрокомп'ютерних систем приділяється перевірці правильності їх функціонування. Сюди входить як налаштування програмного забезпечення, так і перевірка апаратури. Завдяки тісному переплетенню програмних і апаратних засобів та різноманітності взаємодії компонентів питання налагодження і тестування можуть виявитися вельми складними. У зв'язку з цим широке поширення отримали різні прилади і системи, які дозволяють проводити налагодження і тестування програмних і апаратних компонент незалежно. До числа таких засобів відносяться:

1. Системи проектування для конкретних мікрокомп'ютерів, що включають пульт управління і попередньо скомпоновані апаратні модулі, такі, як плата з мікропроцесором, система пам'яті з ОЗП і програмованим ПЗП, порти

введення/виведення та інші інтерфейсні блоки. Це дає можливість створювати прототипи майбутніх систем на самій ранній стадії їх розробки.

2. Вимірювальні комплекти, що складаються з невеликих мікрокомп'ютерних систем на одній або кількох друкованих платах.

3. Аналізатори мікропроцесорів, які реєструють і видають вміст шин та інших ліній працюючого мікрокомп'ютера у кожному машинному циклі.

4. Емулятори апаратури, що дозволяють промоделювати ту чи іншу частину мікрокомп'ютерної системи, щоб отримати часові співвідношення та передавані між різними компонентами сигнали.

5. Програмні інструментальній засоби, наприклад, програми моделювання мікропроцесорів на комп’ютерах, а також крос-асемблери та крос-комп’ютери.

2.13. Система синхронізації мікропроцесорних комплектів

2.13.1. Система двофазної синхронізації

Мікропроцесорні комплекти, як і більшість цифрових пристроїв, використовують синхронний принцип роботи.

Найбільш простою системою синхронізації є двофазна або двотактова. Тут усі схеми пристрою тактуються двома взаємно перемежовуваними у часі послідовностями синхроімпульсів (рис. 2.7) Сі та C_2 (clock pulses).

Для вироблення такої послідовності використовується один задавальний генератор. Вихідні П-імпульси генератора не несуть жодної інформації. Вони слугують лише для прив’язки у часі усіх процесів пристрою.

Основні параметри такої системи:

– тривалість тактового періоду 7’,;

– тривалість фазового періоду $7’_ф$;

– тривалість синхроімпульси 7).

Для симетричної двофазної синхронізації

$$T_т = 2T_ф. \qquad (2.1)$$

На рис. 2.7,а подана схема, на якій показані основні риси ідеї двотактової синхронізації. Тут усі логічні схеми розділені на два класи: послідовні сні (MRG1 і MRG2) та комбінаційні схеми МКС1 та МКС2. Усю сукупність тригерів, які синхронізуються імпульсами СІ, можна розглядати як один мікроелемент з тригерами-засувками MRC1. Усі тригери, що синхронізуються С2, розглядаються як макрорегістр MRG2.

Причому, група MRG1 може містити у собі множину самостійних мікросхем, що мають свої власні назви та не мають між собою сенсового зв’язку. Те же саме стосується і групи MRG2.

Усі комбінаційні схеми КС підключені до *D*-входів тригерів, кожна група МКС1 та МКС2 синхронізуються також різними фазами. Реалізовані мікросхемами МКС1 та МКС2 функції значень не мають. Необхідно лише виконувати правила підключення їх до регістрів MRC1 та MRG2.

Схема, що використовує двофазну синхронізацію, працює наступним чином (рис. 2.7).

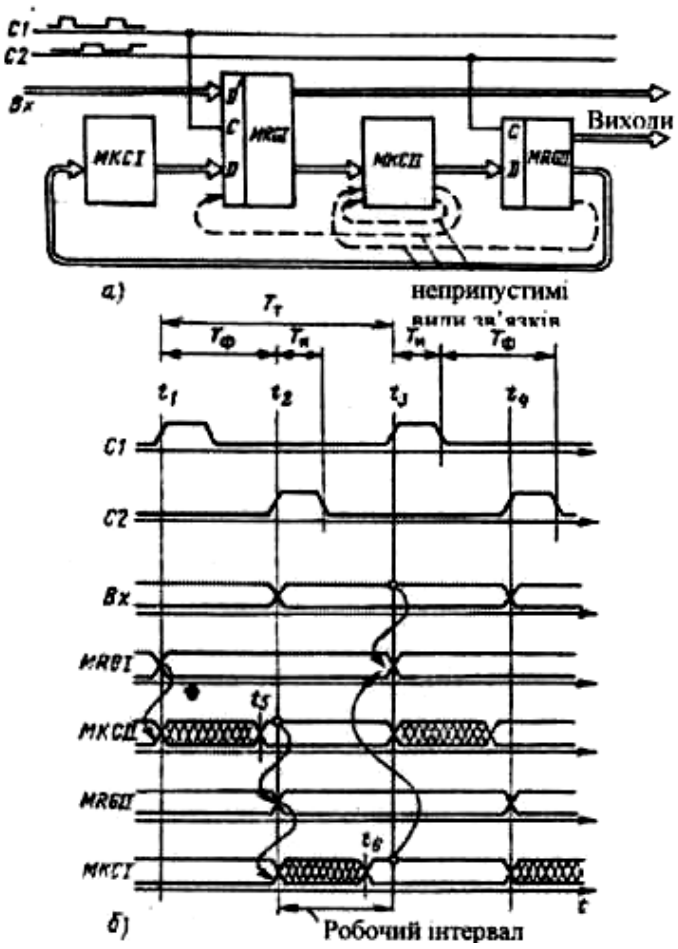


Рис. 2.7. Схема (а) та часова діаграма (б) системи двофазної синхронізації

Фронтом деякого синхроімпульсу С, у момент t_1 вхідні кон'юнктори засувок регістра MRG1 відкриваються і тригери змінюють стан своїх виходів.

Вихідні рівні MRG1 розпочинають відпрацьовуватися комбінаційною схемою МКС2. У схемі існують паралельні шляхи, і виходи її початку спотворені гоночними процесами, показаними на рис. 2.7,*б* у вигляді сітки накладених один на одного фронтів. Для вмісту MRG2 помилкові стани виходів МКС2 не є небезпечними, оскільки його входи замкнені нульовим рівнем синхросигналу C_2. З плином часу, рівного максимально можливій затримку МКС2 (затримки найдовшого її тракту), до моменту t_2 усі перехідні процеси в МКС2 напевно затухнуть і на її виходах встановляться стабільні рівні. Нехай затримка МКС2 менша, ніж тривалість фазового періоду синхронізації $T_ф$, тобто уся МКС2 заспокоїться до моменту t_5, ще до надходження фронту чергового синхросигналу C_2.

На фронті C_2 у момент усталеного значення виходів МКС2 приймаються у тригери MRG2, і з цього моменту можна розпочинати відлік часу перехідних процесів уже в МКС1. Нехай затримка схем МКС1 така, що вони заспокоюються до моменту t_6, до надходження фронту чергового сигналу C_1. Тоді по фронту C_1 е момент t_9 усталені, неспотворені гонками рівні МКС1 будуть прийняті у MRG1. Одночасно у MRG1 приймаються усталені рівні

вхідних сигналів B_x, оскільки за умовою змінюватися вони можуть лише по фронту C_2.

Вміст MRG1 знову обробляється на МКС2, приймається в MRG2 і т. д. У пристрої йде циклічне багатоступеневе оброблення вхідних даних, у кожен момент часу частина комбінаційних схем працює, в них йдуть перехідні процеси, а інша частина схем знаходиться у спокої, чекає своєї черги. Потім вони міняються ролями. Основний результат такої організації: незважаючи на будь-які гоночні процеси, що протікають у будь-яких комбінаційних схемах, інформація у регістри буде прийматися завжди вірна. Для цього потрібно лише, щоб затримка усіх КС, що входять до складу МКС, була якось обмежена зверху, а це розробник схем цілком може забезпечити, спираючись на паспортні значення максимальних затримок елементів.

Важливий момент принципу двофазної синхронізації при використанні прозорих засувок: синхросигнали C_1 і C_2 не повинні взаємно перекриватися у часі, тобто кон'юнкція їх завжди повинна бути рівною 0. Якщо десь станеться перекриття синхросигналів, то інформація пройде послідовно відразу крізь кілька засувок різних фаз і синхронність пристрою буде порушена.

Однак, слід врахувати, що вимагається при розробці схемних реалізацій алгоритмів, що мають цикли, не припуститися помилки при заведенні зворотних зв’язків. В усіх схемах з двофазною синхронізацією петля зворотного зв’язку, як у схемах, що містить логічні елементи, так і у вигляді просто провідника, розпочавшись на виході тригера-засувки синхронізованого одною фазою, повинна закінчитися на вході другої засувки, яка обов’язково синхронізується другою фазою.

Зв’язки, що передають сигнал з виходу однієї засувки на вхід іншої, неприпустимі, якщо засувки синхронізуються однією фазою. Необхідно забезпечити по черговість роботи тригерів у потактовому просуванні інформації.

На вибір часових характеристик синхросигналів $T_ф$ та $T_і$ впливають три групи чинників.

1. Затримки комбінаційних схем, що використовуються у розробках.

2. Тип використовуваних мікросхем послідовнісних пристроїв, тобто використовуваних у них синхронних тригерів.

3. Схеми розв’язання розподілу синхросигналів за блоками пристрою.

Ще краще час тактового періоду використовується у багатофазних системах синхронізації, які отримали широке поширення у швидкодіючих пристроях.

На рис. 2.8,*а* показано, як П-розрядне слово, яке надходить на вхід і несе деякий смисловий зміст, послідовного, крок за кроком, опрацьовується ланцюгом КСМ, КСN, КСР, передаючись з однієї КС в іншу через відповідні регістри.

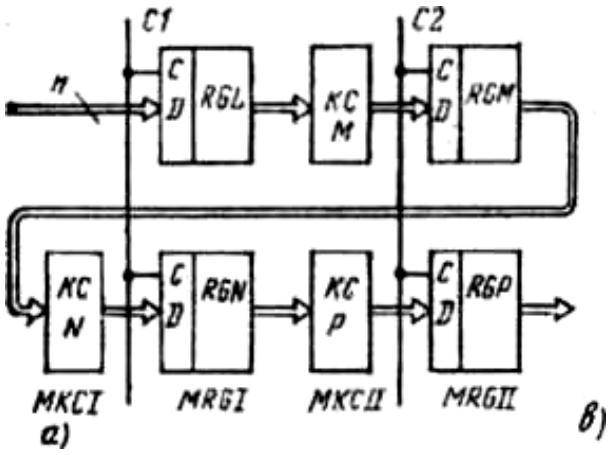


Рис. 2.8. Тракт багатоступеневого оброблення інформації

Якщо перехідний процес у деяких функціональних вузлах в одній з фаз виявився довшим $T_ф$ на Δt, то наступні функціональні вузли, які продовжують переробляти цю інформацію, повинні мати затримку меншу $T_ф$, так щоб за кілька наступних фаз (або всього за одну) вибіг затримки Δt був погашений і перехідні процеси у КС знову стали закінчуватися до моменту надходження фронту чергового С-сигналу. Таким чином, робочий інтервал РІ, протягом якого у схемах, що належать до даної фази, припустимі перехідні процеси, *у* разі двофазної синхронізації і прозорих засувок характеризується двома значеннями: максимальним, $РІ_{засув.макс}$, що дорівнює сумі фазового періоду $T_ф$ і тривалості імпульсу $T_і$, і середнім $РІ_{засув.сер}$, рівним $T_ф$. Якщо затримку якійсь КС довелося зробити більшою, ніж $T_ф$, то протягом кількох фазових періодів після цього середня затримка у даному інформаційному тракті повинна бути меншою $T_ф$.

Щоб за необхідності розробник міг зробити вибіг Δt максимальним, тривалість імпульсу T, потрібно вибирати якомога ближчим до її верхньої припустимої межі, яка обмежена небезпекою накладення C_1 і C_2.

Можливість мати затримку КС, що помітно перевищує фазовий період, – зручна для розробника властивість системи синхронізації. Властивість ця відома не широко, і у більшості розробок вона лише делікатно підправляє помилки схемотехніки й технологів, коли реальна затримка якоїсь КС виявляється більшою розрахункової. Однак кваліфіковані розробники можуть використовувати цю властивість цілком усвідомлено. У багатофазних системах синхронізації за тієї ж самої тривалості такту $T_т$ тривалість імпульсу $T_і$ значно менша, ніж у двофазних, відповідно менший і ефект подовження максимально можливого робочого інтервалу.

Схеми розподілу синхросигналів доводиться будувати у зв'язку з дуже великим числом вузлів-споживачів цих сигналів у цифровому пристрої. Розводити синхросигнали від єдиного потужного генератора не слід, бо потужні кола породжують такі ж самі потужні перешкоди. Тому систему синхронізації будують у вигляді багатоярусного дерева зі звичайних елементів, яка розмножує сигнали малопотужного задавального генератора. При цьому

яруси дерева часто поєднують з ярусами конструктивного поділу пристрою на плати.

2.13.2. Однофазна синхронізація

Схема інформаційних зв'язків окремої частини мікропроцесорного комплекту з однофазною синхронізацією надано на рис. 2.9.

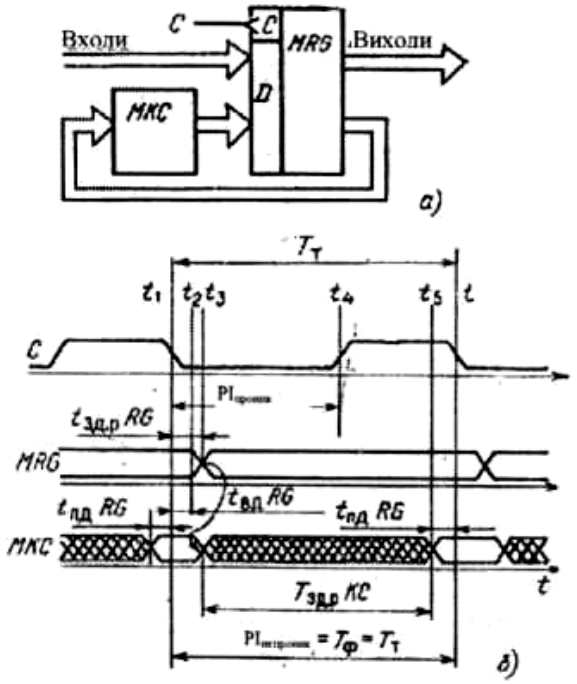


Рис. 2.9. Схема *(а)* та часова діаграма (б) системи однофазної синхронізації

Як і на рис. 2.7 усі комбінаційні схеми пристрою, що отримують дані з виходів тригерів та передають результати на £>-входи тригерів, розглядаються сумісно, як велика комбінаційна мікросхема МКС, у даному випадку одна. Усі послідовні сні пристрої синхронізуються одним і тим самим синхросигналом і розглядаються як великий макрорегістр MRG1. На деякі тригери цього регістра надходять зовнішні сигнали, з деяких тригерів знімаються вихідні сигнали.

Як випливає з часової діаграми (рис. 2.9,*б*) за зрізом С-імпульса у момент t^1 виходи усіх тригерів перемикаються у ті стани, які були на їхні входах безпосередньо перед надходженням зрізу С-імпульсу, тобто у стани, вироблені до цього моменту логічними схемами МКС. Після зрізу синхроімпульса інформація з виходу МКС виявляється переданою на її вхід, і в МКС розпочинаються перехідні процеси чергового циклу. У грамотно спроектованому пристрої усі перехідні процеси повинні закінчитися до зрізу наступного синхроімпульса, а точніше, до моменту t_5 (початку інтервалу підготовки $t_{під}$). Сталі, не спотворені гонками, стани виходів МКС знову будуть передані на вхід тієї ж самої МКС і т.д.

Як видно з рис. 2.9,*б,* робочий інтервал за однофазної синхронізації дорівнює повному періоду синхросерії $T_т$, але тільки якщо тригери не мають

властивості проникності. В іншому випадку перехідні процеси в МКС повинні закінчитися до надходження фронту С-сигналу, тобто до моменту t_4. Інтервал $t_4 - t_5$ при цьому для справи пропадає. Якщо схема побудована на базі тригерів, що перемикаються позитивним фронтом, то робочий інтервал буде займати положення між фронтами С-сигналів. При використанні в одному пристрої тригерів з різними положеннями робочих інтервалів максимально допустима затримка узгоджувальних їх КС буде відрізнятися від номінальної – так само, як і у разі двофазної синхронізації.

Тонким моментом, який помітно впливає на надійність роботи однофазних схем, є співвідношення затримки поширення $t_{зт.р}$ і часу витримки тригерів регістра. Якщо затримка комбінаційної схеми мала, особливо якщо КС є просто провід, до того ж короткий, то новий стан виходу тригера, що швидко переключився, може надійти на вхід іншого тригера занадто рано для останнього, ще протягом його інтервалу витримки, коли рівні його керуючих входів змінювати ще не можна. Для гарантії відсутності збоїв такого типу потрібно, щоб максимальне значення $t_{вт}$ будь-якого тригера не перевищувало мінімального значення $t_{зт.р}$ також будь-якого тригера, тобто момент часу t_2 на рис. 2.9,*б* повинен обов'язково передувати моменту t_3. У цьому сенсі поза конкуренцією виявляються схеми тригерів, у яких $t_{вт} = 0$.

За сумнівного співвідношення $t_{зт.р}$ і $t_{вт}$ доводиться обмежувати мінімальне значення затримки КС, наприклад виключаючи прямі зв'язки тригер – тригер за рахунок уведення між ними одного-двох холостих логічних елементів. Такі обмеження зустрічаються, наприклад, в інструкціях з проектування схем на деяких матричних ВІС. Вимоги до решти часових параметрів тригерів при використанні їх у системі однофазної синхронізації такі ж самі, що й у двофазної.

Однофазна система синхронізації на відміну від двофазної болісно чутлива до розбіжності активних фронтів синхроімпульсів, що виникає в системі розподілу синхросигналів.

Ситуація повністю аналогічна розбіжності у датуванні астрономічного явища, яке може спостерігатися на значній території і сталося близько опівночі за уральським часом: московський і новосибірський спостерігачі датують цю подію різними числами.

Таким чином, однофазна синхронізація не терпить, щоб інформація обганяла синхросигнал. Здебільшого у цьому криється причина обмеженого застосування однофазної синхронізації. На противагу їй двофазна синхронізація легко переносить будь-які затримки у трактах розведення С-сигналів, для цього достатньо лише у потрібній мірі збільшити тривалість тактового періоду задавального генератора. Регістри за двофазної синхронізації будують на основі найпростішого із синхронних тригерів, але за необхідності можуть використовуватися тригери майже усіх типів. Ніяких особливих вимог до значень або співвідношень їх часів затримки, витримки, підготовки не пред'являється.

Однак це ускладнює схему розведення синхросигналів і процес проектування часової діаграми пристрою через ускладнення завдань фазування процесів у різних КС.

Тому, незважаючи на те, що в однофазних системах розводити потрібно усього одну синхросерії, і з розробника, який складає часову діаграму, зняті усі турботи про фазування різних вузлів, однофазную синхронізацію широко застосовують при побудові лише невеликих окремих вузлів - лічильників, зсувних регістрів і т. п. Такі однофазні вкраплення у великий пристрій, який в цілому має двофазну або багатофазну синхронізацію, зручні тим, що дозволяють за один фазовий період, а іноді лише за час активного рівня С-сигналу виконати якусь дію, пов'язану із заміною вмісту деякого регістра (зсув у регістрі, додавання одиниці до лічильника і т. п.), на яку за класичної двофазної синхронізації потрібні два фазові періоди. При уведенні таких однофазних вкраплень потрібно пам'ятати про можливість розбіжності їхніх робочих інтервалів з робочими інтервалами засувок і про пов'язану з цим необхідність зменшити затримки приграничних КС.

Нерідко однофазну синхронізацію використовують у мікропроцесорних наборах або комплектах, де не вимагається багаторазового ступінчатого розмноження синхросигналів. У більшості пристроїв використовують двофазну та багатофазну синхронізацію.

Контрольні питання

2.1. Накресліть структурну схему перетворення інформації в мікропроцесорній системі.
2.2. Що таке алгоритм?
2.3. Назвіть найбільш важливі властивості алгоритмів.
2.4. Що таке клерувальні слова?
2.5. Що таке ЦП (центральний процесор)? Його склад.
2.6. Що таке пам'ять: внутрішня і зовнішня?
2.7. Для чого використовується регістр команд?
2.8. Що таке лічильник команд?
2.9. Що таке арифметико-логічний пристрій (АЛП)?
2.10. Що таке система команд мікропроцесора?
2.11. Система команд однакова для різних мікропроцесорів?
2.12. Назвіть команди пересилання.
2.13. Що таке команда з безпосередньою адресацією?
2.14. Які команди належать до арифметичних і логічних?
2.15. Що являє собою перший етап виконання команди?
2.16. Які елементи входять у мікропроцесорний комплект?
2.17. Навіщо використовується синхронізація у МК?
2.18. Які види синхронізації ви знаєте?
2.19. Що таке однофазна синхронізація?
2.20. Що таке двофазна синхронізація?
2.21. Назвіть переваги однофазної синхронізації.
2.22. Назвіть переваги двофазної синхронізації.

3. ПРОГРАМУВАННЯ ДЛЯ МІКРОПРОЦЕСОРІВ

3.1. Загальні відомості

Для оброблення інформації на мікропроцесорному комплекті або мікрокомп'ютері необхідно крок за кроком описати увесь процес оброблення. Такий опис називається програмою. Реальна програма, яка виконується мікрокомп'ютером, повинна складатися лише з тих команд, які передбачені його конструкцією. Сукупність цих команд складає машинну мову або мову машинних команд комп'ютера.

Команди, які виконуються апаратурою мікропроцесора, прості та небагато чисельні. Проте, обчислювальні можливості мікропроцесора практично не обмежені, оскільки ці команди можна поєднувати у програми. Сукупність програм, написаних для мікропроцесора, складає його програмне забезпечення. Проблемно-орієнтованих мов програмування досить багато. Писати програми на цих мовах зручніше, ніж на машинній мові. Проте, програму на проблемно-орієнтованій мові необхідно перед використанням транслювати на машинну мову. Існують такі засоби програмування, як асемблері та компілятори. Асемблери виконують трансляцію мови програм, написаних у символічній формі, яка близька до машинної мови.

Компілятори – це транслятори для проблемно-орієнтованих мов, або мов вищого рівня.

3.2. Програмування на машинній мові

Програма повинна бути подана у тій формі, в якій її сприймає мікропроцесорний комплект або машина. Зокрема, команди повинні бути подані у вигляді послідовності одиниць та нулів, оскільки це єдина зрозуміла апаратурі форма. Проте складати таку послідовність дуже довго і втомливо.

При запису команд застосовується більш зручна шістнадцяткова система. Як відомо, між шістнадцятковою та двійковою системами перетворення виконуються дуже легко.

Тим не менш і двійкове, і шістнадцяткове подання машинної мови незручне та не наочне для людини. Таке програмування мікропроцесорів застосовується вкрай рідко.

Навіть не дуже складні задачі вимагають для свого розв'язання достатньо великих програм, а їх складання – справа праце ємна ті кропітка.

Тому створюються спеціальні засоби, які полегшують підготовку програми на машинній мові.

3.3. Асемблери

Складати програми більш зручно, якщо поряд із символічними іменами для команд, які називаються мнемонічними кодами операцій, можна

використовувати для позначення адрес у командах також символічні імена, а не числові значення.

Наприклад, щоб завантажити вміст комірки пам’яті (у шістнадцятковому поданні) 5381_{16} в акумулятор, необхідно виконати команду з трьох байтів 705386.

Звичайно ж, більш зручний та наочний запис цієї команди у вигляді

LDR O, X

де LDR – мнемонічний код операції;

O – позначення загального регістра;

X – символічне позначення комірки пам’яті 5386.

Ми можемо отримати ці зручності, якщо будемо мати у своєму розпорядженні деякі засоби, що підставляють числові значення замість символічних імен (операцій та адрес) після того, як програма вже написана. Це й становить основну функцію *програми-асемблера*. Програма, написана у новій зручній формі, називається *програмою на мові асемблера*, а кожна команда називається *пропозицією на мові асемблера* або *оператором на мові асемблера*.

Взагалі кажучи, програма на мові асемблера – це, по суті, та ж сама програма на машинній мові з символічними іменами для адрес операндів, символічними іменами для адрес команд та мнемонічними кодами операцій. Завдання асемблера – замінити кожен мнемонічний код операції відповідним машинним кодом операції, відвести комірки для кожної команди і кожного операнда та підставити замість символічних адрес їх числові значення. Як правило, між пропозиціями на мові асемблера та командами на машинній мові існує взаємооднозначна відповідність. Однак бувають випадки, коли за однією пропозицією асемблер генерує кілька машинних команд.

На рис. 3.1 показано схему використання асемблера. Програма на мові асемблера називається *вихідною програмою*, а згенерована програма на машинній мові – *об'єктною програмою*.

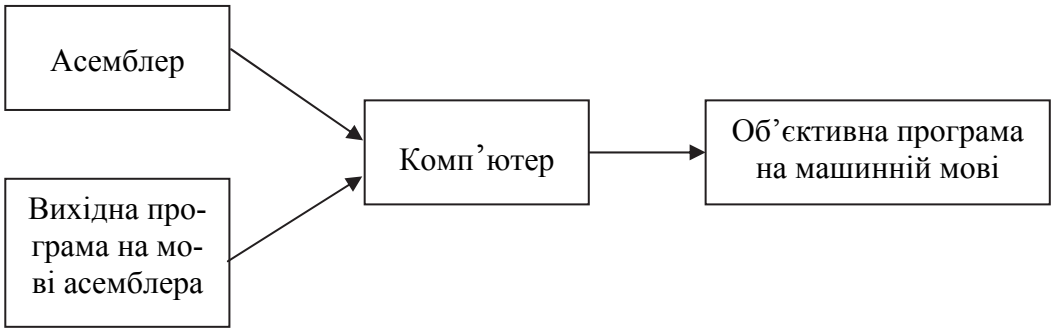


Рис. 3.1. Схема використання асемблера

Асемблер, який виконує функції транслятора і який сам є програмою, завантажується у комп'ютер разом з вихідною програмою. Вихідна програма виступає у ролі «даних», оброблюваних програмою-асамблером. В результаті генерується об'єктна програма. Коли асемблер працює на тому ж самому комп'ютері, що й об'єктна програма, він називається *власним асемблером*.

Проте часто асемблер буває написаним для більшої програми, яка виконує процес трансляції. Такий асемблер називається *крос-асемблером*.

В процесі трансляції асемблер використовує таблиці для присвоювання числових значень мнемонічним кодами операцій і символічним адресам. В асемблер завжди включається таблиця з усіма допустимими мнемонічними кодами операцій і відповідними їм числовими значеннями. Так що заміна операцій виконується просто. Для того, щоб присвоїти числове значення кожній символічній адресі, асемблер спочатку складає так звану *таблицю імен*, до якої заносить усі зустрінуті у програмі символічні адреси. Потім він відводить область пам'яті для програми, обчислює значення для усіх символічних адрес і після цього виконує відповідну підстановку.

Однією з головних переваг мови асемблера є легкість, з якою робляться вставки й видалення пропозицій у вихідній програмі. При написанні програми часто виявляється, що у деяку точку вже написаної частини програми потрібно вставити групу команд. У програмі на машинній мові це вимагає зсуву усіх слів програми нижче вставлених команд. Більш того, може бути, що адресні поля деяких команд у програмі, (якщо, приклад, є цикли або переходи) тепер повинні посилатися на команди, які перемістилися. Це означає, що усі такі адресні поля треба відкоригувати.

У програмі на мові асемблера подібних проблем не виникає, оскільки адресам не даються числові значення. Отже, при вставках пропозицій ніяких труднощів не виникає. Точно так само труднощів не виникає і при викиданні пропозицій.

Інша корисна властивість багатьох асемблерів – це виявлення помилок. Звичайно, асемблер не може виявити логічних помилок у самому задумі програми; проте він може перевіряти дотримання певних синтаксичних обмежень, накладених на мову, і повідомити програмісту про їх порушення.

3.4. Компілятор

Існують транслятори іншого класу, звані *компіляторами*. За їх допомогою, наприклад, транслюються такі мови, як Фортран, ПЛ/1, Паскаль та ін. Мови, що транслюються компіляторами, часто називають *мовами високого рівня*. Запис алгоритмічного процесу на цих мовах має відносну машинну незалежність. Це означає, що програмісту не обов'язково знати особливості конкретного комп'ютера, на якому буде виконуватися програма, потрібно лише, щоб машина мала відповідний компілятор.

При трансляції програми з мови високого рівня на мову машинних команд, взагалі кажучи, кожне речення вихідної мови породжує кілька машинних команд, тоді як для асемблерів в основному дотримується співвідношення «один до одного». Тому програми, написані на мовах високого рівня, бувають значно компактнішими.

У порівнянні з мовою асемблера мови високого рівня мають більшу потужність і надають програмісту більші зручності. Проте трансляція з мов

високого рівня суттєво складніша, а компілятори – це великі й складні програми, що вимагають для своєї роботи великої пам'яті. Більш того, у програмі на мові високого рівня не можна скористатися усіма властивими машинній мові тонкощами; наприклад, на них неможлива пряма робота з регістрами. Саме ці міркування перешкоджають використанню мов високого рівня у деяких додатках мікропроцесорів, особливо для управління процесами у реальному масштабі часу.

3.5. Підпрограми

Часто ми стикаємося з необхідністю вирішувати одну і ту ж саму задачу у кількох місцях програми. У цьому випадку можна написати підпрограму, що вирішує це завдання, і звертатися до неї у міру необхідності, не повторюючи одні і ті ж самі команди у різних точках програми. Таким чином, підпрограма – це допоміжна програма, якою користується головна програма. Оскільки підпрограма має певну самостійність, вона зазвичай розміщується в окремій від головної програми області пам'яті.

3.5.1. Вхід у підпрограму та вихід з підпрограми

У самому понятті підпрограми припускається, що її можна викликати (тобто звертатися до неї) у різних точках головної програми з наступним поверненням на продовження головної програми. Тому повинні існувати засоби для збереження адреси тієї точки у головній програмі, в яку потрібно повернутися. Існують різні методи розв’язання цієї проблеми. Один з них полягає у використанні стека, вбудованого у самому мікропроцесорі або модельованого у головній пам’яті.

По команді переходу на підпрограму вміст програмного лічильника, який на цей момент вказує на наступну команду у головній програмі, запам’ятовується у стеці. Потім ця адреса знову завантажується у програмний лічильник, коли в кінці підпрограми виконується команда «повернення з підпрограми».

Інший підхід полягає в тому, що адреса повернення зберігається в області пам'яті, що містить саму підпрограму. Ще один метод, застосовуваний у деяких системах, передбачає збереження повернення в якомусь спеціально обумовленому регістрі центрального процесора.

Взагалі кажучи, бажано, щоб підпрограма у свою чергу могла звернутися до іншої підпрограми. Такі звернення називаються *вкладеними*. Три згаданих вище методи надають різні можливості для організації вкладених звернень. При використанні стека допускається найзагальніша форма вкладених звернень. Стек дозволяє одній підпрограмі звертатися не лише до інших підпрограм, а й до самої себе. Це можливо, оскільки при кожному вході у підпрограму адрес повернення поміщається у вершину стека, а усі раніше запам’ятовані адреси опускаються у стек і тим самим зберігаються. При поверненні запам’ятовані адреси вибираються зі стека у порядку, зворотному тому, в якому вони

запам'ятовувалися. Це відповідає порядку, в якому повинні виконуватися повернення. Звичайно, число рівнів вкладеності (тобто число незавершених підпрограм) обмежується розмірами стека, оскільки кожна адреса повернення займає один регістр у стеці.

Кількість регістрів стека визначає кількість рівні вкладень.

Метод збереження адреси повернення у самій підпрограмі дозволяє підпрограмі звернутися до іншої підпрограми, але не до самої себе. Дійсно, у ході виконання підпрограми комірка, де зберігається адреса повернення, вже зайнята і не може прийняти ще одну адресу. Однак цей метод не накладає ніяких обмежень на число рівнів вкладень, оскільки кожна підпрограма на кожному рівні надає місце для адреси повернення. До числа недоліків цього методу потрібно віднести те, що підпрограму не можна помістити у постійний ЗП, оскільки в комірку з адресою повернення повинен уводитися запис. Для мікропроцесорів, де часто використовуються постійні програми, ця остання обставина може виявитися вирішальною при виборі методу.

При збереженні адреси повернення в одному фіксованому регістрі вкладеності підпрограм взагалі неможливі, оскільки в одному регістрі можна зберегти адресу лише однієї підпрограми. Однак ця проблема може бути вирішена програмно шляхом додавання у підпрограму команд, які зберігали вміст регістра в пам'яті перед зверненням до іншої підпрограми. Комірку для збереження можна обрати в області пам'яті, відмінній від тієї, яку займає сама підпрограма, тому підпрограми можна розміщувати у постійній пам'яті. Повернення з підпрограми можна здійснювати, передаючи управління на комірку із запам’ятованою адресою. Щодо рівнів вкладеності така схема еквівалентна попередній у тому сенсі, що підпрограма може викликати інші підпрограми, але не себе.

У мікропроцесорах застосовуються усі три описані методи, але домінує метод з використанням стека. Найчастіше стек реалізується не на апаратних регістрах у процесорі, а *моделюється* в комірках головної пам'яті.

При цьому передбачається спеціальний регістр, званий *покажчиком стека*. Покажчик стека містить адресу комірки, яка вважається вершиною стека у поточний момент. На початку головної програми у покажчику стека встановлюється деяке початкове значення. Коли інформація заноситься у стек або видаляється з нього, покажчик автоматично просувається вгору або вниз і, таким чином, увесь час вказує на вершину стека. Стек розширюється при надходженні нової інформації і стискається при її видаленні. У модельованому стеці практично не виникає обмежень на число рівнів вкладеності.

3.5.2. Параметри підпрограми

Важливий аспект поняття підпрограми пов'язаний зі способом передавання даних, які називаються *параметрами* або *аргументами,* між головною програмою і підпрограмою. Існує кілька способів вирішення цієї проблеми. По-перше, аргументи можна завантажувати у загальні регістри перед переходом на підпрограму. Підпрограма, написана у припущенні, що аргументи знаходяться у регістрах, виконується і залишає свої результати також в регістрах. Головна

програма після повернення з підпрограми бере результати із загальних регістрів.

Ілюструє такий підхід програма в табл. 3.1. Передбачається, що аргументи знаходилися у комірках пам'яті 8000 і 8050. Головна програма завантажує їх у загальні регістри D і E, після чого передає управління на підпрограму у комірку 9000 за допомогою команди «перехід на підпрограму».

Таблиця 3.1.

Приклад передавання параметрів від головної програми у підпрограму через загальні регістри

Комірка пам'яті	Команда на машинній мові	Команда в символічній формі	коментар
0020	70	LDR 0	Завантаженість першого аргументу в регістр D
0021	80	80	
0022	00	00	
0023	0D	MOV 0 to D	
0024	70	LDR 0	Завантаження другого аргументу в регістр E
0025	80	80	
0026	50	50	
0027	0E	MOV 0 to E	
0028	7F	JMS	Перехід на підпрограму у комірку 9000
0029	90	90	
002A	00	00	
002B	-	-	Точка повернення з підпрограми.

Якщо потрібні підпрограмі аргументи знаходяться у послідовних комірках пам'яті, у головній програмі перед переходом на підпрограму можна завантажити в регістри H і L (або у будь-яку іншу пару загальних регістрів) адресу першого аргументу. Відповідальність за вибірку аргументів покладається при цьому на підпрограму. Перевага цього способу у тому, що не потрібно завантажувати відразу усі аргументи в регістри – програма буде вибирати з їх пам'яті у міру необхідності. Такий підхід, очевидно, кращий, коли число аргументів перевищує число вільних загальних регістрів. Крім того, цей підхід вирішує проблему зворотної передачі результатів. Наприклад, достатньо під результати зарезервувати частину списку аргументів.

У розглянутому випадку головна програма завантажувала в регістри H і L адресу першого аргументу і тим самим передавала у підпрограму список аргументів. Можливий варіант, коли головна програма передає у підпрограму не список самих аргументів, а список адрес аргументів. У цьому випадку в регістри H і L завантажується адреса першої адреси у списку адрес аргументів. Таким чином, щоб отримати аргумент, підпрограма повинна спочатку отримати адресу зі списку адрес, а вже потім сам аргумент. Цей спосіб має ту перевагу,

що аргументи не обов'язково повинні розташовуватися у послідовних комірках пам'яті, а можуть бути розкидані довільним чином.

3.6. Завантаження програм

Важливу роль у мікропроцесорних комплектах відіграють засоби завантаження виконуваних програм у пам'ять. У багатьох додатках, де мікропроцесори виконують лише деякі раз і назавжди встановлені функції, програми зберігаються у постійній пам'яті. Однак у додатках, що вимагають більшої гнучкості і різноманітності, програми завантажуються в оперативну пам'ять у міру необхідності. Зазвичай це робить спеціальна програма-завантажувач, яка виконує операції по введенню необхідної програми з деякого пристрою введення і розміщення її у пам'яті. Сам завантажувач може або розміщуватися у постійній пам'яті, або вводитися в оперативну пам'ять покомандно вручну з пульта управління.

Оскільки завантажувач використовується для введення у пам'ять самих різних програм і, можливо, даних, йому повинна бути повідомлена певна інформація. У типових випадках потрібні такі відомості:

1. Початкова і кінцева адреса завантажуваної області пам'яті.
2. Початкова адреса завантажуваної області пам'яті і число слів, що уводяться.
3. Початкова адреса завантажуваної області пам'яті і вид обмежувача (ознаки кінця), що вводиться.

Крім уведення програми, на завантажувач можуть бути, взагалі кажучи, покладено додаткові функції. Наприклад, переміщення програми у пам'яті. Виконання такої функції вимагає від завантажувача зміни посилань на комірки пам'яті у командах таким чином, щоб вони відповідали новому положенню програми. Інша поширена функція завантажувачів – це виявлення помилок, наприклад помилок, що виникають при передаванні інформації між пристроями уведення і пам'яттю.

Розглянемо найпростіший завантажувач для мікропроцесора.

Будемо вважати, що завантажувана інформація (програма або дані) надходять з деякого пристрою уведення, яке отримує цю інформацію слово за словом з деякого носія і робить її доступною для мікропроцесора через деякий порт уведення.

Вважаємо, що слова мають довжину 8 біт (аналогічно слову у мікропроцесорі). Перші 2 слова на носії містять 16-бітову адресу (старші розряди у першому слові) та вказують завантажувачу початкову комірку пам’яті, куди має бути завантажена інформація. Наступні два слова визначають кінцеву адрес області пам'яті. Далі слідує інформація яка повинна бути завантажена в цю область.

Отже, завантажувач отримує інформацію послівно від пристрою уведення. Перші чотири слова – це, як було сказано вище, адресна інформація, а решта – завантажувана. Пристрій уведення в силу своєї механічної природи, як правило, працює значно повільніше мікропроцесора, що обробляє слова, які уводяться.

Тому треба запрограмувати мікропроцесор так, щоб він очікував, коли чергове слово, що уводиться, стане доступним. Тому для передачі інформації про стан пристрою уведення ми використовуємо другий порт. У цей порт пристрій уведення встановлює у деякий (скажімо, крайній, лівий) біт кожного разу, коли готове чергове слово, що уводиться. Мікропроцесор, перш, ніж уводити дані, повинен перевірити цей біт і за необхідності чекати, допоки він не стане рівним одиниці. У момент, коли мікропроцесор дійсно уводить слово, цей біт стану автоматично зовнішніми схемами скидається у 0, так, щоб одне й те ж саме слово не було уведено двічі.

Робота програми-завантажувача зображена у вигляді блок-схеми на рис. 3.2. У кількох точках головна процедура звертається до підпрограми уведення. Ця підпрограма спочатку уводить дані про стан пристрою уведення. За яким визначається готовність чергового слова. При готовності уводиться чергове слово (у наступному блоці схеми), при неготовності повторюється опитування стану пристрою уведення. Таким чином, мікропроцесор зациклюється в очікуванні готовності слова.

Головна процедура чотири рази звертається до підпрограми уведення для уведення перших чотирьох адресних слів, передаючи ці слова кожного разу у нові загальні регістри. Потім розпочинається цикл передачі у пам'ять слів, що уводяться. Кожне слово уводиться підпрограмою уведення і потім передається у відповідну комірку пам'яті, адреса якої задається поточним значенням адреси (ПА). ПА збільшується на 1 при кожному проходженні циклу, тобто під час надходження кожного слова. У кінці циклу відбувається перевірка, чи не перевищує ПА значення кінцевої адреси (КА). Якщо це так, процедура завантаження закінчується; в іншому випадку – цикл повторюється.

Програму-завантажувач наведено у табл. 3.2. Як легко бачити, вона починається у комірці 0000. Проте в реальних умовах вона могла б розташовуватися і в інших комірках, скажімо зі старшими адресами, щоб не заважати розміщенню інших програм. Програма звертається до двох портів уведення: до порту 00 – за інформацією про стан та до порту 01 – за даними.

Перша група команд відповідає головній процедурі. Підпрограма уведення починається з комірки 001D. Загальні регістри 1, 2, 3 і 4 зберігають відповідно $ПА_H$ $АА_L$, $КА_H$ і $КА_L$. Вони завантажуються адресною інформацією, що надходить з пристрою уведення. Для цього кожного разу виконується команда переходу на підпрограму і команда пересилання.

Цикл, показаний на блок-схемі, розпочинається у комірці 0010 командою переходу на підпрограму уведення. Після повернення з неї чергове уведене слово знаходиться в акумуляторі. Це слово передається у комірку пам'яті за допомогою наявного в ілюстративному мікропроцесорі механізму непрямої адресації. А саме, команда MOV 0 to F передає вміст акумулятора у комірку пам'яті, адреса якої задана вмістом регістрів 1 і 2 (тобто H і L). Оскільки ці регістри містять значення ПА, слово потрапляє у потрібне місце пам’яті. Наступна команда, IHL, збільшує на 1 значення ПА.

Порівняння адрес ПА і КА здійснюється за допомогою процедури віднімання з подвійною точністю і подальшого умовного переходу. Для

віднімання з подвійною точністю спочатку віднімаються молодші частини ПА і КА, а потім старші частини за участю позики, що залишилася від молодших частин. Остаточна позика залишається у тригері. Наступна команда – «перехід при нульовому перенесенні» на початок циклу. За C = 0 знову входимо у цикл, за C = 1 програма зупиняється.

Умовні позначення:

ПА – поточна адреса; КА – кінцева адреса; Слова – уведення слова;

$( )_H$ – старша частина; $( )_L$ – молодша частина; M(x) – комірка пам'яті з адресою x

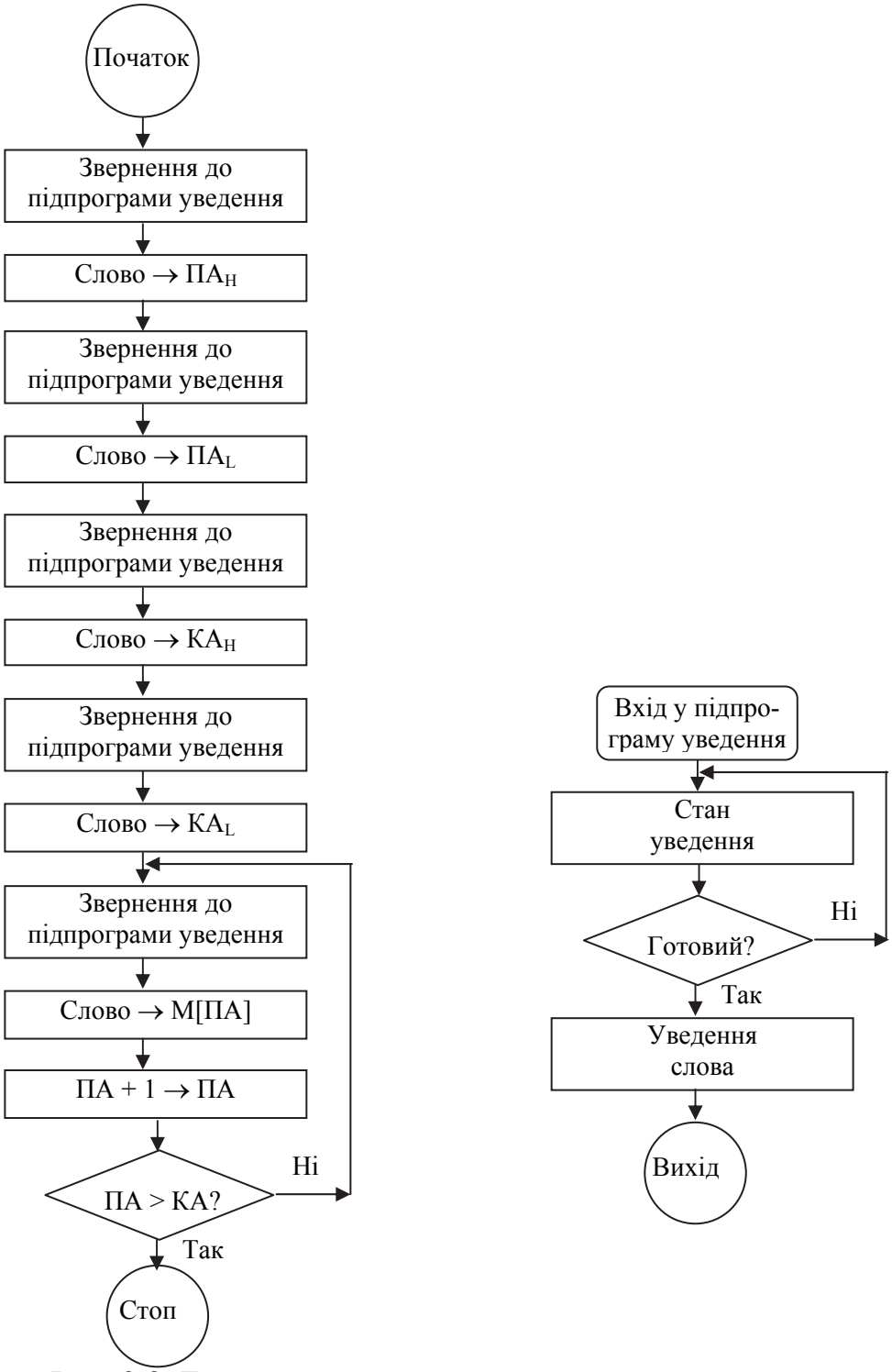


Рис. 3.2. Блок-схема програми-завантажувача

Підпрограма уведення розпочинається з уведення слів з порту стану 00 в акумулятор. Лівий знаковий біт тестується командою «перехід при позитивного акумулятора».

Таблиця 3.2

Програма-завантажувач

Розподіл загальних регістрів: R_1: $ПА_Н$, R_2: $КА_Н$, R_3: $ПА_L$, R_4: $КА_L$

Комірка пам'яті	Команда на машинном мову	Команда у символічній формі	Коментар
0000	7F	JMS	Звернення до підпрограми введення за $ПА_Н$
0001	00	00	
0002	1D	1D	
0003	01	MOV 0 to 1	
0004	7F	JMS	Звернення за $ПА_L$
0005	00	00 3	
0006	1D	1D	
0007	02	MOV 0 to 2	
0008	7F	JMS	Звернення за $КА_Н$
0009	00	00	
000A	1D	ID 1	
000B	03	MOV 0 to 3	
000C	7F	JMS	Звернення за $КА_L$
000D	00	00	
000E	1D	1D	
000F	04	MOV 0 to 4 и	
0010	7F	JMS	Починається цикл введення слів
0011	00	00	
0012	1D	1D	
0013	0F	MOV 0 to F	Передача слова в М [ПА]
0014	F5	IHL	ПА + 1 $\rightarrow$ ПА
0015	14	MOV 0 from 4	Віднімання з подвійною точністю ПА з КА
0016	A2	SUB 2	Встановлюється С = 1, якщо ПА > КА
0017	13	MOV 0 from 3	
0018	B1	SBC 1	
0019	7C	JCZ	Перевірка позики у С. Якщо немає позики, уведення нового слова.
001A	00	00	
001B	10	10	
001C	FA	HLT	Зупинка
001D	FD	INP	Початок підпрограми введення. введення слів стану
001E	00	00	
001F	7A	JAP	Перевірка розряду знака в слові стану.
0020	00	00	Якщо 0, повторити перевірку
0021	ID	1D	
0022	FD	INP	Введення слова в акумулятор з пристроїв уведення
0023	01	01	
0024	F8	RET	Повернення з підпрограми

Якщо вміст акумулятора позитивний (старший біт дорівнює 0), пристрій уведення не готовий. У цьому випадку управління передається на початок підпрограми і процес повторюється. Якщо старший біт в акумуляторі дорівнює 1, наступна команда уводить слово з порту 01 в акумулятор, а команда повернення завершує підпрограму.

Слід звернути увагу на те, що біт готовності у слові стану був обраний спеціально таким, що збігається зі знаковим розрядом акумулятора. Це дозволило виконати тестування біта однією командою. У загальному випадку тестований біт може виявитися у будь-якому іншому розряді. Тоді тестування проводиться шляхом виділення цього розряду. Для цього достатньо логічно помножити слово стану на слово, зване маскою, і яке містить 1 у розряді, що нас цікавить, і нулі – в інших. Після цього результат тестується або командою переходу при нульовому акумуляторі, або командою переходу при ненульовому акумуляторі. Таким чином, команда JAP, яка перевіряє лівий біт слова стану, в нашій програмі можна буде замінити наступними командами

```
LRI   5
80 (маска)
AND 5
JAZ
00
ID
```

3.7. Переривання програми

Найважливіше значення має здатність більшості мікропроцесорів перервати виконувану програму у відповідь на зовнішню подія і виконувати програму, спеціально призначену для оброблення цієї події. Це називається *перериванням програми*. Наприклад, переривною подією може стати завершення очікуваної мікропроцесором дії у зовнішньому пристрої.

Скористатися перериванням за готовністю порту, як правило, набагато зручніше і ефективніше, ніж безперервно опитувати його стан. Мікропроцесор звільняється для виконання інших функцій і, зокрема, може зайнятися іншим портом уведення/виведення.

Переривання програми для більшості комп'ютерів нагадує перехід на підпрограму з тією різницею, що воно ініціюється не командою у програмі, а приходом зовнішнього сигналу по керуючій лінії. Цей сигнал називається *запитом на переривання*. Так само як і підпрограма, програма оброблення переривання розміщується у пам'яті, починаючи з комірки, в яку має передаватися управління.

Знайшовши запит на переривання, процесор відкладає виконання поточної програми і починає виконувати програму переривання. Програма переривання зазвичай закінчується командою повернення, після якої продовжується виконання перерваної програми. Зазвичай процесор має можливість забороняти (блокувати) переривання на деякі відрізки часу, коли їх оброблення з тих чи інших причин незручне. При заблокованих перериваннях запити на

переривання, що надходять, ігноруються.

Звичайно у мікропроцесорних системах запити на переривання можуть надходити від кількох пристроїв. Тому виникає проблема ідентифікації пристрою, який надіслав запит, з тим щоб можна було виконати дії з обслуговування саме цього пристрою. Існує два основні методи вирішення цієї проблеми. Згідно з одним з них, повинна існувати головна програма оброблення переривань, яка під час надходження запиту перевіряє стан кожного пристрою і знаходить пристрій, що вимагає обслуговування. Таку схему часто називають *системою переривань з програмним опитуванням*. За іншого методу інформацію, яка ідентифікує пристрій, який надіслав запит, формує апаратура. Таку систему переривань іноді називають *векторною пріоритетною системою*.

У схемі переривання з програмним опитуванням усі запити на переривання надходять однією керуючою лінією. Кожному пристрою зазвичай виділяється порт стану, а у ньому відводиться один біт, який зберігає запит на переривання. Коли загальною шиною у мікропроцесор надходить запит від будь-якого пристрою; виконання поточної команди завершується, якщо переривання не заблоковане, відбувається передача управління з фіксованою адресою. З цієї комірки розпочинається головна програма оброблення переривань, яка послідовно уводить вміст портів станів і тестує біти запитів на переривання. Виявивши пристрій, який запросив обслуговування, головна програма передає управління програмі, що обслуговує даний пристрій.

Коли запити надходять одночасно від двох або більшої кількості пристроїв, виникають конфліктні ситуації. Їх вирішення закладено у самій схемі опитування. Першим обслуговується той пристрій, який раніше постає у порядку опитування. Звичайно, після обслуговування одного запиту буде обслужений наступний з відкладених запитів і т.д. Таким чином, порядок опитування визначає пріоритетність обслуговування пристроїв.

Головний недолік такої системи переривань пов'язаний з часом, що витрачається програмою на опитування стану окремих пристроїв. Часто ця затримка не відіграє суттєвої ролі, оскільки у багатьох додатках мікропроцесори на відміну від великих машин працюють з нечисленними зовнішніми пристроями. Проте існують додатки, більш чутливі до подібних затримок. Для них краща векторна система переривань, оскільки в ній безпосередньо вказується пристрій, який видав запит.

У векторній системі переривань пристрій, що викликав переривання, ідентифікується за допомогою зовнішніх по відношенню до мікропроцесора схем. Звичайно, ці схеми повинні вирішувати конфлікти, що виникають при одночасному надходженні кількох запитів і надавати обслуговування лише одному пристрою.

Існують різні способи ідентифікації перериваючих пристроїв. Можна, наприклад, мати у процесорі не одну, а більше число ліній для запитів на переривання і кожну з них надати окремому пристрою. У цьому випадку сигнал на кожній конкретній лінії повинен викликати передачу управління у свою комірку пам'яті, якщо тільки ця передача всередині мікропроцесора не

заблокована сигналом переривання з більш високим пріоритетом.

Комірки, в які передається управління, є початковими комірками програм, які обслуговують різні пристрої. Цей спосіб широко застосовують у великих універсальних ЕОМ, але у мікропроцесорах зустрічається рідко. За значного числа пристроїв потрібно багато ліній, а кількість виводів мікропроцесора суттєво обмежена.

Контрольні запитання

1. У чому полягає програмування для мікропроцесорів на машинній мові?
2. Які переваги та недоліки програмування на машинній мові?
3. Що таке асемблер?
4. Накресліть схему використання асемблера.
5. Що таке компілятори?
6. Як виконується завантаження програм у пам'ять?
7. Що таке підпрограма і в якому випадку вона використовується?
8. Як здійснюється вхід у підпрограму?
9. Як здійснюється вихід з підпрограми?
10. Які параметри підпрограми ви знаєте?
11. Що таке переривання програми?
12. Які види переривань програми ви знаєте?
13. Що являє собою переривання програми з програмним опитуванням?
14. Що являє собою векторна система переривань програми?

ЛІТЕРАТУРА

1. Барыбин А.А. Электроника и микроэлектроника : учебн. пособ. / А.А. Барыбин. – М.: Физматлит, 2008. – 424 с.
2. Воронков Э.Н. Твердотельная электроника : учебн. пособ. / Э.Н. Воронков. – М.: Академия, 2018. – 192 с.
3. Галкин В.И. Промышленная электроника и микроэлектроника : учебник / В.И. Галкин. – М.: Высшая школа, 2006.–350 с.
4. Воробйова О.М. Основи схемотехніки: підручник / О.М. Воробйова, В.Д. Іванченко. – Одеса: Фенікс, 2009. – 388 с.
5. Воробьева Е.М. Основы схемотехники: конспект лекций. 4.1 и Ч. 2 / Е.М. Воробьева, В.Д. Иванченко. – Одесса: ОНАС им. А.С. Попова, 2012.
6. Воробйова О.М. Електроніка та мікросхемо техніка : підручник / О.М. Воробйова, Ю.В. Флейта. – Одеса: ОНАЗ ім.. О.С. Попова, 2015. – 298 с.
7. Воробйова О.М. Електроніка : навч. посіб. / Воробйова О.М., Савицька М.П., Флейта Ю.В. – Одеса: ФОП Бондаренко, 2016. – 236 с.
8. Гальперин М.В. Электроника и электроника : учебник / М.В. Гальперин. – М.: Форум НИЦ ИНФРА-М, 2013. – 480 с.
9. Горбачев Г.Н. Промышленная электроника : учебник ; 2-е узд. / Г.Н. Горбачев, Е.Е. Чаплыгин. – М.: Энергоатомизлат, 1998. – 340 с.
10. Джеймс А. Рог. Промышленная электроника / Джеймс А. – М.: ДМК Пресс, 2016. – 1136 с.
11. Малахов В.П. Схемотехника аналоговых устройств: Учебник. – Одесса: АстроПринт, 2000. – 212 с.
12. Маляр В.С. Теоретичні основи електроніки : навч. посіб. – Львів: Політехніка, 2018. – 210 с.
13. Матвієнко М. Промислова електроніка : підручник. – К.: Ліра, 2012. – 424 с.
14. Миклашевский С.П. Промышленная электроника : учебн. пособ. – 2-е
15. Панфилов И.П. Техническая электроника: учеб, пособ. / И.П. Панфилов, Ю.В. Флейта. – Одесса: ОЭИС им. А.С. Попова, 1990. – 78 с.
16. Панфілов І.П. Компонентна база радіоелектронної апаратури: навч. посіб. – Модуль 1 / Панфілов І.П., Савицька М.П., Флейта Ю.В. – Одеса: ОНАЗ ім. О.С. Попова, 2013. – 180 с.; модуль 2. – 2014. – 188 с.
17. Промышленная электроника : ученик для вузов ; под. ред. В.А. Лабунцова. – М.: Энергоатомиздат, 1988. – 311 с.
18. Руденко В.С. Приборы и устройства промышленной электроники. – К.: Техника, 1993. – 368 с.
19. Скаржепа В.А. Электроника и микросхемотехника. Ч. 1. Электронные устройства информационной автоматики: учебник / Скаржепа В.А., Луценко А.Н.; под общ. ред. А.А. Краснопрошиной. – К.: Вища школа, Головное изд-во, 1989. – 431 с.
20. Словник: Російсько-український політехнічний / Укл. В.С. Підлипенський, В.М. Петренко; за ред. Бусела В.Т. – К.: Ірпінь: ВТФ «Перун», 2000. – 512 с.

21. Антонов О.С. Обчислювальна техніка та мікропроцесори : навч. посіб. Ч. 1 / О.С. Антонов, І.В. Хіхловська. – Одеса: ОНАЗ ім.. О.С. Попова, 2008. – 262 с.

22. Брэй Б. Микропроцессоры Intel: 8086/8088, 80186/80188, 80286, 80386, 80486, Pentium, Pentium Pro Processor, Pentium II, Pentium III, Pentium 4. Архитектура, программирование и интерфейсы. – 6-е изд. / Пер. с англ. – С.Пб.: БХВ – Петербург, 2005. – 1328 с.: ил.

23. Майко Г.В. Ассемблер для IBM PC. – М.: Бизнес-Информ, Сирин, 1997. – 250 с.; ил.

24. Митрофанов Ю.М., Ошаровська О.В., Хіхловська І.В. Програмування на мові Асемблер: навч. посіб. для самостійної роботи з курсу «Цифрова техніка та мікропроцесори». – Одеса: УДАЗ, 1997. – 25 с.; іл..

25. Брамм П., Брамм Д. Микропроцессор 80386 и его программирование. Пер. с англ. – М.: Мир, 1990. – 448 с.; ил.

26. Майоров В.Г., Гаврилов А.И. Практический курс программирования микропроцессорных систем. – М.: Машиностроение, 1989. – 272 с.; ил.

27. Григорьев В.Л. Программирование однокристальных микропроцессоров. – М.: Энергоатомиздат, 1987. – 288 с.; ил.

Навчально-методичне видання

Воробйова Олена Михайлівна
Флейта Юрій Вікторович

ПРОМИСЛОВА ЕЛЕКТРОНІКА

Навчальний посібник

Частина ІІ

Підписано до друку 22.06.2021 р.
Формат 60/88/8 Обсяг 4,8 ум.-друк. арк.
Тираж 50 прим.