Міністерство освіти і науки України
Донбаська державна машинобудівна академія

Методичні вказівки
до лабораторного практикуму з дисципліни “Мікропроцесорні пристрої”

(для спеціальності 7.092203 “Електромеханічні системи автоматизації та електропривод”)

Частина 2

Перезатверджено
на засіданні кафедри ЕСА
Протокол № 1 від 28 серпня 2012р.

Краматорськ 2012

УДК 681.3

Методичні вказівки до лабораторного практикуму з дисципліни “Мікропроцесорні пристрої” (для спеціальності 7.092203 “Електромеханічні системи автоматизації та електропривод”) / Укл. О.М. Наливайко. – Краматорськ: ДДМА, 2012. - 98 с.

Викладені основні відомості про мікроконтролери сімейства PIC та основи їх програмування. Наведені індивідуальні завдання та контрольні питання для виконання лабораторних робіт на навчальних стендах PICDEM 2 PLUS

Укладач О.М. Наливайко, доц.

Відповід. за випуск О.М. Наливайко, доц.

Зміст

ВСТУП

Однокристальні мікроконтролери (ОМК) дозволяють істотно розширити інтелектуальні можливості різного роду пристроїв і систем. Вони є по суті спеціалізованими однокристальними МІКРОЕОМ, що містять для зв'язку із зовнішнім середовищем вбудовані периферійні вузли і пристрої, набір яких багато в чому визначає їх функціональні можливості і області застосування. Останнє спричинило за собою появу величезної різноманітності типів ОМК, які випускаються в даний час такими фірмами як, Intel, Motorola, Zilog, National, Mitsubishi Electric і ряд інших. Однокристальні мікроконтролери стали сьогодні одним з найпоширеніших елементів "програмованої логіки". Більше двох третин світового ринку мікропроцесорних засобів в даний час складають саме однокристальні мікроконтролери.

Переважне число ОМК мають традиційну (Фон-Неймановську або Прінстонську) архітектуру, в якої команди і дані передаються по одній шині. Особливий клас є мікроконтролери, архітектура яких заснована на концепції роздільних шин і областей пам'яті для даних і команд (Гарвардська архітектура). Дані мікроконтролери мають RISC-архітектуру, забезпечуючу просту але могутню систему команд, які виконуються за один цикл. До таких мікроконтролерів відносяться, зокрема, ОМК фірми Microchip сімейства PIC (12CXX, 16CXX, 17CXX).

У структуру ОМК сімейства PIC закладено багато різних функціональних особливостей, що роблять їх на сьогоднішній день самими високопродуктивними, мікроспоживаючими, перешкодозахисними, програмованими користувачем 8-ми бітовими мікроконтролерами. Завдяки цим особливостям ОМК сімейства PIC можуть обробляти апаратно-програмним способом як дискретні, так і аналогові сигнали, формувати різного роду керуючі сигнали, а також здійснювати зв'язок між собою і з ЕОМ, що знаходиться на вищому ієрархічному рівні в системі.

Фірмою Microchip також здійснюється могутня програмна, апаратна і інформаційна підтримка своїх виробів через мережу Internet і широко розгалужену у всьому світі ділерську мережу.

Друга частина лабораторного практикуму з дисципліни “Мікропроцесорні пристрої” призначена для закріплення теоретичних знань та придбання практичних навичок програмування PIC-мікроконтролерів.

1 КОРОТКИЙ ОПИС ОДНОКРИСТАЛЬНИХ МІКРОКОНТРОЛЕРІВ PIC

1.1 Загальні відомості

У залежності від розрядності команд, архітектурних особливостей і функціональних можливостей однокристальні мікроконтролери (ОМК) PIC поділяються на три основні групи (підсімейства):

1 Молодше підсімейство (12-розрядне процесорне ядро)-PIC 12C5xx, PIC 12C6xx, PIC 16C5xx.

2 Середнє підсімейство (14-розрядне процесорне ядро)- PIC 16C55x, PIC 16C6xx, PIC 16C7xx, PIC 16F8xx, PIC 16C9xx.

3 Старше підсімейство (16-розрядне процесорне ядро)- PIC 17Cxx, PIC 18Cxx, PIC 18Fxx.

Більшість ОМК, також як і мікроконтролери серії КР1816, має традиційну (Фон-Неймановську чи Прінстонську) архітектуру в якій команди і дані передаються по одній шині. Архітектура ж ОМК PIC заснована на концепції роздільних шин і областей пам'яті для даних і команд (Гарвардська архітектура) . Шина і пам'ять даних (ОЗП) має ширину 8 біт, а програмна шина і пам'ять (ПЗУ чи ППЗУ) має ширину 12, 14 чи 16 біт у залежності від сімейства ОМК. Така концепція забезпечує просту, але могутню систему команд, а двоступінчастий конвеєр забезпечує їхню одночасну вибірку і виконання. Усі команди складаються з одного слова (шириною 12, 14 чи 16 біт) і виповнюються за один цикл (200 нс при тактовій частоті 20Мгц), крім команд переходу, що виконуються за два цикли. За рахунок цього ОМК із RISC-архітектурою типу PIC 16/17/18 мають саму високу швидкодію в порівнянні з більшістю найбільш розповсюджених 8-бітових мікроконтролерів аналогічного класу і забезпечують більш ніж у 5-10 разів кращу продуктивність.

Контрольні іспити показують, що застосування ОМК серії PIC дозволяють зменшити час налагодження в 1,5-2 рази в порівнянні зі звичайними 8- розрядними мікроконтролерами.

Система команд PIC 12/16/17/18 включає тільки 33/37/57/75 команд і може бути легко і швидко вивчена. У конструкцію PIC включено багато енергозберігаючих особливостей, що роблять їх на сьогоднішній день самими мікроспоживаючими (у режимі SLEEP споживаний струм менш 1 мка), самими низьковольтними по напрузі живлення (2В) програмувальними користувачем мікроконтролерами.

Найпростіші типи таких ОМК містять 8-бітний таймер-лічильник з 8-бітним програмованим попереднім дільником (фактично 16-бітний таймер) і 6(20) ліній двохнаправленого вводу/виводу. Корпус таких ОМК має 8(18) виводів. Мікроконтролери середнього і старшого підсімейств містять крім

цього цілий ряд додаткових функціональних вузлів і блоків таких, наприклад, як: багатоканальні аналого-цифрові перетворювачі, розгалужену систему переривань, блоки керування рідкокристалевими індикаторами, компаратори, широтно-імпульсні модулятори, паралельні і послідовні інтерфейси типу I2C, RS-232 і т.п., цифрові перемножувачі, додаткові таймери-лічильники, збільшена кількість портів вводу/виводу дискретних сигналів та інше.

Таким чином, PIC 16/17/18 мають істотні переваги в порівнянні з іншими типами мікроконтролерів такого ж класу.

В даний час випускаються мікроконтролери з різним обсягом постійної й оперативної пам'яті, з різними типами тактових генераторів, з різною швидкодією і конструктивним виконанням, а також з різними функціональними можливостями. Конкретний тип мікроконтролера для рішення визначеної задачі можна вибрати використовуючи інформацію приведену в [1].

У залежності від технології виготовлення ПЗУ всі типи МК розділяються на п'ять груп:

1 Мікроконтролери, багаторазово програмовані користувачем, що, у свою чергу можуть бути розділені також на дві наступні групи:

1) Мікроконтролери з ультрафіолетовим витиранням. Ці МК оптимальні для експериментальних розробок і налагодження програм.

2) Мікроконтролери з багаторазово електрично програмованим користувачем ППЗУ (EEPROM) програм і даних. Ці МК дозволяють легко підбудовувати програму і дані під конкретні вимоги навіть після завершення асемблирування і тестування. Ця можливість може бути використана як для тиражування, так і для занесення каліброваних даних уже після остаточного тестування розробленого МКП.

Однак, дані МК мають обмежена кількість циклів перепрограмування.

2 Однократно програмовані мікроконтролери (OTP).

Ці МК можуть бути однократно запрограмовані користувачем і застосовуються в тих випадках, коли немає необхідності часто змінювати зміст програми чи конфігурацію мікроконтролера в розробленому МКП.

3 Мікроконтролери, програмовані виготовлювачем (QTP). Ці МК є замовленими і цілком програмованими на заводі-виготовлювачі по заздалегідь наданої користувачемем інформації.

4 Мікроконтролери, послідовно програмовані виготовлювачем (SQTP). Це так само замовлені однократно програмовані на заводі-виготовлювачі МК типу QTP, у яких декілька комірок, що задаються користувачем, у кожнім мікроконтролері програмуються різними серійними номерами.

5 Масочні мікроконтролери (ROM).

Ці МК також є замовленими і забезпечують максимально низьку вартість при крупносерійних замовленнях (наприклад, такими МК є PIC16CR54, PIC16CR56, PIC16CR57, PIC16CR58 і т.п.).

Для мікроконтролерних пристроїв (МКП) чи систем (МКС), програма яких може змінюватися, або містить які-небудь перемінні частини (таблиці, параметри калібрування, ключі і т.п.), випускаються мікроконтролери типу PIC 16F874(16F84) з багаторазово електрично перепрограмованою пам'яттю програм і даних – констант. Саме цей МК буде в основному використаний у прикладах, що будуть розглянуті далі. Це дозволить (при бажанні чи необхідності) перевірити роботу написаної і налагодженої програми за допомогою спеціальної макетної плати (універсальної – PICDEM-1, PICDEM-2, PICDEM-2plus чи саморобної – створеної самим користувачем).

1.3 Особливості структурної організації

Розглянемо особливості структурної організації, що виділяють PIC мікроконтролери серед інших ОМК такого ж класу, а також основні відмінності різних сімейств PIC друг від друга.

Для застосувань, пов'язаних із захистом інформації, кожен PIC має біт секретності, який може бути запрограмований для заборони прочитування програмного коду і ПЗП даних. При програмуванні спочатку записується програмний код, перевіряється на правильність запису, а потім встановлюється біт секретності. Якщо спробувати прочитати мікросхему зі встановленим бітом секретності, то для PIC16C5X старші 8 розрядів коду прочитуватимуться як 0, а молодші 4 розряди будуть скремблировані 12 розрядів команди. Для PIC16C84 аналогічні 7 старших розрядів прочитуватимуться нулями, а 7 молодших розрядів представлятимуть скремблировані 14 розрядів команди. Електрично перепрограмоване ПЗП даних PIC16C84 при установці біта захисту не може бути прочитане.

Мікроконтролери сімейства PIC мають дуже ефективну систему команд. Всі інструкції виконуються за один цикл, за винятком умовних переходів і команд, що змінюють програмний лічильник, які виконуються за 2 цикли. Один цикл виконання інструкції складається з 4 періодів тактової частоти. Таким чином, при частоті 4 Мгц, час виконання інструкції складає 1 мікросекунду. Кожна інструкція складається з 14 бітів, що діляться на код операції і операнд (можлива маніпуляція з регістрами, елементами пам'яті і безпосередніми даними). Висока швидкість виконання команд в PIC досягається за рахунок використання двохшинної Гарвардської архітектури замість традиційної одношинною Фон-Неймановської. Гарвардська архітектура грунтується на наборі регістрів з розділеними шинами і адресним простором для команд і для даних. Набір регістрів визначає, що всі програмні об'єкти, такі як порти введення/виводу, елементи пам'яті і таймери, є фізично реализовані апаратні регістри. Пам'ять даних (ОЗП) для PIC16CXX має розрядність 8 бітів, пам'ять програм (ППЗП) має розрядність 12 бітів для PIC16C5X і 14

бітів для PIC16XXX. Використання Гарвардської архітектури дозволяє досягти високої швидкості виконання бітових, байтових і регістрових операцій. Крім того, Гарвадська архітектура допускає конвейєрне виконання інструкцій, коли одночасно виконується поточна інструкція і прочитується наступна. У традиційній же Фон-Неймановській архітектурі команди і дані передаються через одну шину, що розділяється або мультиплексується, тим самим обмежуючи можливості конвейєризації. Писати програми для PIC не складніше, ніж для будь-якого іншого процесора. Логіка, і лише логіка... Звичайно, Гарвардська архітектура і велика розрядність команди дозволяють зробити код для PIC значно компактнішим, ніж для інших мікроконтролерів і істотно підвищити швидкість виконання програм.

1.2.1 Набір регістрів

Всі програмні об'єкти, з якими може працювати PIC, є фізичні регістри. Щоб зрозуміти, як працює PIC, потрібно розібратися з тим, які регістри у нього існують і як з кожним з них працювати.

РЕГІСТР НЕПРЯМОЇ АДРЕСАЦІЇ. Регістр непрямої адресації фізично не існує. Він використовується для непрямої вибірки одного з 64 можливих регістрів.

РЕГІСТР ТАЙМЕРА/ЛІЧИЛЬНИКА TMRX. Регістр таймера/лічильника TMRX може бути записаний і прочитаний як і будь-який інший регістр. TMRX може збільшуватися по зовнішньому сигналу, що подається на виведення RTCC, або по внутрішній частоті, відповідній частоті команд. Основне застосування таймера/лічильника - підрахунок числа зовнішніх подій і вимірювання часу. Сигнал від зовнішнього або внутрішнього джерела також може бути заздалегідь поділений за допомогою вбудованого в PIC програмованого дільника.

ПРОГРАМНИЙ ЛІЧИЛЬНИК PCL. Програмний лічильник (PC) використовується для генерації послідовності адрес комірок ПЗП програми, що містять 14-розрядні команди. PC має розрядність 13 бітів, що дозволяє прямо адресувати 8Кх14 комірок ПЗП.

РЕГІСТР СЛОВА СТАНУ STATUS. Регістр слова стану схожий на регістр PSW, що існує в більшості мікропроцесорів. У нім знаходяться біти перенесення, десяткового перенесення і нуля, а також біти режиму включення і біти сторінкової адресації.

РЕГІСТР ВИБОРУ FSR. Регістр вибору FSR використовується разом з регістром непрямої адресації для непрямої вибірки одного з 64 можливих регістрів. Фізично задіяно 36 регістрів ОЗП користувача, розташованих по адресах 0Ch-2Fh і 15 службових регістрів, розташованих по різних адресах.

РЕГІСТРИ ВВЕДЕННЯ/ВИВОДУ PORTX Регістри можуть бути індивідуально запрограмовані як входи або виходи за допомогою регістра

TRISX. Завдання 1 в розряді регістра TRIS програмує відповідний розряд порту як вхід. При читанні порту прочитується безпосередній стан виводу, при записі в порт запис відбувається в буферний регістр.

РЕГІСТРИ ЭППЗУ EEDATA, EEADR. PIC16XXX має вбудоване електрично перепрограмувальне ПЗП, яке може бути прочитане і записане за допомогою регістра даних EEDATA і регістра адреси EEADR. Запис нового байта триває близько 10 мсек і управляється вбудованим таймером. Управління записом і прочитуванням здійснюється через регістр EECON1. Для додаткового контролю за записом служить регістр EECON2.

РЕГІСТРИ ЗАГАЛЬНОГО ПРИЗНАЧЕННЯ. Регістри загального призначення є статичним ОЗП.

СПЕЦІАЛЬНІ РЕГІСТРИ W, INTCON, OPTION. На завершення розглянемо спеціальні регістри PIC. До них відносяться робочий регістр W, використовуваний в більшості команд як регістр акумулятора і регістри INTCON і OPTION. Регістр переривань INTCON (адреса 0Bh) служить для управління режимами переривання і містить біти дозволу переривань від різних джерел і прапори переривань. Регістр режимів OPTION (адреса 81h) служить для завдання джерел сигналу для попереднього дільника і таймера/лічильника, а також для завдання коефіцієнта ділення попереднього дільника, активного фронту сигналу для RTCC і входу переривання. Крім того за допомогою регістра OPTION можуть бути включені резистори навантажень для розрядів порту B, запрограмованих як входи.

1.2.2 Сторожовий таймер WDT

Сторожовий таймер WDT призначений для запобігання катастрофічним наслідкам від випадкових збоїв програми. Він також може бути використаний в додатках, пов'язаних з відліком часу, наприклад, в детекторі пропущених імпульсів. Ідея використання сторожового таймера полягає в регулярному його скиданні під управлінням програми або зовнішньої дії до того, як закінчиться його витримка часу і не відбудеться скидання процесора. Якщо програма працює нормально, то команда скидання сторожового таймера CLRWDT повинна регулярно виконуватися, оберігаючи процесор від скидання. Якщо ж мікропроцесор випадково вийшов за межі програми (наприклад, від сильної перешкоди по ланцюгу живлення) або зациклився на якій-небудь ділянці програми, команда скидання сторожового таймера швидше за все не буде виконана протягом достатнього часу, і відбудеться повне скидання процесора, що ініціалізує всі регістри і приводить систему в робочий стан. Сторожовий таймер не вимагає яких-небудь зовнішніх компонентів і працює на вбудованому генераторі RC, причому генерація не припиняється навіть у разі відсутності тактової частоти процесора. Типовий період сторожового таймера 18 мсек. Можна підключити попереднього дільника на сторожовий таймер і

збільшити його період аж до 2 сек. Ще однією функцією сторожового таймера служить включення процесора з режиму зниженого енергоспоживання, в який процесор переводиться командою SLEEP. У цьому режимі PIC споживає дуже малий струм - близько 1 мкА. Перейти з цього режиму в робочий режим можна або по зовнішній події натиснення кнопки, спрацьовуванню датчика, або по сторожовому таймеру.

1.2.3 Тактовий генератор

Для мікроконтролерів сімейства PIC можливе використання чотирьох типів тактового генератора: XT кварцовий резонатор, HS високочастотний кварцовий резонатор, LP мікроспоживаючий кварцовий резонатор і RC ланцюжок. Завдання типу використовуваного тактового генератора здійснюється в процесі програмування мікросхеми. У разі завдання варіантів XT, HS і LP до мікросхеми підключається кварцовий або керамічний резонатор або зовнішнє джерело тактової частоти, а у разі завдання варіанту RC - резистор і конденсатор. Звичайно, керамічний і, особливо, кварцовий резонатор значно точніший і стабільніший, але якщо висока точність відліку часу не потрібна, використання RC генератора може зменшити вартість і габарити пристрою.

1.2.4 Схема скидання

Мікроконтролери сімейства PIC використовують внутрішню схему скидання по включенню живлення у поєднанні з таймером запуску генератора, що дозволяє в більшості ситуацій обійтися без традиційного резистора і конденсатора. Досить просто підключити вхід MCLR до джерела живлення. Якщо при включенні живлення можливі імпульсні перешкоди або викиди, то краще використовувати послідовний резистор 100-300 Ом. Якщо живлення наростає дуже поволі (повільніше, ніж за 70 мсек), або Ви працюєте на дуже низьких тактових частотах, то необхідно використовувати традиційну схему скидання з резистора і конденсатора.

2 МІКРОКОНТРОЛЕР PIC16F877

2.1 Характеристика мікроконтролера

- Високошвидкісна архітектура RISC.
- 35 інструкцій.
- Всі команди виконуються за один цикл, окрім інструкцій переходів, що виконуються за два цикли.
- Тактова частота:

DC - 20МГц, тактовий сигнал DC - 200нс, один машинний цикл.

- 8к х 14 слів FLASH пам'яті програм.
- 368 байтів пам'яті даних (ОЗП).
- 256 EEPROM пам'яті даних.
- Сумісність по виводах з PIC16C73B/74B/76/77
- Система переривань (до 14 джерел).
- 8-рівневий апаратний стек.
- Прямій, непрямий і відносний режими адресації.
- Скидання по включенню живлення (POR).
- Таймер скидання (PWRT) і таймер очікування запуску генератора (OST) після включення живлення.
- Сторожовий таймер WDT з власним генератором RC.
- Програмований захист пам'яті програм.
- Режим енергозбереження SLEEP.
- 8 каналів 10-розрядного АЦП.
- Вибір параметрів тактового генератора.
- Високошвидкісна, енергозберігаюча CMOS FLASH/EEPROM технологія.
- Повністю статична архітектура.
- Програмування в готовому пристрої (використовується два виведення мікроконтролера).
- Низьковольтний режим програмування.
- Режим внутрішньосхемної відладки (використовується два виведення мікроконтролера).
- Широкий діапазон напруги живлення від 2.0В до 5.5В.
- Підвищена здатність навантаження портів введення/виводу (25мА).
- Мале енергоспоживання: < 0.6 мА при 3.0В, 4.0МГц; 20мкА при 3.0В, 32кГц ; < 1 мкА в режимі енергозбереження.

2.2 Структурна схема мікроконтролера PIC16F877

Структурна схема мікроконтролера PIC16F877 приведена на рисунку 2.1.

З розгляду даної структурної схеми видно, що фізичні і логічні компоненти, з яких складається PIC 16FXX аналогічні будь-якому іншому мікроконтролеру. Тому писати програми для PIC не складніше, ніж для будь-якого іншого процесора. Звичайно, Гарвардська архітектура і велика розрядність команди дозволяє зробити код для PIC значно більш компактним, чим для інших мікроконтролерів і істотно підвищити швидкість виконання програми.

Основу структури даного мікроконтролера складають дві внутрішні шини: двонаправлена 8-бітова шина даних і 14-бітова шина команд. Це відповідає, як вже згадувалося раніше, Гарвардській архітектурі, заснованій на концепції роздільних шин і областей пам'яті для даних і

команд. Шина даних зв'язує між собою всі основні функціональні блоки МК: пам'ять даних (RAM); арифметико-логічний пристрій (ALU); порти

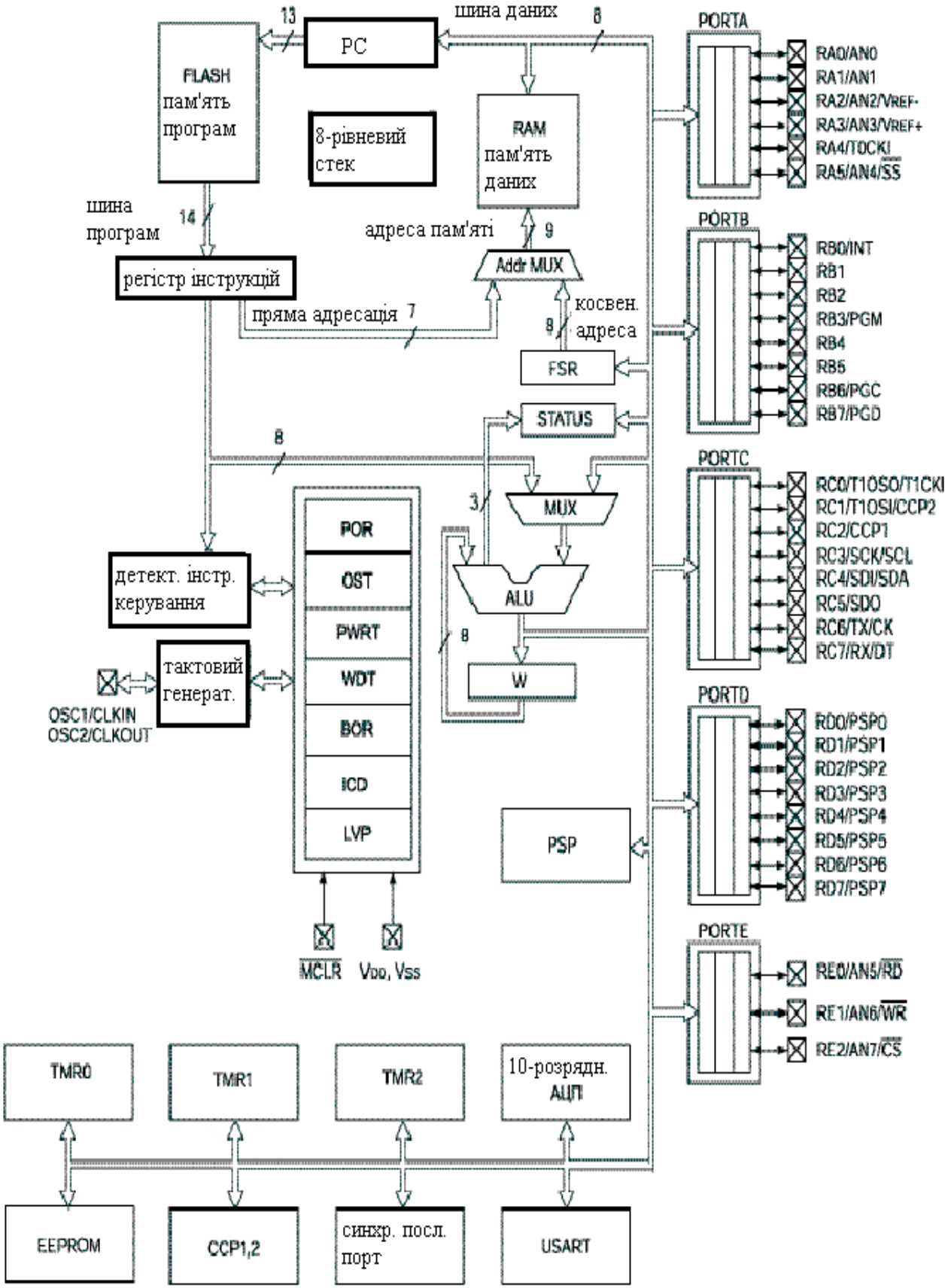


Рисунок 2.1 – Структура мікроконтролера PIC16F877

введення/виводу (PORT A,B,C,D,E); регістри стану (STATUS), непрямої адресації (FSR), таймерів-лічильників (TMR 0,1,2), програмного лічильника (PC).

У мікроконтролерах сімейства PIC існують пряма і непряма адресація всіх регістрів і елементів пам'яті. Всі спеціальні регістри і лічильник команд також відображаються на пам'ять даних.

Мікроконтролери PIC16FXX мають ортогональну (симетричну) систему команд, що дозволяє виконувати будь-яку операцію з будь-яким регістром, використовуючи будь-який метод адресації. Це полегшує програмування для них і значно зменшує час, необхідний на навчання роботі з ними.

Арифметико-логічний пристрій (ALU) 8-розрядний і виконує складання, віднімання, зрушення, бітові і логічні операції. У командах, що мають два операнди, одним з операндів є робочий регістр W. Другий операнд може бути константою або вмістом будь-якого регістра ОЗП. У командах з одним операндом, операнд може бути вмістом робочого регістра або вмістом будь-якого регістр. Для виконання всіх операцій ALU використовує робочий регістр W, який не може бути прямо адресований. Залежно від результату виконання операції, можуть змінитися значення бітів перенесення C, десяткового перенесення DC і нуля Z в регістрі стану STATUS. При відніманні биті C і DC працюють як біти позики і десяткової позики, відповідно.

Вхідна тактова частота, що поступає з виведення OSC1/CLKIN, усередині ділиться на чотири і з неї формуються чотири циклічні тактові послідовності що не перекриваються Q1, Q2, Q3 і Q4.

Вибірка команди і її виконання суміщені за часом таким чином, що вибірка команди займає один цикл, а виконання наступний цикл. Ефективний час виконання команди складає один цикл. Якщо команда змінює лічильник команд (наприклад, команда GOTO), то для виконання цієї команди буде потрібно два цикли. Цикл вибірки починається із збільшення лічильника команд в такті Q1. У циклі виконання команди вибрана команда защипується в регістр команд в такті Q1. Протягом тактів Q2, Q3 і Q4 відбувається декодування і виконання команди. У такті Q3 прочитується пам'ять даних (читання операнда), а запис відбувається в такті Q4. Таким чином, цикл виконання команди складається з 4-х тактів Q1-Q4, в кожному з яких проводяться різні, заздалегідь визначені дії:

Q1 - Вибірка певної команди з пам'яті програм і її декодування або вимушений NOP.

Q2 - Вибірка даних або NOP.

Q3 - Виконання команди і обробка даних.

Q4 - Запис даних або NOP.

Характеристика периферійних модулів:

Таймер 0: 8-розрядний таймер/лічильник з 8-розрядним програмованим перед дільником.

- Таймер 1: 16-розрядний таймер/лічильник з можливістю підключення зовнішнього резонатора.
- Таймер 2: 8-розрядний таймер/лічильник з 8-розрядним програмованим переддільником і вихідним дільником.
- Два модулі порівняння/захват/ШІМ (PCP): 16-розрядне захоплення (максимальна роздільна здатність 12.5нс); 16-розрядне порівняння (максимальна роздільна здатність 200нс); 10-розрядний ШІМ.
- Багатоканальний 10-розрядний АЦП.
- Послідовний синхронний порт MSSP; ведучий/ведомий режим SPI; ведучий/ведомий режим I2C.
- Послідовний синхронно-асинхронний приймач USART з підтримкою детектування адреси.
- Ведений 8-розрядний паралельний порт PSP з підтримкою зовнішніх сигналів -RD.-WR, -CS.
- Детектор зниженої напруги (BOD) для скидання по зниженню напруги живлення (BOR).

2.3 Організація пам'яті

У мікроконтролерах PIC16F87X існує два види пам'яті. Пам'ять програм і пам'ять даних мають роздільні шини даних і адреси, що дозволяє виконувати паралельний доступ.

2.3.1 Пам’ять програм

Мікроконтролери PIC16F87X мають 13-розрядний лічильник команд PC, здатний адресувати 8К х 14 слів пам'яті програм. Фізично реалізовано FLASH пам'яті програм 8К х 14 у PIC16F877. Адреса вектора скидання – 0000h. Адреса вектора переривань – 0004h.

Пам'ять програм (ППЗП) має сторінкову організацію (рисунок 2.2). Об'єм однієї сторінки 2048 14-ти розрядних слів . Мікроконтролер PIC 16F877 має чотири сторінки пам'яті програм для зберігання 14-ти розрядних кодів команд.

Всі мікроконтролери PIC16F87X здатні адресувати 8К слів пам'яті програм. Інструкції переходів (CALL і GOTO) мають 11 -разрядное поле для вказівки адреси, що дозволяє безпосередньо адресувати 2Кслів пам'яті програм. Для адресації верхніх сторінок пам'яті програм використовуються 2 бита в регістрі PCLATH<4:3>. Перед виконанням команди переходу (CALL або GOTO) необхідно запрограмувати біти регістра PCLATH<4:3> для адресації необхідної сторінки.

При виконанні інструкцій повернення з підпрограми, 13-розрядне значення для лічильника програм PC береться з вершины стека, тому маніпуляція бітами регістра PCLATH<3:4> не потрібна.

Рисунок 2.2 – Структура пам’яті програм

2.3.2 Організація пам'яті даних

Пам'ять даних розділена на чотири банки, які містять регістри загального і спеціального (SFR) призначення. Биті RP1 (STATUS<6>) і

RP0 (STATUS<5>) призначені для управління банками даних. У таблиці 2.1 показаний стан бітів, що управляють, при зверненні до банків пам'яті даних. Карта пам'яті даних показана на рисунку 2.3.

Таблиця 2.1 – Адресація банків даних

RP1:RPO	Банк
00	**0**
01	**1**
10	**2**
11	**3**

Об'єм банків пам'яті даних до 128 байтів (7Fh). На початку банку розміщуються регістри спеціального призначення, потім регістри загального призначення виконані як статичний ОЗП. Всі реалізовані банки містять регістри спеціального призначення. Деякі, часто використовувані регістри спеціального призначення можуть відображатися і в інших банках пам'яті.

Регістри можуть бути адресовані прямо або побічно, з використанням регістра непрямої адресації FSR. Безпосередня адресація підтримується спеціальними командами, що завантажують дані з пам'яті програми в робочий регістр W.

Регістри спеціального призначення використовуються для керування функціями мікроконтролера і можуть бути розділені на два набори: регістри базових функцій і регістри периферійних пристроїв. Регістри базових функцій містять у собі регістр-перемикач непряму адресації (INDF), програмний лічильник (PC), представлений двома регістрами PCL і PCLATH, регістр слова стану (STATUS), регістр-покажчик непряму адресації (FSR), робочий регістр (W), регістр переривань (INTCON), а також регістр режимів роботи чи конфігурації попереднього дільника і таймера (OPTION). Регістри периферійних пристроїв містять у собі регістри (RA,RB,RC,RD,RE-порти A,B,C,D,E), регістри даних (EEDATA) і адреси (EEADR) пам'яті даних-констант, регістри таймерів-лічильників (TMR0,1,2) і регістри керування конфігурацією портів вводу/виводу (TRISA, TRISB, TRISC, TRISD, TRISE).

Регістри загального призначення використовуються для зберігання даних по розсуду користувача.

Таблиця спеціальних регістрів приведена в додатку А.

Для виконання непрямої адресації необхідно звернутися до фізично не реалізованого регістра INDF. Звернення до регістра INDF фактично викличе дію з регістром, адреса якого вказана в FSR. Непряме читання регістра INDF (FSR=0) дасть результат 00h. Непрямий запис в регістр INDF не викличе ніяких дій (викликає дії на прапори АЛП в регістрі STATUS). 9-й біт непрямої адреси IRP зберігається в регістрі STATUS<7>. Приклад 9-розрядної непрямої адресації показаний на рисунку 2.4.

	Адреса		Адреса		Адреса		Адреса
Регістр * непрямої адресації	00h	Регістр * непрямої адресації	80h	Регістр * непрямої адресації	100h	Регістр * непрямої адресації	180h
TMR0	01h	OPTION	81h	TMR0	101h	OPTION REG	181h
PCL	02h	PCL	82h	PCL	102h	PCL	182h
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h
FSR	04h	FSR	84h	FSR	104h	FSR	184h
PORTA	05h	TRISA	85h		105h	x	185h
PORTB	06h	TRISB	86h	PORTB	106h	TRISB	186h
PORTC	07h	TRISC	87h	x	107h	x	187h
PORTD(1)	08h	TRISD(1)	88h	x	108h	x	188h
PORTE(1)	09h	TRISE(1)	89h	x	109h	x	189h
PCLATH	0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah
INTCON	0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh
PIR1	0Ch	PIE1	8Ch	EEDATA	10Ch	EECON1	18Ch
PIR2	0Dh	PIE2	8Dh	EEADR	10Dh	EECON2	18Dh
TMR1L	0Eh	PCON	8Eh	EEDATH	10Eh	Резерв^'	18Eh
TMR1H	0Fh	x	8Fh	EEADRH	10Fh	Резерв^'	18Fh
T1CON	10h	x	90h		110h		
TMR2	11h	SSPCON2	91h	Регістри		Регістри	
T2CON	12h	PR2	92h				
SSPBUF	13h	SSPADD	93h				
SSPCON	14h	SSPSTAT	94h	загального		загального	
CCPR1L	15h	x	95h				
CCPR1 H	16h	x	96h	призна-		призна-	
CCP1CON	17h	x	97h	чення		чення	
RCSTA	18h	TXSTA	98h				
TXREG	19h	SPBRG	99h				
RCREG	1Ah	x	9Ah				
CCPR2L	1Bh	x	9Bh	16 байтів		16 байтів	
CCPR2H	1Ch	x	9Ch				
CCP2CON	1Dh	x	9Dh				
ADRESH	1Eh	ADRESL	9Eh				
ADCONO	1Fh	ADCON1	9Fh		11Fh		19Fh
Регістри загаль-ного призна-чення 96 байтів	20h	Регістри загаль-ного призна-чення 80 байтів	A0h	Регістри загаль-ного призна-чення 80 байтів	120h	Регістри загаль-ного призна-чення 80 байтів	1A0h
			EFh		16Fh		1EFh
		Доступ до 70h- 7Fh	F0h	Доступ до 70h- 7Fh	170h	Доступ до 70h- 7Fh	1F0h
	7Fh		FFh		17Fh		1FFh
Банк 0		Банк 1		Банк 2		БанкЗ	

*- не фізичний регистр
x - не реалізовані, значення при читанні 00h

Рисунок 2.3 - Карта пам'яті даних мікроконтролерів PIC16F876/877

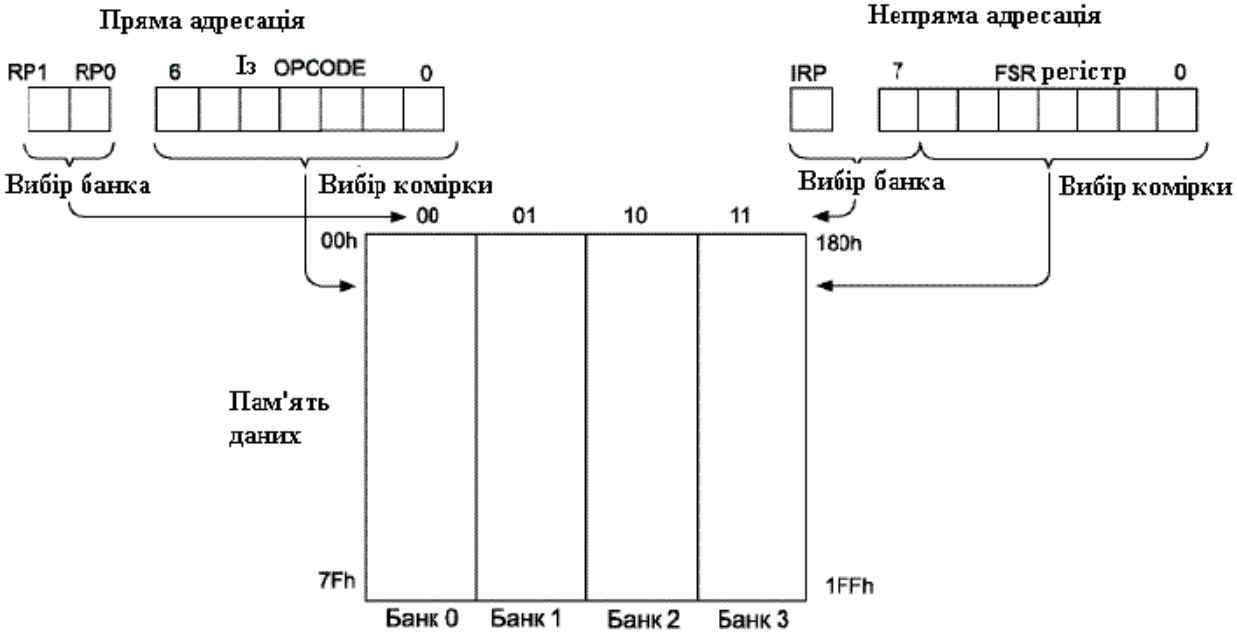


Рисунок 2.4 - Приклад 9-розрядної непрямої адресації

2.4 Регістр стану STATUS

Регістр стану (адреса 03h, 83h, 103h або 183h) містить арифметичні прапори АЛП, біти стану контролера при скиданні і біти вибору сторінок пам’яті. Регістр STATUS доступний для будь-якої команди так само, як будь-який інший регістр. Проте, биті TO і PD встановлюються апаратно і не можуть бути записані в регістр статусу програмно. Це слід мати на увазі при виконанні команди з використанням регістра статусу. Наприклад, команда CLRF обнулить всі біти, окрім бітів TO і PD, а потім встановить біт Z=1. Після виконання цієї команди регістр статусу може і не мати нульового значення (із-за бітів TO і PD) 000??100. Тому рекомендується для зміни регістра статусу використовувати тільки команди бітової установки BCF, BSF, MOVWF, які не змінюють решту біт статусу.

Розміщення прапорів в регістрі STATUS наступне:

b7	b6	b5	b4	b3	b2	b1	b0
IRP	RP1	RP0	-TO	- PD	Z	DC	C

Біт 7: IRP: Біт вибору банку при непрямій адресації
1- банк 2, 3(100h-1FFh)
0 - банк 0, 1 (000h - 0FFh)

Біти 6-5: RP1:RPO: Біти вибору банку при безпосередній адресації
11 – банк 3 (180h-1FFh)
10 – банк 2 (100h-17Fh)
01 - банк 1 (080h - 0FFh)
00 - банк 0 (000h - 07Fh)

Біт 4: -TO: Прапор переповнювання сторожового таймера
1 - після POR або виконань команд CLRWDT, SLEEP
0 - після переповнювання WDT

Біт 3: -PD: Прапор включення живлення
1 - після POR або виконань команди CLRWDT
0 - після виконання команди SLEEP

Біт 2: Z: Прапор нульового результату
1 - нульовий результат виконання арифметичної або логічної операції
0 - не нульовий результат виконання арифметичної або логічної операції

Біт 1: DC: Прапор десяткового перенесення/позики (для команд ADDWF, ADDWL, SUBWF, SUBWL), позика має інверсне значення
1 - було перенесення з молодшого півбайта
0 - не було перенесення з молодшого півбайта

Біт 0: C: Прапор перенесення/позики (для команд ADDWF, ADDWL, SUBWF, SUBWL), позика має інверсне значення
1 - було перенесення із старшого біта
0 - не було перенесення із старшого біта

Примітка. Прапор позики має інверсне значення. Віднімання виконується шляхом збільшення додаткового коду другого операнда. При виконанні команд зрушення (RRF, RLF) біт C завантажується старшим або молодшим бітом зрушуваного регістра.

2.5 Регістр OPTION

Регістр OPTION доступний для читання і запису, містить біти управління:

- Попереднім дільником TMR0/WDT;
- Активним фронтом зовнішнього переривання RB0/INT;
- Підтягаючими резисторами на входах PORTB.

Примітка. Якщо попередній дільник включений перед WDT, то коефіцієнт ділення тактового сигналу для TMR0 рівний 1:1.

Розміщення бітів керування в регістрі OPTION наступне:

b7	b6	b5	b4	b3	b2	b1	b0
-RBPU	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0

біт 7: -RBPU: Включення підтягаючих резисторів на входах PORTB:
1 = підтягаючі резистори відключені, 0 = підтягаючі резистори включені;
біт 6: INTEDG: Вибір активного фронту сигналу на вході зовнішнього переривання: INT 1 = переривання по передньому фронту сигналу,
0 = переривання по задньому фронту сигналу;
біт 5:TOCS: Вибір тактового сигналу для TMR0: 1 = зовнішній тактовий сигнал з виведення RA4/TOCKI, 0 = внутрішній тактовий сигнал CLKOUT;
біт 4: TOSE: Вибір фронту приросту TMR0 при зовнішньому тактовому сигналі: 1 = приріст по задньому фронту сигналу (з високого до низького рівня) на виведенні RA4/TOCKI, 0 = приріст по передньому фронту сигналу (з низького до високого рівня) на виведенні RA4/TOCKI;
біт 3: PSA: Вибір включення переддільника: 1 = переддільник включений перед WDT, 0 = переддільник включений перед TMR0;
біти 2-0: PS2 – PS0: Установка коефіцієнта ділення переддільника, коефіцієнти ділення приведені в таблиці 2.2.

Таблиця 2.2 - Коефіцієнти ділення переддільника

Значення PS2 – PS0	Для TMR0	Для WDT
000	1 :2	1:1
001	1 :4	1:2
010	1:8	1:4
011	1:16	1:8
100	1:32	1:16
101	1:64	1:32
110	1:128	1:64
111	1 :256	1:128

Примітка. При використанні режиму низьковольтного програмування і включених підтягаючих резисторах на PORTB необхідно скинути в '0' 3-й біт регістра TRISB для виключення підтягаючого резистора на виведенні RB3.

2.6 Регістр INTCON

Регістр INTCON доступний для читання і запису, містить біти дозволів і прапори переривань: переповнювання TMR0; зміни рівня сигналу на виводах PORTB; зовнішнє джерело переривань RBO/INT.

Розміщення бітів в регістрі INTCON наступне:

b7	b6	b5	b4	b3	b2	b1	b0
GIE	|PEIE	TOIE	INTE	RBI	TOIF	INTF	RBIF

біт 7: **GIE:** Глобальний дозвіл переривань:
1 = дозволені всі немасковані переривання,
0= всі переривання заборонені;

біт **6:** **PEIE:** Дозвіл переривань від периферійних модулів:
1= дозволені всі немасковані переривання периферійних модулів,
0= переривання від периферійних модулів заборонені;

біт **5:** **TOIE:** Дозвіл переривання по переповнюванню TMR0:
1= переривання дозволене,
0= переривання заборонене;

біт **4:** **INTE:** Дозвіл зовнішнього переривання INT:
1= переривання дозволене,
0= переривання заборонене;

біт **3:** **RBIE:** Дозвіл переривання по зміні сигналу на входах RB7:RB4 PORTB:
1= переривання дозволене,
0= переривання заборонене;

біт **2:** **TOIF:** Прапор переривання по переповнюванню TMR0:
1= відбулося переповнювання TMR0 (скидається програмно),
0= переповнювання TMR0 не було;

біт **1:** **INTF:** Прапор зовнішнього переривання INT:
1= виконана умова зовнішнього переривання на виведенні RBO/INT (скидається програмно),
0= зовнішнього переривання не було;

біт **0:** **RBIF:** Прапор переривання по зміні рівня сигналу на входах RB7:RB4 PORTB:
1 = зафіксована зміна рівня сигналу на одному з входів RB7:RB4 (скидається програмно),

0 = не було зміни рівня сигналу ні на одному з входів RB7:RB4.

Примітка. Прапори переривань встановлюються при виникненні умов переривань незалежно від відповідних бітів дозволу і біта загального дозволу переривань GIE (INTCON<7>).

2.7 Лічильник команд

13-розрядний регістр лічильника команд PC указує адресу виконуваної інструкції. Молодший байт лічильника команд PCL доступний для читання і запису. Старший байт PCH, що містить <12:8> біти лічильника команд PC, не доступний для читання і запису. Всі операції з регістром PCH відбуваються через додатковий регістр PCLATH. При будь-якому виді скидання мікроконтролера лічильник команд PC очищається. На рисунку 2.5 показано дві ситуації завантаження значення в лічильник команд PC. Приклад зверху, запис в лічильник команд PC відбувається при записі значення в регістр PCL (PCLATH <4:0> —> PCH). Приклад знизу, запис значення в лічильник команд PC відбувається при виконанні команди CALL або GOTO (PCLATH <4:3> -» PCH).

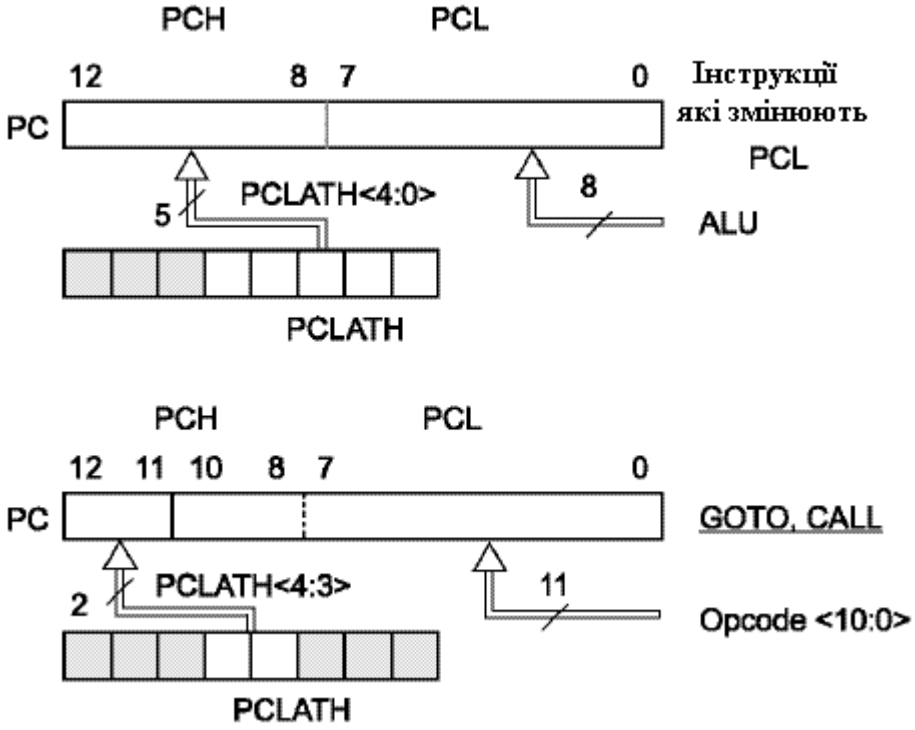


Рисунок 2.5 – Приклади завантаження лічильника команд

Обчислюваний перехід може бути виконаний командою приросту до регістра PCL (наприклад, ADDWF PCL). При виконанні табличного

читання обчислюваним переходом слід піклуватися про те, щоб значення PCL не перетнуло межу блоку пам'яті (кожен блок 256 байтів).

Всі мікроконтролери PIC16F87X здатні адресувати 8К слів пам'яті програм. Інструкції переходів (CALL і GOTO) мають 11 -разрядное поле для вказівки адреси, що дозволяє безпосередньо адресувати 2Кслов пам'яті програм. Для адресації верхніх сторінок пам'яті програм використовуються 2 біта в регістрі PCLATH<4:3>. Перед виконанням команди переходу (CALL або GOTO) необхідно запрограмувати біти регістра PCLATH<4:3> для адресації необхідної сторінки.

При виконанні інструкцій повернення з підпрограми, 13-розрядне значення для лічильника програм PC береться з вершины стека, тому маніпуляція бітами регістра PCLATH<3:4> не потрібна.

Примітка. Вміст регістра PCLATH не змінюється після виконання інструкції RETURN або RETFIE. Користувач винен сам змінити значення регістра PCLATH для подальшого виконання команд GOTO і CALL.

2.8 Стік

PIC16F87X мають 8-рівневий 13-розрядний апаратний стік. Стік не має відображення на пам'ять програм і пам'ять даних, не можна записати або прочитати дані із стека. Значення лічильника команд заноситься у вершину стека при виконанні інструкцій переходу на підпрограму (CALL) або обробки переривань. Читання із стека і запис в лічильник команд PC відбувається при виконанні інструкцій повернення з підпрограми або обробки переривань (RETURN, RETLW, RETFIE), при цьому значення регістра PCLATH не змінюється.

Стек працює як циклічний буфер. Після 8 записів в стек, дев'ятий запис запишеться на місце першого, а десятий запис замінить другий і так далі.

Примітки:

1. У мікроконтролерах не існує ніяких покажчиків про переповнювання стека.
2. У мікроконтролерах не передбачено команд запису/читання із стека, окрім команд виклику/повернення з підпрограм (CALL, RETURN, RETLW і RETFIE) або умов переходу по вектору переривань.

2.9 Порти введення/виводу

Регістри введення/виводу можуть управлятися, як будь-які інші регістри. Проте, команда «читання» (наприклад MOVF 6,W) завжди

прочитує фактичний рівень сигналу на виводі порту, незалежно від того, запрограмований цей розряд порту на введення або на вивід. Після сигналу «Скидання» всі порти введення/виводу встановлюються на «введення» (електрично еквівалентно третьому стану), а регістри керування введенням/виводом (TRISA, TRISB, TRISC, TRISD, TRISE) встановлюються в одиниці (конфігурація на введення). Для того, щоб конфігурувати деякі лінії порту на вивід, необхідно встановити відповідні біти в потрібному регістрі TRIS в «0». Це можна робити командою "TRIS f".

При операціях введення порти не защипуються. Вхідний сигнал повинен бути присутнім поки йде процес читання (напр. MOVF 6, W). При операціях виводу порти защипуються і зберігають значення до тих пір, поки не будуть перезаписані. На рисунку 2.6 не показані діоди, які захищають ніжку порту від зовнішніх імпульсів великої напруги. Вони обмежують імпульсну напругу на ніжці значеннями від Vss - 0,6 до Vdd + 0,6 В. Якщо статична напруга вийде за вказані межі, то виникнуть великі статичні струми, здатні вивести мікроконтролер з ладу.

Деякі канали портів введення/виводу мультипліційовані з периферійними модулями мікроконтролера. Коли периферійний модуль включений, вивід не може використовуватися як універсальний канал введення/виводу.

2.9.1 Регістри PORTA і TRISA

PORTA - 6-розрядний порт введення виводу. Всі канали PORTA мають відповідні біти напряму в регістрі TRISA, що дозволяють настроювати канал як вхід або вихід. Запис ‘1’ в TRISA переводить відповідний вихідний буфер в 3-й стан. Запис '0' в регістр TRISA визначає відповідний канал як вихід, вміст защіпки PORTA передається на виведення мікроконтролера (якщо вихідна защіпка підключена до виведення мікроконтролера).

Читання регістра PORTA повертає стан на виводах порту, а запис проводиться в защіпку PORTA. Всі операції запису в порт виконуються за принципом «читання - модифікація - запис», тобто спочатку проводиться читання стану виводів порту, потім зміна і запис в защіпку.

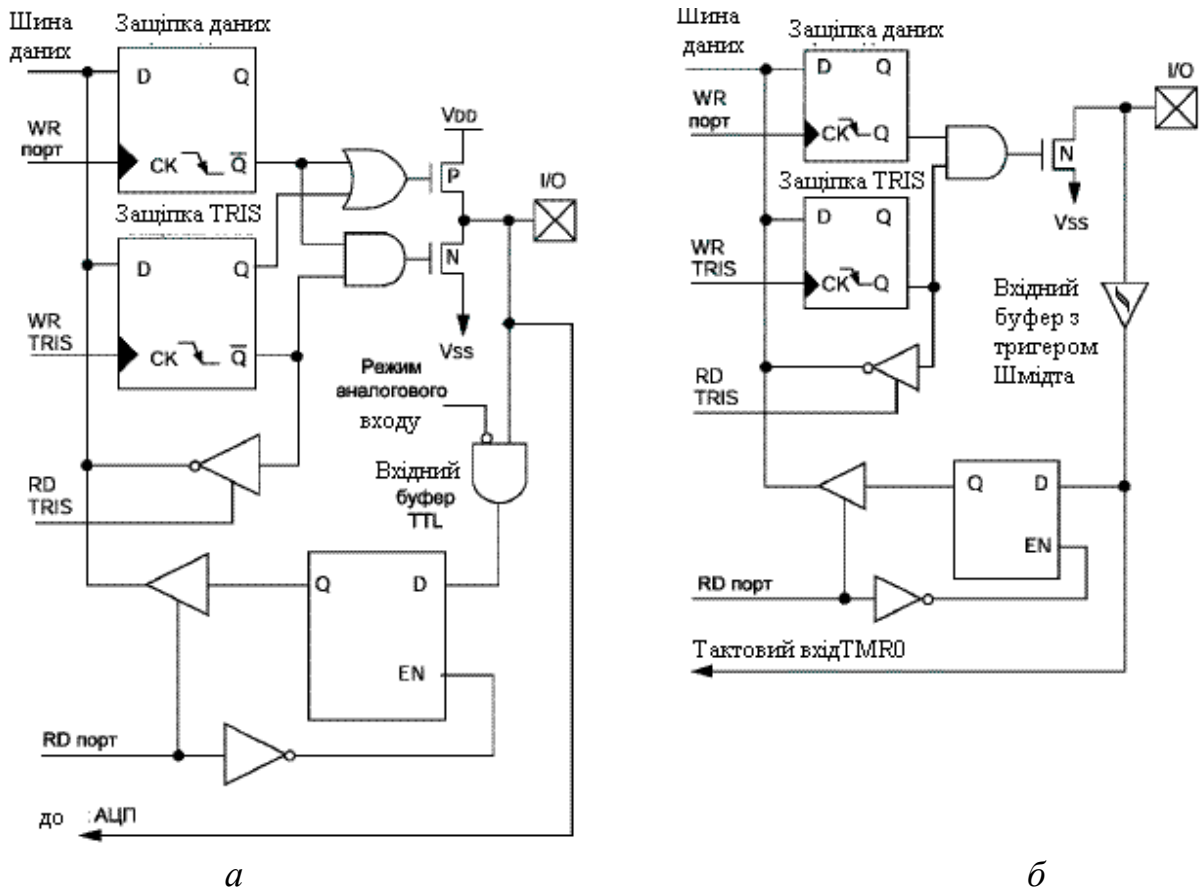
RA4 - має трігер Шмідта на вході і відкритий стік на виході, мультипліційований з тактовим входом TOCKI. Решта всіх каналів PORTA має TTL буфер на вході і повнофункціональні вихідні КМОП буфери.

Канали PORTA мультипліковані з аналоговими входами АЦП і аналоговим входом джерела опорної напруги VREF. Біти управління режимів роботи каналів порту введення/виводу PORTA знаходяться в регістрі ADCON1.

Примітка. Після скидання по включенню живлення виводи настроюються як аналогові входи, а читання дає результат '0'.

Біти регістра TRISA управляють напрямом каналів PORTA, навіть коли вони використовуються як аналогові входи. Користувач повинен упевнитися, що відповідні канали PORTA налаштовані на вхід при використанні їх як аналогові входи.

Структурні схеми виводів порта PORTA показані на рисунку 2.6.



а *б*

а – виводи RA3-RA0 та RA5 ; б – вивод RA4

Рисунок 2.6 – Структурна схема виводів порта PORTA

2.9.2 Регістри PORTB і TRISB

PORTB - 8-розрядний двонаправлений порт введення/виводу. Біти регістра TRISB визначають напрям каналів порту. Установка битва в 1 регістра TRISB переводить вихідний буфер в 3-й стан. Запис '0' в регістр

TRISB настроює відповідний канал як вихід, вміст защіпки PORTB передається на виведення мікроконтролера (якщо вихідна защіпка підключена до виведення мікроконтролера).

Три виведення PORTB мультиплікові з схемою низьковольтного програмування: RB3/PGM, RB6/PGC, RB7/PGD.

До кожного виведення PORTB підключений внутрішній підтягаючий резистор. Біт -RBPU (OPTION_REG <7>) визначає підключені (-RBPU=0) чи ні (-RBPU=1) підтягаючі резистори. Підтягаючі резистори автоматично відключаються, коли канали порту настроюються на вихід і після скидання по включенню живлення POR.

Чотири канали PORTB RB7:RB4, налаштовані на вхід, можуть генерувати переривання по зміні логічного рівня сигналу на вході. Якщо один з каналів RB7:RB4 налаштований на вихід, то він не може бути джерелом переривань. Сигнал на виводах RB7:RB4 порівнюється із значенням, збереженим при останньому читанні PORTB. У разі неспівпадання одного із значень встановлюється прапор RBIF (INTCON<0>), і якщо дозволено, генерується переривання.

Це переривання може вивести мікроконтролер з режиму SLEEP. У підпрограмі обробки переривань необхідно зробити наступні дії:

- Виконати читання або запис в PORTB, виключивши невідповідність;
- Скинути прапор RBIF в '0'.

Невідповідність збереженого значення з сигналом на вході PORTB завжди встановлює біт RBIF в 1. Читання з PORTB перерве умову невідповідності і дозволить скинути прапор RBIF в '0'.

Переривання по зміні сигналу на входах рекомендується використовувати для визначення натиснення клавіш, коли PORTB повністю задіяний для реалізації клавіатури. Не рекомендується опитувати PORTB при використанні переривань по зміні вхідного сигналу.

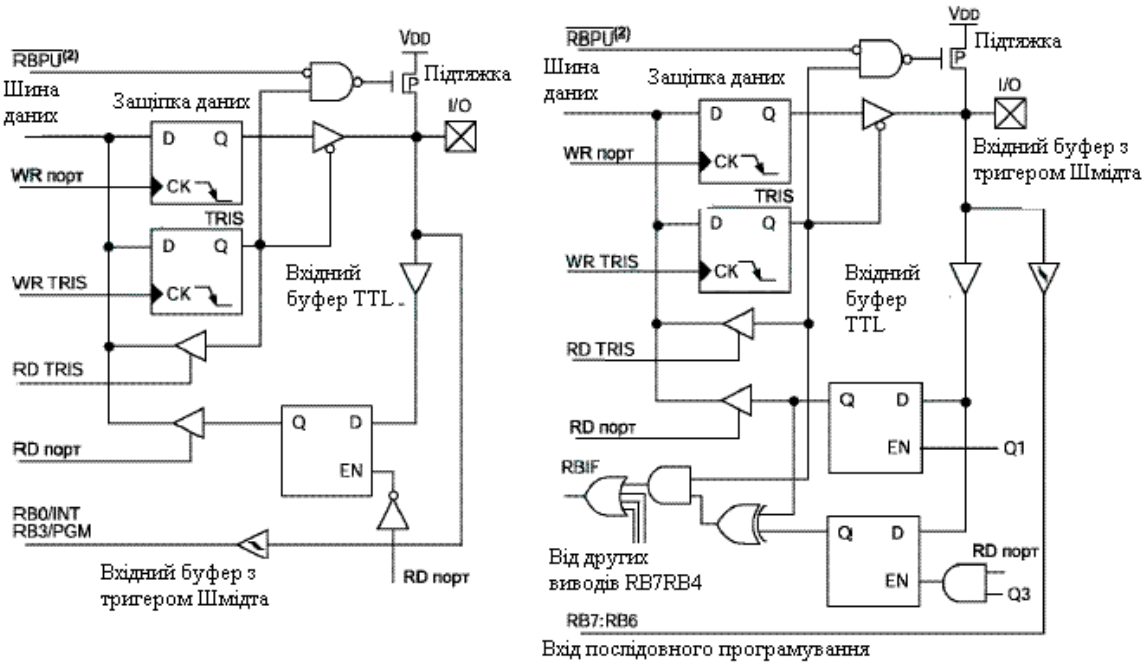
Переривання по зміні сигналу на входах PORTB і програма перемикання конфігурації цих каналів дозволяє реалізувати простий інтерфейс обслуговування клавіатури з виходом з режиму SLEEP по натисненню клавіш .

RBO/INT вхід зовнішнього джерела переривань, що настроюються бітом INTEDG (OPTION_REG<6>).

Структурна схема виводів порта PORTB показана на рисунку 2.7.

Примітки:

1. Виводи портів мають захисні діоди, підключені до V_{DD} і Vss-
2. Для включення підтягаючих резисторів необхідно встановити в '1' відповідний біт TRIS і скинути в '0' біт -RBPU (OPTION_REG<7>).



а *б*

а – виводи RB3-RB0 ; *б – виводи RB7-RB4*

Рисунок 2.7 – Структурна схема виводів порта PORTB

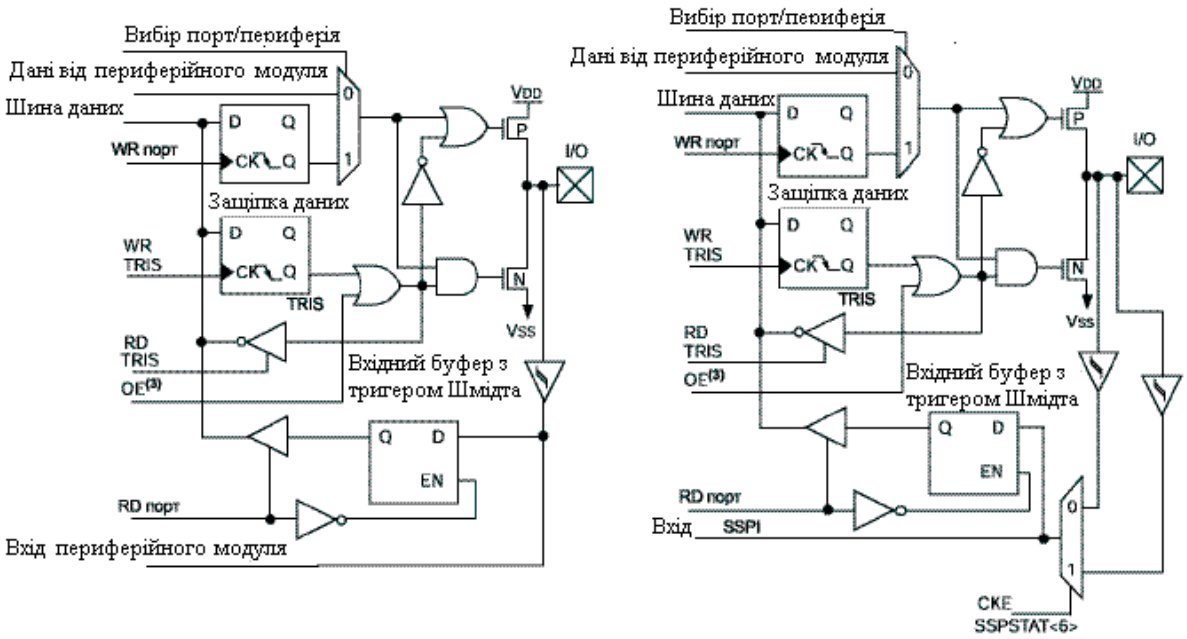
2.9.3 Регістри PORTC і TRISC

PORTC - 8-розрядний двонаправлений порт введення/виводу. Біти регістра TRISC визначають напрям каналів порту. Установка бита в '1' регістра TRISC переводить вихідний буфер в 3-й стан. Запис '0' в регістр TRISC настроює відповідний канал як вихід, вміст защіпки PORTC передається на виведення мікроконтролера (якщо вихідна защіпка підключена до виведення мікроконтролера).

Виводи PORTC мультиплексовані з декількома периферійними модулями. На каналах PORTC присутній вхідний буфер з трігером Шмідта.

Коли модуль MSSP включений в режимі I2C, виводи PORTC<4:3> можуть підтримувати рівні вихідних сигналів по специфікації I2C або SMBus залежно від стану бита CKE(SSPSTAT<6>).

При використанні периферійних модулів необхідно відповідним чином настроювати біти регістра TRISC для кожного виведення PORTC. Деякі периферійні модулі відміняють дію бітів TRISC примусово настроюючи вивід на вхід або вихід. У зв'язку з чим не рекомендується використовувати команди "Читання - модифікація - запис" з регістром TRISC. Структурна схема виводів порта PORTC показана на рисунку 2.8.



а *б*

а – виводи RC7-RC5, RC2-RC0; *б – виводи RC4-RC3*

Рисунок 2.8 – Структурна схема виводів порта PORTC

Примітки:

1. Виводи портів мають захисні діоди, підключені до VDD і Vss.
2. Сигнал режиму каналу - вивід використовується периферійним модулем або цифровий порт введення/виводу.
3. Сигнал дозволу (OE) від периферійного модуля, настроює канал як вихід.

2.9.4 Регістри PORTD і TRISD

PORTD - 8-розрядний двонаправлений порт введення/виводу. Біти регістра TRISD визначають напрям каналів порту.

PORTD може працювати як 8-розрядний мікропроцесорний порт (ведений паралельний порт), якщо біт PSPMODE (TRISE<4>) встановлений в '1'. У режимі веденого паралельного порту до входів підключені буфери TTL. Структурна схема виводів порта PORTD показана на рисунку 2.9.

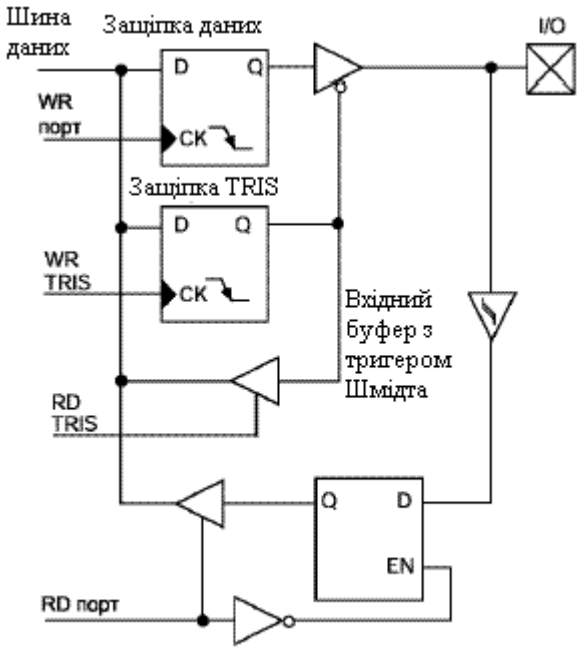


Рисунок 2.9 – Структурна схема виводів порта PORTD

2.9.5 Регістри PORTE і TRISE

PORTE має три виводи (REO/-RD/AN5, RE1/-WR/AN6, RE2/-CS/AN7), що індивідуально настроюються на вхід або вихід. Виводи PORTE мають вхідний буфер Шмідта.

Канали PORTE стануть керуючими виводами веденого паралельного порту коли біт PSPMODE(TRISE<4>) встановлений в '1'. У цьому режимі біти TRISE<2:0> повинні бути встановлені в '1'. У регістрі ADCON1 необхідно також настроїти виводи PORTE як цифрові канали введення/виводу. У режимі веденого паралельного порту до виводів PORTE підключені вхідні буфери TTL.

Виводи PORTE мультипліковані з аналоговими входами. Коли канали PORTE настроєні як аналогові входи, біти регістра TRISE управляють напрямом даних PORTE (читання даватиме результат '0'). Структурна схема виводів порта PORTE аналогічна схемі порта PORTD.

2.10 Таймери

Контролер PIC16F877 має три багатофункціональні таймери. Кожен модуль таймера/лічільника може працювати окремо або входити складовою частиною в склад модулів спеціального призначення.

2.10.1 Модуль таймера TMR0

TMR0 - таймер/лічильник, має наступні особливості:

- 8-розрядний таймер/лічильник;
- Можливість читання і запису поточного значення лічильника;
- 8-розрядний програмований переддільник;
- Внутрішнє або зовнішнє джерело тактового сигналу;
- Вибір активного фронту зовнішнього тактового сигналу;
- Переривання при переповнюванні (перехід від FFh до 00h).

Блок схема модуля TMR0 і загального з WDT переддільника показана на рисунку 2.10.

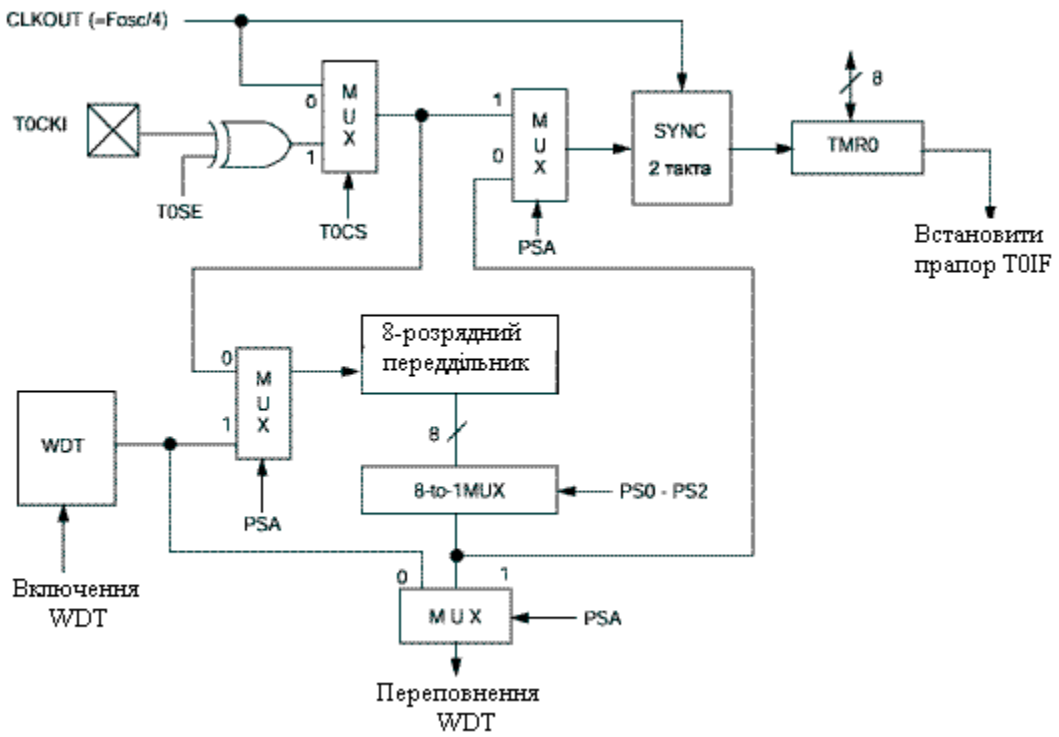


Рисунок 2.10 – Блок схема модуля TMR0

Примітка. Біти управління TOCS, TOSE, PS2, PS1, PSO, PSA розташовані в регістрі OPTION_REG.

Коли біт TOCS скинутий в '0' (OPTION_REG<5>), TMR0 працює від

внутрішнього тактового сигналу. Приріст лічильника TMR0 відбувається в кожному машинному циклі (якщо переддільник відключений). Після запису в TMR0 приріст лічильника заборонений два наступні цикли. Користувач повинен скоректувати цю затримку перед записом нового значення в TMR0.

Якщо біт TOCS встановлений в '1' (OPTION_REG<5>), TMR0 працює від зовнішнього джерела тактового сигналу з входу RA4/TOCKI. Активний фронт зовнішнього тактового сигналу вибирається бітом TOSE в регістрі OPTION_REG<4> (TOSE=0 - активним є передній фронт сигналу). Переддільник може бути включений перед WDT або TMR0, залежно від стану бита PSA (OPTION_REG<3>). Не можна прочитати або записати нове значення в переддільник.

Переривання від TMR0 виникають при переповнюванні лічильника, тобто під час переходу його значення від FFh до 00h. При виникненні переривання встановлюється в '1' біт TOIF(INTCON<2>). Само переривання може бути дозволено/заборонено установкою/скиданням біта TOIE (INTCON<5>). Прапор переривання від TMR0 TOIF (INTCON<2>) повинен бути скинутий в підпрограмі обробки переривань. У SLEEP режимі мікроконтролера модуль TMR0 вимкнений і не може генерувати переривання.

Якщо переддільник не використовується, зовнішній тактовий сигнал поступає безпосередньо на синхронізатор. Синхронізація TOCKI з таким сигналом мікроконтролера ускладнюється із-за опиту виходу синхронізатора в машинні цикли Q2 і Q4. Тому тривалість високого або низького логічного рівня зовнішнього сигналу повинна бути не менше 2Tosc (плюс невелика затримка внутрішнього RC ланцюга 20нс).

8-розрядний лічильник може працювати як переддільник TMR0 або вихідний дільник WDT. Для простоти опису цей лічильник завжди називатимемо «переддільник». Зверніть увагу, що існує тільки один переддільник, який може бути включений перед TMR0 або WDT. Використання переддільника перед TMR0 означає, що WDT працює без переддільника, і навпаки.

Коефіцієнт ділення переддільника визначається бітами PSA і PS2:PSO в регістрі OPTION_REG<3:0>.

Якщо переддільник включений перед TMR0, будь-які команди запису в TMR0 (наприклад, CLRF 1, MOVWF 1, BSF 1,x і т.д.) скидають переддільника. Коли переддільник підключений до WDT, команда CLRWDT скине переддільника разом з WDT. Переддільник також очищається при скиданні мікроконтролера. Переддільник недоступний для читання/запису.

Примітка. Запис в регістр TMR0 скине переддільника, якщо він підключений до TMR0, але не змінить його режиму роботи.

2.10.2 Модуль таймера TMR1

TMR1 - 16-розрядний таймер/лічильник, що складається з двох 8-розрядних регістрів (TMR1H і TMR1L) доступних для читання і запису. Рахунок виконується в спарених регістрах (TMR1H : TMR1L), інкрементуючи їх значення від 0000h до FFFFh, далі рахує з 0000h. При переповнюванні лічильника встановлюється в ‘1’ прапор переривання TMR1IF в регістрі PIR1<0>. Само переривання можна дозволити/заборонити установкою/скиданням, біта TMR1IE в регістрі P1E1<0>. Блок схема модуля TMR1 показана на рисунку 2.11.

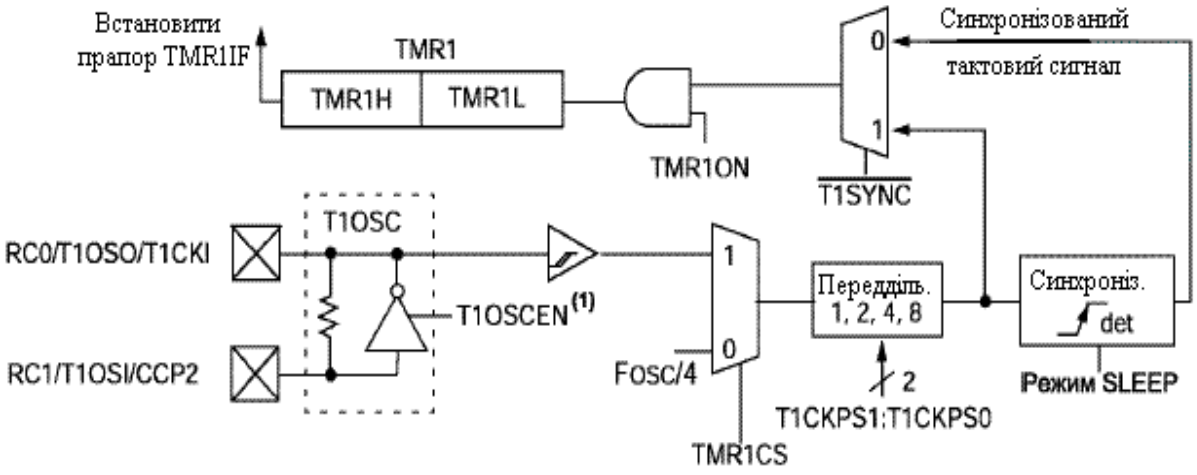


Рисунок 2.11 – Блок схема модуля TMR1

Примітка. Якщо T1OSCEN=0, то інвертуючий елемент і резистивний зворотний зв'язок вимкнені для зменшення струму споживання.

TMR1 може працювати в двох режимах:

- Режим таймера;
- Режим лічильника.

Включення модуля TMR1 здійснюється установкою бита TMR1ON в ‘1’ (T1COM<0>).

Бітом TMR1CS (T1CON<1>) вибирається джерело тактових імпульсів. У режимі таймера TMR1 інкрементується на кожному машинному циклі. Якщо TMR1 працює із зовнішнім джерелом тактового сигналу, то приріст відбувається по кожному передньому фронту сигналу.

TMR1 має внутрішній вхід скидання від модуля CPP .

Коли включений генератор тактових імпульсів (T1OSCEN=1), виводи RC1/T1OSI/CCP2 і RCO/T1OSO/T1CKI настроєні як входи. Значення бітів TRISC<1:0> ігнорується, а читання даних з цих виводів дає результат '0'.

Керуючі биті TMR1 знаходяться в регістрі T1CON (адреса 10h):

b7	b6	b5	b4	b3	b2	b1	b0
-	-	T1CKPS1	T1CKPS0	T1OSCEN	-T1SYNC	TMR1CS	TMR1ON

біти 7-6: не реалізовані: читаються як '0';

біти 5-4: T1CKPS1:T1CKPS0: вибір коефіцієнта ділення переддільника TMR1 11 = 1:8, 10= 1:4, 01 = 1:2, 00=1:1;

біт 3:T1OSCEN: включення тактового генератора TMR1: 1 = генератор включений, 0 = генератор вимкнений (інвертуючий елемент і резистивний зворотний зв'язок вимкнені для зменшення струму споживання);

біт 2: -T1SYNC: синхронізація зовнішнього тактового сигналу TMR1CS = 1
1= не синхронізувати зовнішній тактовий,
0= синхронізувати зовнішній тактовий,

TMR1CS = 0 : значення біта ігнорується;

біт 1: TMR1CS: вибір джерела тактового сигналу:
1= зовнішнє джерело з виведення RCO/T1OSO/T1CKI (активним є передній фронт сигналу),
0= внутрішнє джерело Fosc/4

біт 0: TMR1ON: Включення модуля TMR1: 1=включений, 0 = вимкнений.

В режимі таймера приріст походить від внутрішнього сигналу Fosc/4, коли біт TMR1CS (T1CON<1>) скинутий в '0'. У цьому режимі біт синхронізації T1SYNC (T1COM<2>) ігнорується, тому що внутрішній тактовий сигнал завжди синхронізований.

В режимі лічильника TMR1 може працювати в синхронному або асинхронному режимі залежно від стані бита TMR1CS. Коли TMR1 використовує зовнішній тактовий сигнал, приріст таймера відбувається по передньому фронту. Включивши TMR1 в режим зовнішнього тактового сигналу рахунок почнеться тільки після появи заднього фронту.

В режимі синхронного лічильника робота TMR1 від зовнішнього джерела тактового сигналу вибирається установкою біта TMR1CS в ‘1’. В цьому режимі приріст таймера відбувається по кожному передньому фронту сигналу на виведенні RC1/T1OSI/CCP2 (якщо T1OSCEN=1) або RCO/T1OSO/T1CKI (якщо T1OSCEN=0).

Якщо -T1SYNC=0, то активний фронт зовнішнього тактового сигналу синхронізується з внутрішнім тактовим сигналом на виході асинхронного переддільника.

У SLEEP режимі мікроконтролера лічильник не буде інкрементуватися (за наявності тактового сигналу), оскільки синхронізатор вимкнений (переддільник продовжує рахунок тактових імпульсів)

В режимі асинхронного лічильника, якщо біт -T1SYNC (T1CON<2>) встановлений в ‘1’, зовнішній тактовий сигнал TMR1 не синхронізуватиметься з внутрішнім тактовим сигналом мікроконтролера, таймер продовжує працювати в режимі SLEEP. Переповнювання таймера викличе «пробудження» мікроконтролера, якщо дозволено переривання від TMR1. Проте потрібна обережність при записі/читанні TMR1. У цьому режимі TMR1 не може використовуватися для захоплення/порівняння даних модуля PCP.

Читання TMR1H або TMR1L, під час рахунку в асинхронному режимі, гарантує отримання поточного значення лічильника (реалізовано апаратно). Проте користувач повинен мати на увазі, що читання 16-розрядного значення виконується по байтно. Це накладає деякі обмеження, оскільки таймер може переповнитися між читаннями байтів.

Запис в TMR1 рекомендується виконувати після зупинки таймера. Запис в регістри TMR1 під час приросту таймера може привести до непередбачуваного значення регістра.

Якщо модуль CCP1 або CCP2 працює в режимі порівняння з трігером спеціальних функцій (CCP1M3 : CCP1 M0=1011), то сигнал трігера скине TMR1. TMR1 повинен працювати в режимі синхронізованого зовнішнього тактового сигналу або внутрішнього тактового сигналу. У асинхронному режимі ця функція не працює.

Коли запис в TMR1 співпадає з сигналом скидання від трігера спеціальних подій, пріоритет віддається запису в TMR1.

У цьому режимі модуля CCP період скидання TMR1 зберігається в регістрах CCPRxH:CCPRxL.

Регістри TMR1H і TMR1L не скидаються в 00h при скиданні по включенню живлення POR і інших видах скидання, окрім скидання по сигналу трігера спеціальних подій модуля CCP1 або CCP2.

Регістр T1CON скидається в 00h при скиданні POR і BOR (TMR1 вимикається, коефіцієнт переддільника рівний 1:1). При решті всіх видів скидання значення регістра T1CON не змінюється.

Переддільник TMR1 очищається при записі в регістр TMR1L або TMR1H.

2.10.3 Модуль таймера TMR2

TMR2 - 8-розрядний таймер з програмованими переддільником і вихідним дільником, 8-розрядним регістром періоду PR2. TMR2 може бути опорним таймером для CCP модуля в ШІМ режимі. Регістри TMR2 доступні для запису/читання і очищаються при будь-якому виді скидання.

Вхідний тактовий сигнал (Fosc/4) поступає через переддільника з програмованим коефіцієнтом ділення (1:1, 1:4 або 1:16), визначуваний бітами T2CKPS1 :T2CKPSO (T2CON<1:0>).

TMR2 рахує, інкрементуючи від 00h до значення в регістрі PR2, потім скидається в 00h на наступному машинному циклі. Регістр PR2 доступний для запису і читання. Після скидання значення регістра PR2 рівне FFh.

Блок схема модуля TMR2 показана на рисунку 2.12.

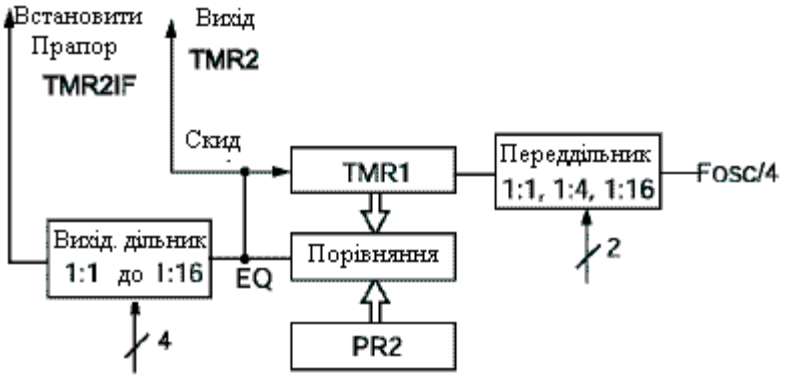


Рисунок 2.12 – Блок схема модуля TMR2

Сигнал переповнювання TMR2 проходить через вихідного 4-розрядного дільника з програмованим коефіцієнтом ділення (від 1:1 до 1:16 включно) для установки прапора TMR2IF в регістрі PIR1 <1 >.

Для зменшення енергоспоживання таймер TMR2 може бути вимкнений скиданням біта TMR2ON (T2COM<2>) в'0'.

TMR2 може використовуватися для програмного вибору швидкості обміну даними модуля SSP.

Керуючі біти TMR2 знаходяться в регістрі T2CON (адреса 12h):

b7	b6	b5	b4	b3	b2	b1	b0
-	**TOUTPS3**	**TOUTPS2**	**TOUTPS1**	**TOUTPS0**	**TMR2ON**	**T2CKPS1**	**T2CKPS0**

біт 7: не реалізований: читається як '0';
біти 6-3: TOUTPS3:TOUTPS0: вибір коефіцієнта вихідного дільника TMR2
0000=1:1
0001 = 1:2
0010 = 1:3
...
1111 =1:16

біт 2: TMR2ON: включення модуля TMR2 1 = включений,0= вимкнений;
біти 1 - 0: T2CKPS1 :T2CKPSO: вибір коефіцієнта ділення переддільника TMR2:
00= 1:1
01=1:4

1x= 1:16

Лічильник переддільника і вихідного дільника скидаються у випадку:

- Запису в регістр TMR2;
- Запису в регістр T2CON;
- Будь-якого виду скидання мікроконтролера (POR, BOR, скидання WDT або активний сигнал -MCLR).

Регістр TMR2 не очищається при запису в T2CON.

Сигнал переповнювання TMR2 (до вихідного переддільника) поступає в модуль SSP для управління швидкістю передачі даних.

2.11 Модуль 10-розрядного АЦП

Модуль аналого-цифрового перетворення (АЦП) має п'ять каналів у 28-вивідних мікросхем і вісім каналів у 40/44-вивідних мікросхем.

Вхідний аналоговий сигнал через комутатор каналів заряджає внутрішній конденсатор АЦП C_{HOLD}. Модуль АЦП перетворить напруга, що утримується на конденсаторі C_{HOLD} у відповідний 10-розрядний цифровий код методом послідовного наближення. Джерело верхньої і нижньої опорної напруги може бути програмно вибране з виводів Vdd, Vss, RA2 або RA3.

Допускається робота модуля АЦП в SLEEP режимі мікроконтролера, при цьому як джерело тактових імпульсів для АЦП повинен бути вибраний RC генератор.

Для управління АЦП в мікроконтролері використовується 4 регістри:

- Регістр результату ADRESH (старший байт);
- Регістр результату ADRESL (молодший байт);
- Регістр управління ADCON0;
- Регістр управління ADCON1.

Регістр ADCON0 використовується для настройки роботи модуля АЦП, а за допомогою регістра ADCON1 встановлюється які входи мікроконтролера використовуватимуться модулем АЦП і в якому режимі (аналоговий вхід або цифровий порт введення/виводу).

ADCON0 (адреса 1Fh)

b7	b6	b5	b4	b3	b2	b1	b0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/-	-	ADON

біти 7-6: ADCS1: ADCS0: Вибір джерела тактового сигналу

00 = Fosc/2
01 = Fosc/8
10 = Fosc/32
11 = Frc (внутрішній RC генератор модуля АЦП)

біти 5-3: CHS2:CHS0: Вибір аналогового каналу
000= канал 0 (RAO/ANO)
001= канал 1(RA1/AN1)
010= канал 2 (RA2/AN2)
011= канал 3 (RA3/AN3)
100= канал 4 (RA5/AN4)
101= канал 5 (RE0/AN5)
110= канал 6 (RE1/AN6)
111 = канал 7 (RE2/AN7)

біт 2: GO/-DONE: Біт статусу модуля АЦП
Якщо ADON=1
1 = модуль АЦП виконує перетворення (установка біта викликає початок перетворення)
0 = стан очікування (апаратно скидається по завершенню перетворення)

біт 1: Не використовується: читається як '0'

біт 0: ADON: Біт включення модуля АЦП
1= модуль АЦП включений
0 = модуль АЦП вимкнений і не споживає струму.

ADCON1 (адреса 9Fh)

b7	b6	b5	b4	b3	b2	b1	b0
ADFM	-	-	-	PCFG3	PCFG2	PCFG1	PCFG0

біт 7: ADFM: Формат збереження 10-розрядного результату
1 = праве вирівнювання, 6 старших біт ADRESH читаються як '0'
0 = ліве вирівнювання, 6 молодших біт ADRESL читаються як '0'

біти 6-4: Не використовуються: читаються як '0';

биті 3-0: PCFG3:PCFG0: Керівні біти настройки каналів АЦП їх значення приведене в таблиці 2.3.

Таблиця 2.3 - Біти настройки каналів АЦП

PCGF3: PCGF0	AN7 RE2	AN6 RE1	AN5 RED	AN4 RA5	AN3 RA3	AN2 RA2	AN1 RA1	AND RAO	VREF+	VREF-	Кан./ VREF
0000	A	A	A	A	A	A	A	A	VDD	V_{SS}	8/0
0001	A	A	A	A	VREF+	A	A	A	RA3	V_{SS}	7/1
0010	D	D	D	A	A	A	A	A	VDD	V_{SS}	5/0
0011	D	D	D	A	VREF+	A	A	A	RA3	V_{SS}	4/1
0100	D	D	D	D	A	D	A	A	VDD	V_{SS}	3/0
0101	D	D	D	D	VREF+	D	A	A	RA3	V_{SS}	2/1

011x	D	D	D	D	D	D	D	D	VDD	V_{SS}	0/0
1000	A	A	A	A	VREF+	VREF-	A	A	RA3	RA2	6/2
1001	D	D	A	A	A	A	A	A	VDD	V_{SS}	6/0
1010	D	D	A	A	VREF+	A	A	A	RA3	V_{SS}	5/1
1011	D	D	A	A	VREF+	VREF-	A	A	RA3	RA2	4/2
1100	D	D	D	A	VREF+	VREF-	A	A	RA3	RA2	3/2
1101	D	D	D	D	VREF+	VREF-	A	A	RA3	RA2	2/2
1110	D	D	D	D	D	D	D	A	VDD	Vss	1/0
1111	D	D	D	D	VREF+	VREF-	D	A	RA3	RA2	1/2

A = аналоговий вхід; D = цифровий канал введення/виводу.

В останньому стовпці указується число аналогових каналів, доступних для перетворення і число входів джерела опорної напруги.

У регістрі ADRESH:ADRESL зберігається 10-розрядний результат аналого-цифрового перетворення. Коли перетворення завершене, результат перетворення записується в регістр ADRESH:ADRESL, після чого скидається прапор GO/-DONE (ADCON0<2>) і встановлюється прапор переривання ADIF. Структурна схема модуля АЦП показана на рисунку 2.13.

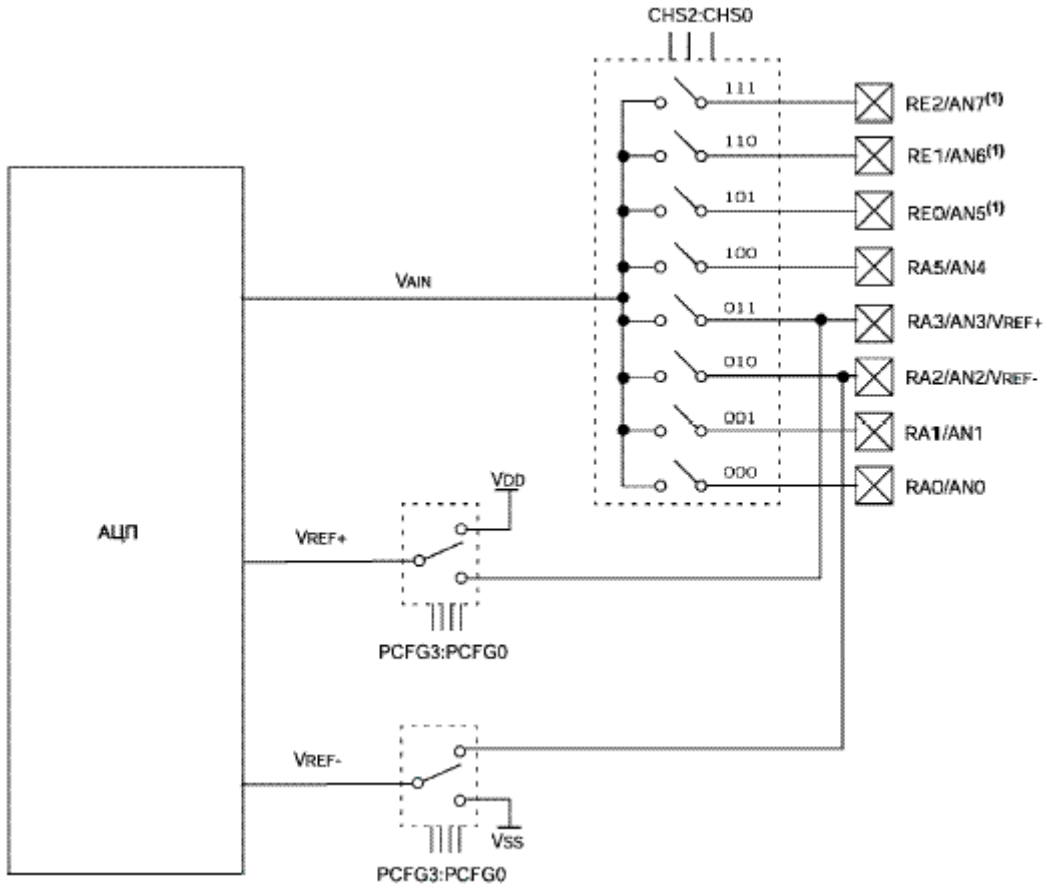


Рисунок 2.13 - Структурна схема модуля АЦП

Після включення і конфігурації АЦП вибирається робочий аналоговий канал. Відповідні біти TRIS аналогових каналів повинні настроювати порт введення/виводу на вхід.

Рекомендована послідовність дій для роботи з АЦП:

1. Настроїти модуль АЦП:
   - Настроїти виводи як аналогові входи, входи VREF або цифрові канали введення/виводу (ADCON1);
   - Вибрати вхідний канал АЦП (ADCON0);
   - Вибрати джерело тактових імпульсів для АЦП (ADCON0);
   - Включити модуль АЦП (ADCON0).
2. Настроїти переривання від модуля АЦП (якщо необхідно):
   - Скинути біт ADIF в '0';
   - Встановити біт ADIE в '1';
   - Встановити біт PEIE в '1';
   - Встановити біт GIE в '1'.
3. Витримати паузу, необхідну для зарядки конденсатора CHOLD-
4. Почати аналого-цифрове перетворення:
   - Встановити біт GO/-DONE в '1' (ADCON0).
5. Чекати закінчення перетворення:
   - Чекати поки біт GO/-DONE не буде скинутий в '0'; або
   - Чекати переривання по закінченню перетворення.
6. Прочитати результат перетворення з регістрів ADRESH: ADRESL, скинути біт ADIF в '0', якщо це необхідно.
7. Для наступного перетворення необхідно виконати кроки починаючи з пункту 1 або 2. Час перетворення одного біта визначається як час TAD. Мінімальний час очікування перед наступним перетворенням повинен складати не менше 2TAD.

2.12 Переривання

PIC16F87X мають 14 джерел переривань. Регістр INTCON містить прапори окремих переривань, біти дозволу цих переривань і біт глобального дозволу переривань.

Якщо біт GIE (INTCON<7>) встановлений в ‘1’, дозволені всі немасковані переривання. Якщо GIE=0, то всі переривання заборонені. Кожне переривання окремо може бути дозволене/заборонене установкою/скиданням відповідного біта в регістрах INTCON, PIE1 і PIE2. При скиданні мікроконтролера біт GIE скидається в '0'. При поверненні з підпрограмі обробки переривання, по команді RETFIE, біт GIE апаратно встановлюється в ‘1’ дозволяючи всі немасковані переривання.

У регістрі INTCON знаходяться прапори наступних переривань: зовнішнього сигналу INT, зміни рівня сигналу на входах RB7:RB4, переповнювання TMR0.

У регістрах PIR1, PIR2 містяться прапори переривань периферійних

модулів мікроконтролера, а в регістрах PIE1, PIE2 відповідні біти дозволу переривань. У регістрі INTCON знаходиться біт дозволу переривань від периферійних модулів.

При переході на підпрограму обробки переривань біт GIE апаратно скидається в '0', забороняючи переривання, адреса повернення з підпрограми обробки переривань поміщається в стек, а в лічильник команд PC завантажується вектор переривання 0004h. Джерело переривань може бути визначене перевіркою прапорів переривань, які повинні бути скинуті програмно перед дозволом переривань, щоб уникнути повторного виклику.

Для зовнішніх джерел переривань (сигнал INT, зміни рівня сигналу на входах RB7:RB4) час переходу на підпрограму обробки переривань складатиме 3-4 машинних цикли. Точний час переходу залежить від конкретного випадку, він однаковий для 1 і 2-х циклових команд. Прапори переривань встановлюються незалежно від стану відповідних бітів маски і біта GIE.

Примітка. Індивідуальні прапори переривань встановлюються незалежно від стану відповідних бітів маски і біта GIE.

Структурна схема логіки переривань показана на рисунку 2.14.

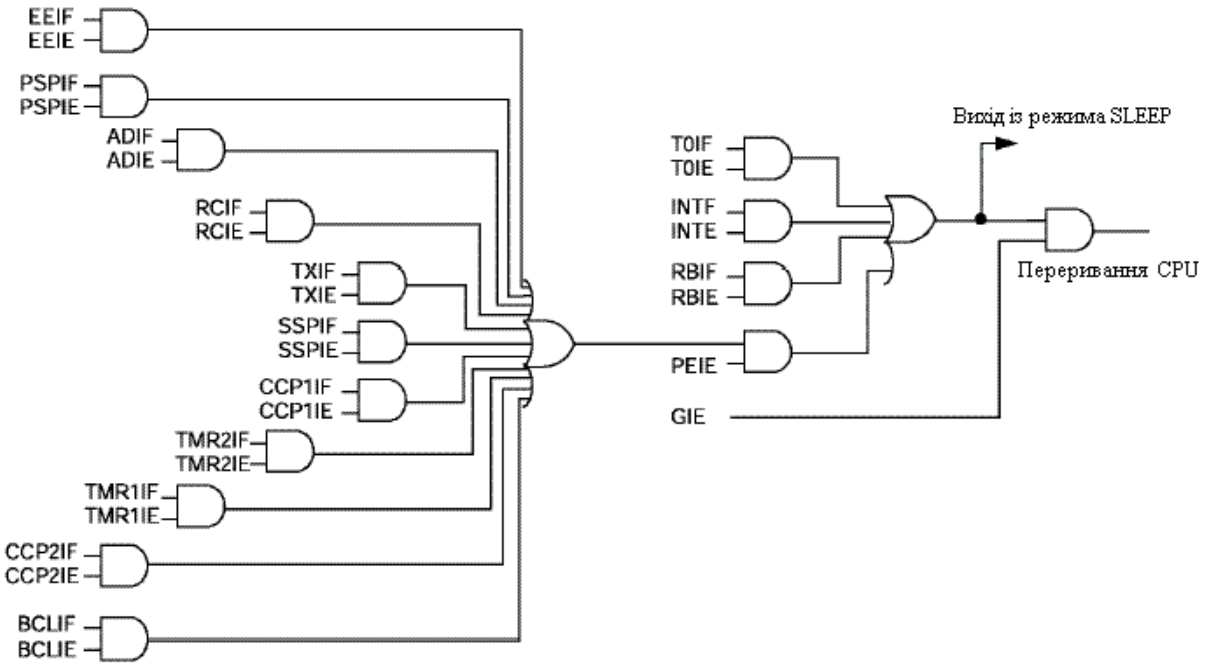


Рисунок 2.14 - Структурна схема логіки переривань

Зовнішнє переривання з входу RB0/INT відбувається: по передньому фронту сигналу, якщо біт INTEDG (OPTION_REG<6>) встановлений в '1'; по задньому фронту сигналу, якщо біт INTEDG скинутий в '0'. Коли активний фронт сигналу з'являється на вході RB0/INT біт INTF (INTCON<1>) встановлюється в '1'. Переривання може бути заборонене

скиданням біта INTE (INTCON<4>) в '0'. Прапор переривання INTF повинен бути скинутий програмно в підпрограмі обробки переривань. Переривання INT може вивести мікроконтролер з режиму SLEEP, якщо біт INTE=1 до переходу в режим SLEEP. Стан бита GIE визначає, переходити на підпрограму обробки переривань після виходу з режиму SLEEP чи ні.

Переповнювання таймера TMR0 (FFh -> 00h) встановлює прапор TOIF (INTCON<2>) в ‘1’. Переривання від TMR0 можна дозволити/заборонити установкою/скиданням біта TOIE (INTCON<5>). Зміна рівня сигналу на входах RB7:RB4 викликає установку прапора RBIF (INTCON<0>). Переривання можна дозволити/заборонити установкою/скиданням біта RBIE (INTCON<4>).

2.13 Сторожовий таймер WDT

Вбудований сторожовий таймер WDT працює від окремого генератора RC, що не вимагає зовнішніх компонентів. Це дозволяє працювати сторожовому таймеру WDT при вимкненому тактовому генераторі (виводи OSC1, OSC2) в SLEEP режимі мікроконтролера. У нормальному режимі роботи при переповнюванні WDT відбувається скидання мікроконтролера. Якщо мікроконтролер знаходиться в SLEEP режимі, переповнювання WDT виводить його з режиму SLEEP з продовженням нормальної роботи. WDT вимкнений, якщо WDTE = 0 в слові конфігурації.

Час переповнювання залежить від температури, напруги живлення V_{DD} і розкиду технологічних параметрів мікроконтролера. Якщо потрібний більший час переповнювання WDT, необхідно програмно підключити переддільника в регістрі OPTION_REG з максимальним коефіцієнтом ділення 1:128.

Примітки:

1. Команди CLRWDT і SLEEP скидають сторожовий таймер і переддільника, якщо він підключений до WDT, відкладаючи скидання пристрою.
2. Команда CLRWDT скидає сторожовий таймер і переддільника, якщо він підключений до WDT, але не змінює коефіцієнт ділення переддільника.

Структурна схема сторожового таймера WDT показана на рисунку 2.15.

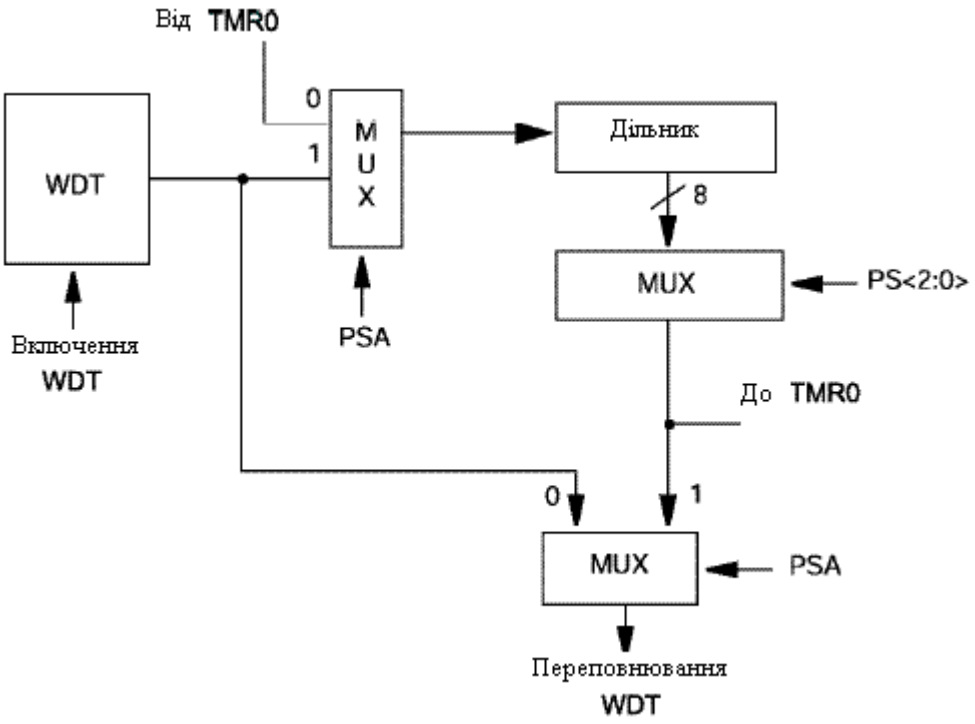


Рисунок 2.15 - Структурна схема сторожового таймера WDT

2.14 Біти конфігурації

Біти конфігурації розташовані в пам'яті програм за адресою 2007h, вони можуть бути запрограмовані в '0' або залишеними в ‘1’. Відмітьте, що адреса 2007h розташована за межами призначеної для користувача пам'яті програм. Фактично, до конфігураційного регістра (область пам'яті 2000h - SFFFh) можна звернутися тільки в режимі програмування мікроконтролера.

Слово конфігурації (адреса 2007h)

CP1	CPO	DEBUG	-	WRT	CPD	LVP	BODEN	CP1	CPO	-PWRTE	WDTE	FOSC1	FOSCO

Біт 13 Біт 0

Біти 13-12: **CP1:CPO:** Біти захисту пам'яті програм

11 = захист пам'яті програм вимкнений
10 = захищена пам'ять програм з адресами 1F00h-1FFFh
01 = захищена пам'ять програм з адресами 1000h-1FFFh
00 = захищена пам'ять програм з адресами 0000h-1FFFh

біт 11: DEBUG: Біт включення режиму внутрішньосхемної відладки
1 = внутрішньосхемна відладка вимкнена, виводи RB6 і RB7 працюють як канали вводу/виводу
0= внутрішньосхемна відладка включена, виводи RB6 і RB7 використовуються відладчиком

біт 10: Не реалізований**:** читається як '1'

біт 9: WRT: Біт дозволу запису в FLASH пам'ять програм
1= дозволений запис в FLASH пам'ять програм через регістри управління EECON
0= заборонений запис в FLASH пам'ять програм через регістри управління EECON

біт 8: CPD: Біт захисту EEPROM пам'яті даних
1= захист пам'яті даних вимкнений
0= захист пам'яті даних включений

біт 7: LVP: Біт дозволу низьковольтного програмування
1= виведення RB3/PGM працює як PGM, режим низьковольтного програмування включений
0= виведення RB3/PGM працює як цифровий порт введення/виводу, виведення HV використовується для програмування мікроконтролера

біт 6: BODEN**:** Біт дозволу скидання по зниженню напруги живлення
1= дозволено скидання BOR
0= заборонено скидання BOR

біт 3: -PWRTE**:** Біт дозволу роботи таймера включення живлення
1= PWRT вимкнений
0= PWRT включений

біт 2**:** WDTE: Біт дозволу роботи сторожового таймера
1= WDT включений
0= WDT вимкнений

біті 1-0: FOSC1:FOSCO: Біти вибору режиму тактового генератора
11= RC генератор
10 = HS генератор
01= XT генератор
00 = LP генератор

Примітки:

1. При стиранні всієї пам'яті мікроконтролера в слово конфігурації записується значення SFFFh.
2. Щоб встановити захист пам'яті програм, всі пари CP1 :CPO повинні мати однакове значення.
3. При виникненні скидання по зниженню напруги живлення (BOR) автоматично запускається таймер PWRT, незалежно від стану бита - PWRTE.

2.15 Система команд

Кожна команда мікроконтролерів PIC16F87X складається з одного 14-розрядного слова, розділеного на код операції (OPCODE), що визначає тип команди і один або декілька операндів, що визначають операцію команди. Повний список команд дивитеся в додатку В. Команди розділені на наступні групи: байт орієнтовані команди, біт орієнтовані команди, команди управління і операцій з константами. Опис полів коду операції дивитеся в таблиці 2.4.

Для байт орієнтованих команд ‘f’ є покажчиком регістра, а ‘d’ покажчиком адресата результату. Покажчик регістра визначає, який регістр повинен використовуватися в команді. Покажчик адресата визначає, де буде збережений результат. Якщо 'd'=0, результат зберігається в регістрі W. Якщо 'd'=1, результат зберігається в регістрі, який використовується в команді.

У біт орієнтованих командах 'b' визначає номер бита що бере участь в операції, а ‘f’ - покажчик регістра, який містить цей біт.

Таблиця 2.4 - Опис полів коду операції

Поле	Опис
f	Адреса файлового регістра (від 0x00 до 0x7F)
W	Робочий регістр (акумулятор)
b	Адреса бита усередині 8-ми бітового регістра
k	Символьне поле, константа або мітка
x	Будь-яке значення(=0 або 1) Компілятор згенерує код з x=0 це потрібно для сумісності зі всіма програмними продуктами Microchip.
d	Вибір де зберігати результат: d=0 (зберігати в W) d=1 (зберігати в f) За умовчанням d=1
label	Ім'я мітки
TOS	Вершина стека
PC	Лічильник команд (програмний лічильник)
PCLATH	Записуваний буфер для старших 5-ти біт PC
GIE	Прапор дозволу глобальних переривань
WDT	Сторожовий таймер
TO	біт Таймауту (Time-out bit)
PD	біт пониження живлення (Power-Down bit)
dest	Призначення, або регістр W або інший регістр вказаний в описі команди
[ ]	Опціонально, тобто необов'язкове використання запису, який поміщений в квадратні дужки
( )	Вміст
->	Занести в
< >	Бітове поле в регістрі

У командах управління або операціях з константами 'к' представляє вісім або одинадцять біт константи або значення літералів.

Система команд акумуляторного типу, ортогональна і розділена на три основні групи:

- Байт орієнтовані команди;
- Біт орієнтовані команди;
- Команди управління і операцій з константами.

Всі команди виконуються за один машинний цикл, окрім команд умови, в яких отриманий дійсний результат і інструкцій , що змінюють значення лічильника команд PC. У разі виконання команди за два машинні цикли, в другому циклі виконується інструкція NOP. Один машинний цикл складається з чотирьох тактів генератора. Для тактового генератора з

частотою 4 Мгц всі команди виконуються за 1мкс, якщо умова істинна або змінюється лічильник команд PC, команда виконується за 2мкс.

На рисунку 2.16 показана форма команд трьох основних груп.

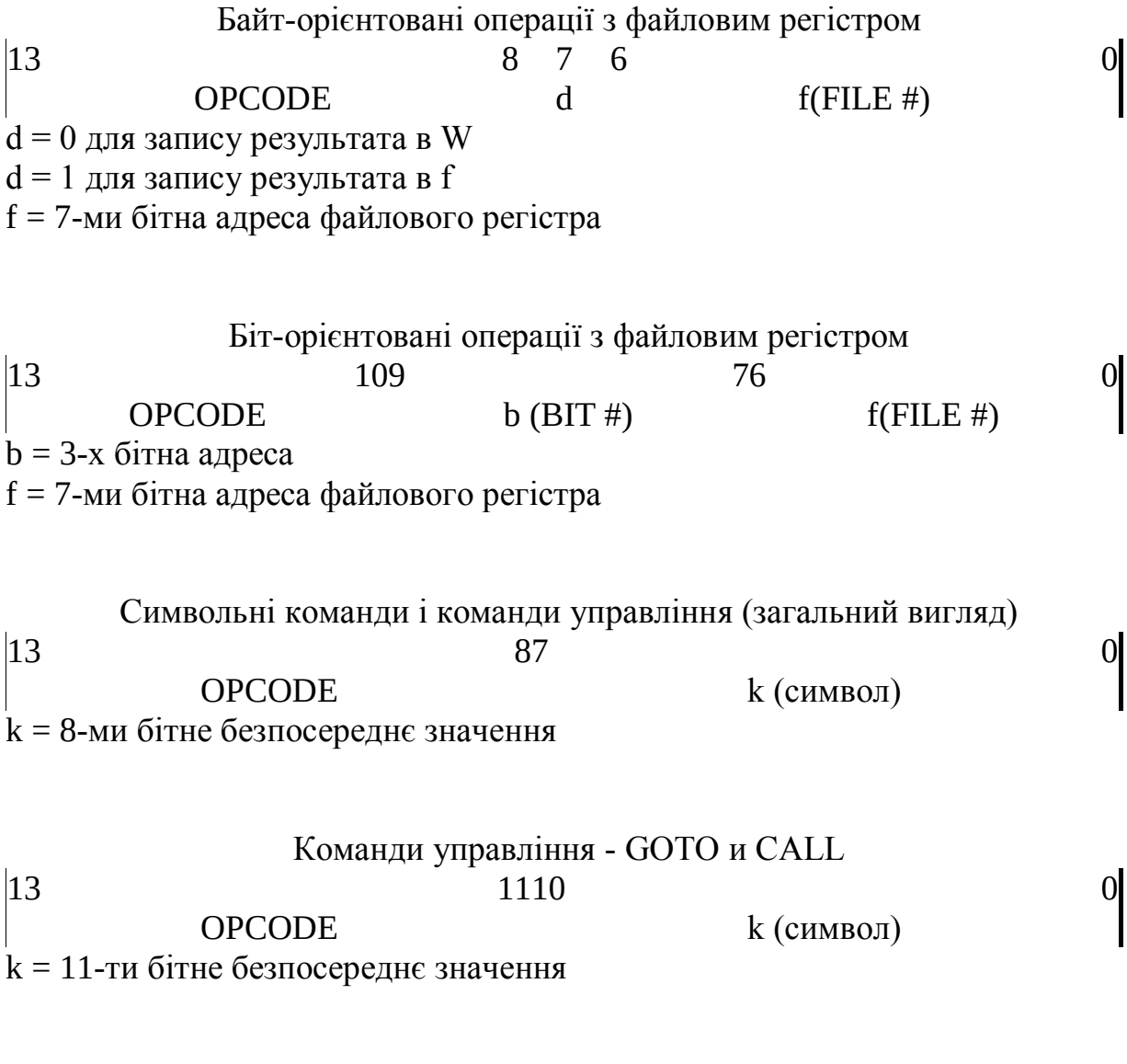


Рисунок 2.16 -Форма команд трьох основних груп

Примітка. Для сумісності програмного забезпечення з наступними версіями мікроконтролерів PICmicro не використовуйте команди TRIS і OPTION.

3 ПРОГРАМУВАННЯ PIC – МІКРОКОНТРОЛЕРІВ

Технологія розробки і відладки робочих програм для ОМК PIC відрізняється від традиційної тільки набором інструментальних засобів. Написання початкового тексту програми можливе на одній з наступних мов: Асемблері, Макроасемблері і СІ. На цьому етапі можуть використовуватися будь-які текстові редактори. Для компіляції програм може бути використаний відповідний компілятор з мови СІ або Асемблер MPASM . Відладка програм може бути здійснена з використанням програмних симуляторов MPSIM або внутрішньосхемних емуляторів реального часу PICMASTER або ICE PIC . Запис відладженої програми в ПЗУ ОМК здійснюється за допомогою спеціальних програматорів, наприклад, типу PIC START, KOM PIC, PIC LAB-16 і т.п..

Для цих же цілей можна скористатися інтегрованими середовищами picDesigner або MPLAB IDE 6.50, які є могутні пакети інструментальних засобів аналогічні вказаним вище.

Розглянемо детальніше ці питання для випадку написання робочих програм на Асемблері.

3.1 Правила запису програм на мові Асемблера

Початковий текст програми на мові асемблера має певний формат. Кожна команда (і псевдокоманда) є рядком чотирьохланкової конструкції:

МІТКА ОПЕРАЦІЯ ОПЕРАНД(и) КОМЕНТАР

Ланки (поля) можуть відділятися одна від одної довільним числом пропусків. Порядок і позиція полів важливі. Так, мітки повинні починатися в першому стовпці. Операція (мнемоніка команди) може починатися в другому стовпці або поза ним.

Операнди слідують за мнемонікою команди. Коментарі можуть слідувати за операндами, мнемонікою або мітками, і можуть починатися в будь-якому стовпці. Максимальна ширина стовпця - 255 символів. Один або більшу кількість пропусків повинен відокремити мітку і мнемоніку команди, і мнемоніку і операнд(и). Операнди повинні відділятися комою.

Наприклад:

```
;
; Приклад фрагмента початкової програми “Ініціалізація МК“
BEGIN

        MOVLW   INITA     ; Завантаження в робочий регістр W
                          ; значення, привласненого імені
                          ; INITA (значення ,константа або
                          ; число) повинно бути
                          ; привласнено раніше в
                          ; попередніх фрагментах
```

```
                              ; програми!)

MOVWF     TRISA     ; Завантаження значення з робочого
                    ; регістра W в регістр
                    ; управління конфігурацією
                    ; порту A
MOVLW     INITB     ; Завантаження в робочий регістр W
                    ; значення, привласненого імені
                    ; INITB
MOVWF     TRISB     ; Завантаження значення з робочого
                    ; регістра W в регістр управління
                    ; конфігурацією порту B
```

Мітка. У полі мітки розміщується символічне ім'я елементу пам'яті, в якій зберігається відмічена команда або операнд. Мітка є буквено-цифровою комбінацією, що починається з букви. Використовуються тільки букви латинського алфавіту. Асемблер допускає використання в мітках символу підкреслення(_). Довжина мітки може бути від 6 до 32 (наприклад для MPASM) символів. Мітки можуть супроводжуватися двокрапкою (:), пропуском, табуляцією або кінцем рядка.

Як символічні імена і мітки не можуть бути використані мнемокоди команд, псевдокоманд і операторів Асемблера, а також мнемонічні позначення регістрів і інших внутрішніх блоків МК.

Операція. У полі операції записується мнемонічне позначення команди МК або псевдокоманди асемблера, яке є скороченням (абревіатурою) повного англійського найменування виконуваної дії. Якщо є мітка на тому ж самому рядку, то мнемоніка команди асемблера, директиви асемблера і макрозвернення повинні відділятися від цієї мітки двокрапкою або одним або великою кількістю пропусків або міток табуляції.

Операнди. У цьому полі поміщаються операнди (або операнд), які приймають участь в операції. Операнди повинні відділятися від мнемоніки одним або більше пропусків або міток табуляції. Списки операндів (операнди) повинні відділятися комами.

Команди можуть бути без-, одно-, або двооперандними.

Операнд може бути заданий безпосередньо або у вигляді його адреси (прямої або непрямої). Безпосередній операнд представляється числом (MOVLW 0FFh, де символ 0 для Асемблера означає, що це число FFh, а не мітка або, MOVLW B'01010011', де B - позначає двійкове число) або символічним ім'ям (MOVWF DATAPORT).

Використовувані як операнди символічні імена і мітки повинні бути визначені, а числа представлені з вказівкою системи числення, для чого використовується префікс (буква, що стоїть перед числом): B - для

двійкової, Q - для вісімкової, D - для десяткової, H - для шістнадцяткової. Число без префікса за умовчанням вважається десятковим.

Псевдокоманди асемблера. Асемблююча програма транслює початкову програму в об’єктні коди. Хоча транслююча програма бере на себе багато з рутинних завдань програміста, такі як привласнення дійсних адрес, перетворення чисел, привласнення дійсних значень символьним змінним і т.п., програміст все ж таки повинен вказати їй деякі параметри: початкова адреса робочої програми, кінець асембльованої програми, формати даних і т.п. Всю цю інформацію програміст вставляє в початковий текст своєї прикладної програми у вигляді псевдокоманд (директив) асемблера, які тільки управляють процесом трансляції і не перетворяться в коди об’єктної програми.

Псевдокоманда ORG 100H задає асемблеру адресу елементу пам'яті (100H), в якій повинна бути розташована наступна за нею команда прикладної програми.

Псевдокомандою EQU можна будь-якому символічному імені, використовуваному в програмі, поставити у відповідність певний операнд. Наприклад, запис

```
TMR0   EQU   01h
```

приводить до того, що в процесі асемблювання усюди, де зустрінеться символічне ім'я TMR0, воно буде замінено числом 1.

Псевдокомандою END програміст дає асемблеру вказівку про закінчення трансляції.

В результаті трансляції повинна бути отримана карта пам'яті програм, де кожному елементу пам'яті поставлений у відповідність код, що зберігається в ньому.

Нижче приведений фрагмент робочої програми, що містить операнди з різним способом завдання і псевдокоманду END:

```
MOVLW   B'01010101'   ; Завантажити  константу
                      ;01010101
                      ; у регістр W
MOVLW   DATAPORT      ; Записати вміст регістра
                      ; W в регістр DATAPORT
                      ; (визначений раніше як порт
                      ; B)
GOTO    $             ; Безкінцевий цикл (вічний
                      ; цикл для перевірки програми в
                      ; динамічному режимі). Вихід
                      ; тільки по перериванню.
END                   ; Кінець трансляції і закінчення
                      ; програми
```

3.2 Структура робочої програми

Робоча програма для ОМК PIC 16/18 складається з трьох основних секцій:

1.Секція заголовка;

2.Робоча секція;

3.Секція закінчення.

У секції заголовка визначаються логічні імена для всіх використовуємих в програмі ресурсів: портів, бітових і байтових змінних, регістрів. Це початкова частина початкової програми до рядка з виразом ORG 0 (для PIC 16F877). Для інших типів мікроконтролерів сімейств PIC 16/18 адреса початку наступної секції (сегменту) робочої програми може бути іншою (див. організацію пам'яті програм конкретного типу ОМК).

Робоча секція програми починається з виразу ORG 0 (для PIC16F877), який є покажчиком для Асемблера про те, що код наступний за цим виразом починається з нульової адреси пам'яті програм. Приклад цього фрагмента робочої програми для PIC 16F877 приведений нижче:

```
; Робоча секція
;
; Початок виконуємої частини програми
        ORG                     ; Наступна команда буде
                                ; розташована в пам’яті
                                ; програм за адресою 0h
        GOTO    BEGIN           ; Перша команда, яка буде
                                ; виконана процесором
        ORG     100h            ; Наступна команда буде
                                ; розташована за адресою 100h
BEGIN
; Ініціалізація мікроконтролера
; (Конфігурація портів введення/виводу, TMR0, WDT и т.п.)
;
```

Секція закінчення в простому випадку містить тільки псевдо-команду END.

3.3 Приклад написання початкового тексту програми

Нижче приведений приклад написання початкового тексту робочої програми для ОМК PIC 16C877 (16F877):

```
; Приклад початкового тексту програми (Назва програми)
;
LIST P=16C877
```

```
;
;Секція заголовка
;
;Опис операційних регістрів
TMR0        EQU     01h         ; Імені TMR0 привласнено
                                ; значення 01h ( де, 01h-
                                ; адреса  регістра TMRO в
                                ; памяти даних ПД )
PC          EQU     02h
STATUS      EQU     03h
FSR         EQU     04h
;  Опис регістрів введення/виводу
CNTRLPORT   EQU     05h         ; Імені CNTRLPORT
                                ; привласнено значення 05h
DATAPORT    EQU     06h
; Опис комірок ОЗП
SCRATCH     EQU     0Ch         ; Імені SCRATCH
                                ;привласнено значення 0Ch
DIGIT       EQU     D
; Опис бітів регістра STATUS
C           EQU     0h          ; Ідентифікатору (імені)
                                ; С привласнено значення 0
                                ; ( 0-й розряд регістра
                                ; STATUS )
DC          EQU     1h
Z           EQU     2h
PD          EQU     3h
TO          EQU     4h
RP          EQU     5h
; Опис керуючих регістрів
TRISA       EQU     85h         ; Імені TRISA привласнено
                                ; значення 85h ( де, 85h –
                                ; адреса регістра TRISA в
                                ; ПД )
TRISB       EQU     86h
; Опис  слів ініціалізації (констант) для портів введення/виводу
; що визначають призначення кожного розряду регістрів портів
INITA       EQU     B'00000000' ; Імені INITA привласнено
                                ; значення двійкового коду
                                ; 00000000, згідно
                                ; якому всі розряди
                                ; порта А призначені
                                ; виходами
INITB       EQU     B'00000000'
```

```
;
; Робоча секція
; Початок виконуємої  частини робочої програми
                ORG     0
                GOTO    BEGIN
                ORG     100h
; Ініціалізація мікроконтролера
                MOVLW   INITA
                MOVLF   TRISA
                MOVLW   INITB
                MOVWF   TRISB
;
; Виведення дискретних сигналів
                MOVLW   B'01010101'     ;Завантажити01010101 у W
                MOVWF   DATAPORT        ; Записати W у порт B
                GOTO    $               ; Нескінченний цикл
;
; Секція закінчення
;
END
```

Спробуємо на підставі приведеного прикладу пояснити деякі основні особливості і правила написання початкових текстів програм для ОМК PIC на мові Асемблера. Для цього аналізуватимемо і обговорюватимемо рядок за рядком дану програму.

По-перше, всі рядки, що починаються із знаку ";", сприймаються асемблером як коментарі. Перейдемо до виразу TMR0. Ми задали асемблеру, що кожного разу, коли зустрінеться слово TMR0, необхідно підставити значення 01h (01 шістнадцяткове). Слово "EQU" означає рівність. Таким чином, ми привласнили TMR0 значення 1h. Ви можете використовувати 01h кожного разу, коли хочете адресувати регістр TMR0, але це значно складніше відладжувати, оскільки Ви повинні будете весь час пам'ятати, що 01h означає TMR0. У Вас можуть існувати і дані, рівні 01h. Використання символьних імен усуває двозначність і дозволяє полегшити читання початкового тексту. Ви також можете бачити вирази для визначення регістрів PC, STATUS і FSR. Ім'я PC відповідає регістру з адресою 02h, ім'я STATUS відповідає регістру з адресою 03h, ім'я FSR - регістру з адресою 04h і так далі. Ми також задали імена для портів введення/виводу, CNTRLPORT (05h) і DATAPORT (06h).

Осередки ОЗП також можуть мати імена. Ми вибрали імена "SCRATCH" для осередку з адресою 0Ch і "DIGIT" для осередку з адресою 0Dh.

Якщо Ви прочитаєте до кінця цей початковий текст програми, то побачите, що ми ніде не використовуємо PC безпосередньо, хоча це ім'я і

визначено. У цьому немає помилки - можна визначати імена і потім не використовувати їх, хоча, звичайно, не можна використовувати ім'я, якщо воно не було заздалегідь визначене. Не дуже піклуйтесь про це - робота асемблера якраз і полягає в перевірці тексту на дотримання всіх правил, і Ви отримаєте повідомлення про помилки, якщо щось не відповідатиме.

Ви можете не тільки іменувати регістри, але і окремі біти усередині регістрів. Зверніть увагу на секцію, задаючу регістр STATUS. Символу C привласнено значення 0h, оскільки C або CARRY, це нульовий біт слова стану STATUS. Кожного разу, коли ми повинні будемо перевірити біт CARRY (біт 0), ми користуватимемося заздалегідь певним символом "C". Кожного разу, коли ми захочемо звернутися до біта 2, або біту ZERO, ми використовуватимемо символ "Z" замість 02h. Ви можете визначити повну структуру бітів регістра, навіть якщо Ви потім не все з них використовуватимете.

Тепер нам стало ясно, як описуються регістри, і ми можемо перейти до виконуваного коду. Перед тим, як почати виконуваний код, ми повинні задати вираз ORG 0. Це покажчик для Асемблера, що код, наступний за цим виразом, починається з нульової адреси ППЗП. Вираз "ORG" використовується для розміщення сегментів коду по різних адресах в межах розмірів ППЗП. Ще один вираз ORG знаходиться перед міткою BEGIN, що має адресу 100h, як задано виразом ORG 100h. Виконуваний код повинен закінчуватися директивою END, що означає, що за цією директивою відсутні виконувані команди.

При включенні живлення PIC16F877 переходить на адресу 0000h. Перша інструкція, яка буде виконана процесором, це команда GOTO BEGIN, яка передасть управління на адресу 100h і подальша робота продовжиться з цієї адреси. BEGIN - це вибиране користувачем ім'я мітки (мітки завжди повинні починатися з першої позиції рядка), яке Асемблер використовує для адресного посилання. В процесі роботи Асемблер визначає розташування мітки BEGIN і запам'ятовує, що якщо це ім'я буде зустрінуте ще раз, замість нього буде підставлена адреса мітки. Команди CALL і GOTO використовують мітки для посилань в початковому тексті.

Тепер подивимося на наступні команди, що виконуються процесором. Команда MOVLW INITA завантажує в робочий регістр W значення, привласнене імені INITA. Це значення задане в заголовку і рівне B'00000000', тобто 00h. Символи B' означають, що дані задані в двійковому форматі. Можна було б написати в цьому ж місці 0 (десятковий) або 0h (шістнадцятковий) і отримати той же самий результат. Двійкове уявлення зручніше використовувати в тих випадках, коли передбачається операція з бітами в регістрі.

Наступна команда MOVWF TRISA завантажує значення з робочого регістра W в регістр управління конфігурацією порту A TRISA. Завдання 0 в розряді цього регістра визначає, що відповідний розряд порту A є виходом. У нашому випадку всі розряди порту A встановлюються

виходами. Якби ми захотіли, наприклад, встановити молодший розряд порту А як вхід, ми б задали в секції описи регістрів значення INITA рівним B'00000001'. Якщо по ходу роботи програми нам потрібно буде перевизначати призначення окремих розрядів портів, наприклад, при двонаправленій передачі, то найзручніше задати всі необхідні слова конфігурації в секції опису, як ми зробили для INITA і INITB.

Наступні дві команди MOVLW INITB і MOVWF TRISB визначають конфігурацію порту В. Ми могли б зекономити і не писати команду MOVLW INITB, оскільки в нашому випадку INITB також рівне 0h. Проте ми не стали цього робити, оскільки це може привести до важко виявлених помилок, якщо згодом нам потрібно буде змінити призначення якого-небудь одного розряду. Замість того, щоб змінити тільки один розряд в одному порту, зміняться два розряди з однаковим номером в двох портах. Тому поки програма не закінчена, таку економію робити не бажано, хоча в кінці, на етапі оптимізації коду, такі повтори можна видаляти.

Ось тепер по суті ми тільки підійшли до аналізу основної частини початкового тексту робочої програми – "Виведення дискретних сигналів".

У цій частині програми ми використовували всього три команди:

```
MOVLW   k
MOVWF   f
GOTO    k.
```

Команда MOVLW завантажує байтовий літерал або константу в робочий регістр W. Наступна команда MOVWF пересилає байт з робочого регістра W в заданий регістр f. Команда GOTO передає управління на адресу k. Таким чином, ця частина програми записує в робочий регістр W значення 01010101 і потім видає його вміст на порт В.

Директива асемблера "$" означає поточне значення програмного лічильника (PC). Тому команда GOTO $ означає перехід туди, де ми в даний момент знаходимося. Такий цикл нескінченний, оскільки не існує способу (окрім переривання) вийти з нього. Команда GOTO $ часто застосовується для зупинки коду при відладці.

Якби до всіх виводів порту В були, наприклад, підключені світлодіоди, то після запуску розглянутої програми ми побачили б свічення тільки чотирьох з них.

3.4 Перетворення початкового тексту робочої програми у об'єктний модуль

Написанням тексту програми закінчується перший етап розробки прикладного програмного забезпечення - "від постановки завдання до початкової програми" і починається наступний - "від початкової програми до об'єктного модуля".

Для простих програм об'єктний код може бути отриманий уручну (ручна трансляція). Проте для больш складних програм потрібні спеціальні

засоби автоматизації підготовки програм. Зазвичай такі засоби використовують великі об'єми пам'яті і широкий набір периферійних пристроїв, через що вони не можуть бути резидентними, а використовуються тільки в кросс-режимі на універсальних ЕОМ.

У мінімальний склад програмного забезпечення кросс-засобів входять:

- системна програма для введення початкового тексту прикладної програми, його редагування і запису на зовнішній носій інформації - так званий редактор текстів(EDIT);
- програма-транслятор, що забезпечує перетворення початкового тексту прикладної програми в об'єктний модуль (ASM, PASM, MPASM).

Могутніші кросс-засоби припускають наявність редактора зовнішніх зв'язків (LINK), що дозволяє включати в програму модулі, розроблені незалежно один від одного, і програму, що забезпечує настроювання переміщуваних програмних модулів на абсолютні адреси (LOCATE).

Для трансляції початкового тексту програми необхідно викликати танслятор, вказавши йому файл з початковим текстом, місце розміщення об'єктного коду, а також умови формування і виведення лістингу.

Всі виявлені в процесі трансляції помилки виправляються в початковому тексті прикладної програми (це відноситься і до помилок, виявлених на етапі відладки). Для цього необхідно знов викликати редактор тексту і здійснити редагування початкового тексту програми, а потім виконати повторну трансляцію.

Якщо початковий текст прикладної програми не мав зовнішніх посилань і містив директиву ORG, то після успішного завершення трансляції етап розробки програмного забезпечення "від початкової програми до об'єктного модуля" можна вважати закінченим.

Для ОМК сімейств PIC 16/18 також існує інтегроване середовище для розробки робочих програм MPLAB IDE 6.50, у складі якої є всі перераховані вище засоби. Існують і окремі програми-транслятори з Асемблера PASM і MPASM.

Розглянемо трохи докладніше процес перетворення початкового тексту робочої програми для ОМК PIC 16/18 з використанням транслятора MPASM.

3.5 Використання програми-транслятора MPASM

3.5.1. Запуск транслятора

Для того, щоб запустити транслятор необхідно вибрати курсором MPASM.EXE і натиснути "Введення". На екрані з'явиться меню, за допомогою першої опції якого можна вибрати файл з початковим текстом ("власне ім'я".asm), якщо він існує в поточному каталозі. Для цього

необхідно вибрати перший пункт і натиснути "Введення", у вікні, що з'явилося, вибрати файл і знову натиснути "Введення".

Друга опція дозволяє вибрати тип процесора. Натискаючи введенням на цей пункт, можна перебирати типи процесорів, поки не буде знайдений потрібний. За допомогою решти пунктів можна вибрати: створювати обьектный файл чи ні, формат HEX-файлу і так далі.

Для трансляції досить використовувати перші дві опції, остальні за умовчанням. Для запуску транслятора необхідно натиснути F10.

Аналогічні дії можна провести, написавши в командному рядку DOS наступну команду:

<MPASM ІМ'Я.ASM / P16F**>

і натиснувши "Введення". Тоді трансляція проведеться без входження в меню MPASM.

2.5.2. Результати трансляції

Результатом роботи транслятора є файли з початковим ім'ям і розширеннями HEX, OBJ, LST,ERR.

Файл з розширенням OBJ містить інформацію про значення змінних, описаних в секції заголовка початкового тексту програми, і є перемістимим об'єктним модулем.

Файл лістингу з розширенням LST містить інформацію про описані змінні, про адресацію програми і використовується на стадії відладки відтрансльованої програми за допомогою симулятора.

Файл з розширенням HEX містить шістнадцяткові коди команд процесора (мікроконтролера), які використовуються для запису програми в пам'ять програм мікроконтролера за допомогою програматора.

У файлі з розширенням ERR містяться відомості про допущені помилки і некоректні записи, зроблені в програмі. Файл має наступний формат:

<ключове слово (warning, message, error)> [<номер помилки>]<шлях до файлу> <номер рядка> <коментар до помилки>

Ключові слова Warning і Message містять рекомендації і повідомлення, але не є помилками і не підлягають обов'язковому виправленню.

2.5.3 Особливості використання транслятора MPASM

При використанні транслятора MPASM з версіями до 3.11 необхідно враховувати те, що ці програми мають друкарські помилки в командах операцій над регістрами виду MOVF reg. Ці транслятори за умовчанням

проводять підстановку, що позначає перенесення результату операції в регістр W.

Окрім цього, ранні версії транслятора MPASM некоректно працюють з другою сторінкою пам'яті ОМК PIC 16F8**. Тому, для коректної роботи програм необхідно безпосередньо самому програмістові перемикати сторінки шляхом установки або скидання 6-го біта в регістрі OPTION.

3.6 Відладка робочих програм

Після отримання об’єктного коду робочої програми неминуче наступає етап відладки, тобто встановлення факту її працездатності, а також виявлення (локалізації) і усунення помилок. Без цього етапу розробки ніяке програмне забезпечення взагалі не має права на існування. Відладка робочих програм є окремим складним завданням, яке майже не піддається формалізації і вимагає для свого виконання високого професіоналізму і глибоких знань розробника.

Зазвичай відладка робочої програми здійснюється у декілька етапів. Прості (синтаксичні) помилки виявляються вже на етапі трансляції. Далі необхідно виконати:

- автономну відладку кожної процедури в статичному режимі, що дозволяє перевірити правильність обчислень, що проводяться, правильність послідовності переходів усередині процедури (відсутність "зациклення") і т.п.;
- комплексну відладку робочої програми в статичному режимі, що дозволяє перевірити правильність алгоритму управління (по послідовності формування керуючих дій);
- комплексну відладку в динамічному режимі без підключення об’єкта для визначення реального часу виконання програми і її окремих фрагментів.

Слід мати на увазі, що автономна відладка окремих модулів настільки простіша і ефективніша за відладку всієї робочої програми, що переходити до етапу комплексної відладки доцільно тільки після вичерпання всіх засобів автономної відладки.

Вищеперелічені етапи відладки здійснюються зазвичай з використанням крос-систем (наприклад, MPLAB IDE для ОМК PIC).

До складу крос-систем входять програми-відладчики (узагальнене ім'я - DEBUG), що інтерпретують (що моделюють) виконання програм, написаних для МК. Такі програмні імітатори дозволяють ефективно відладжувати обчислювальні процедури, а також алгоритм функціонування контроллера.

Розробникові наданий доступ до будь-якого ресурсу МК, є можливість покомандного і пофрагментного виконання програми і останову по умові, а також підрахунку числа тактів виконання тих або

інших фрагментів програми, ініціювання переривань, дизасемблювання вмісту пам'яті програм і т.п.

Кросс-відладчики дозволяють промоделювати практично всі можливі варіанти роботи програми і тим самим переконатися в її працездатності. На цьому ж етапі можлива перевірка працездатності програми при нештатних ситуаціях в умовах надходження некоректних вхідних дій (для застосувань з підвищенними вимогами по безпеці).

Головним недоліком крос-систем є неможливість прогону програми в реальному масштабі часу, тобто з швидкістю, близькою до швидкості виконання програми в самому МК, а також неможливість комплексування апаратурних і програмних засобів системи, що розробляється. Через ці причини достовірність прикладних програм, відладжених в кросс-режимі, недостатньо велика.

Якнайповніша і комплексна відладка прикладного прогаммного забезпечення спільно з апаратурними засобами контроллера може бути проведена на інструментальному відладчику PICDEM 2 PLUS. Під управлінням МІКРОЕОМ PICDEM 2 PLUS дозволяє проганяти прикладну програму або її окремі фрагменти в реальному темпі, зупиняти виконання програм по багатьом ознакам, робити трасування зовнішніх сигналів МК і системи під час виконання програм. Достовірність програмного забезпечення, відладженого на інструментальній МІКРОЕОМ за допомогою PICDEM 2 PLUS , висока хоч і не рівна одиниці.

У будь-якому випадку для доведення прикладного програмного забезпечення контроллера необхідні комплексні і всесторонні випробування розробленої системи в реальному оточенні і у всіляких режимах.

4 MPLAB IDE

MPLAB IDE - безкоштовне інтегроване середовище розробки для мікроконтролерів PICmicro фірми Microchip Technology Incorporated. MPLAB IDE дозволяє писати, відладжувати і оптимізувати текст програми. MPLAB IDE включає редактор тексту, симулятор і менеджер проектів, підтримує роботу емуляторів (MPLAB-ICE, PICMASTER) і програматорів (PICSTART plus, PRO MATE) фірми Microchip і інших налагоджувальних засобів фірми Microchip і третіх виробників.

Інструментальні засоби, що легко настроюються, тематична допомога, «випадні» меню і «призначення гарячих» клавіш в MPLAB IDE дозволяють Вам:

- отримати код програми;
- спостерігати виконання програми за допомогою симулятора, або в реальному часі, використовуючи емулятор (потрібна апаратна частина);

- визначати час виконання програми;
- проглядати поточне значення змінних і спеціальних регістрів;
- працювати з програматорами PICSTAR і PRO MATE II;
- використовувати систему допомоги по MPLAB IDE.

MPLAB IDE дозволяє Вам створювати початковий текст програми в повнофункціональному текстовому редакторові, легко виконати виправлення помилок за допомогою вікна результатів компіляції, в якому указуються виниклі помилки і попередження.

Використовуючи менеджер проектів можна вказати початкові файли програми, об'єктні файли, бібліотеки і файли сценарію.

MPLAB IDE забезпечує різноманітні засоби симуляції і емуляції виконуваного коду для виявлення логічних помилок. Ось їх основні особливості:

- велика кількість сервісних вікон, щоб контролювати значення регістрів пам'яті даних і виконання інструкцій мікроконтролера;
- вікна початкового коду програми, лістингу програми, коду програми - дозволяють оцінити якість компіляції;
- покрокове виконання програми, система точок зупинки, трасування, призначена для швидкої і зручної відладки вашої програми.

4.1 Засоби розробки MPLAB IDE

MPLAB IDE складається з декількох модулів, що забезпечують єдине середовище розробки.

Менеджер проекту MPLAB

Використовується для створення і роботи з файлами, що відносяться до проекту. Дозволяє одним клацанням «миші» виконати компіляцію початкового тексту, включити симулятор або внутрішньосхемний емулятор і т.д.

Редактор MPLAB

Призначений для написання і редагування початкового тексту програми, шаблонів і файлів сценарію лінкера.

Відладчик MPLAB /CD

Недорогий внутрішньосхемний відладчик для мікро контролерів сімейства PIC16F87X.

MPLAB-SIM симулятор

Програмний симулятор моделює виконання програми в мікроконтролері з урахуванням стану портів введення/виводу.

MPLAB ICE емулятор

Емулює роботу мікроконтролера в масштабі реального часу безпосередньо в пристрої користувача.

MPASM ассемблер/MPLINK линкер/MPLIB редактор бібліотек

MPASM компілює початковий текст програми. MPLINK створює завершальний код програми, зв'язуючи різні модулі отримані з MPASM, MPLAB-C17, MPLAB-C18. MPLIB управляє бібліотеками.

MPLAB-CXX компілятори

MPLAB-C17 і MPLAB-C18 виконують компіляцію тексту програми написаному на мові ANSI C. Складні проекти можуть складатися із частин написаних на мові С і Асемблера.

Програматори PRO MATE і PICSTARTplus

Працюють під управлінням MPLAB IDE і призначені для програмування мікроконтролерів кодом програми, отриманої в результаті компіляції початкових файлів. Програматор PRO MATE може працювати самостійно, без використання MPLAB IDE.

Емулятори MPLAB-ICE, PICMASTER-CE і PICMASTER

Застосовуються для моделювання роботи мікроконтролера в пристрої користувача в масштабі реального часу.

4.2 Створення нового проекту

Для роботи вибираємо контролер PIC16F877. Обмежимося стандартною програмою типу «моргнути світлодіодиком».

Запустіть MPLAB IDE будь-яким зручним для Вас способом.

Створіть на жорсткому диску папку, в якій зберігатиметься Ваша робота. У нашому випадку створимо папку C:\work\led

Скористаємося помічником створення проектів.

Вибираємо Project\Project Wizard.

У першому віконці просто клацаємо «Далі».

У наступному вибираємо мікроконтролер. Як ми домовилися, це буде PIC16F877. Натискаєм «Далі».

На наступному листі пропонується вибрати мову розробки.

У рядку «Active Toolsuite» вибираємо «Microchip MPASM Toolsuite»

Нижче вказаний зміст даного програмного продукту і його розташування.

Натискаємо «Далі»

На наступному листі необхідно вказати назву проекту і місцерозташування папки, в якій зберігатиметься код програми.

Указуємо назву «led» і шлях «c:\work\led»

Для вказівки шляху можна скористатися кнопкою «Browse.»

На наступній сторінці пропонується приєднати існуючі файли до проекту.

Оскільки ми тільки починаємо, то сміливо тиснемо «Далі»

На цьому створення проекту завершується.

Ми бачимо підсумкові дані нашого проекту. Натискаємо «Готово»

4.3 Створення початкового файлу

Для написання програми нам необхідний файл.

Створимо його!
Натискаємо піктограму у вигляді чистого листа.
У нас з'явився новий порожній файл.
Збережемо його в папці c:\work\led «File/Save as.»
Назва – led.asm
Додамо цей файл до проекту.
Для цього клацаємо правою кнопочкою на «Source Files» і виберемо «Add Files.»
Відкриється вікно, в якому необхідно вибрати «main»
Після цього файл led.asm буде доданий до проекту.
Для того, щоб компілятор розумів символьні імена, такі як PORTB, STATUS та інші необхідно підключити файл header.
Це робиться шляхом додавання рядка ***#include P16F877.INC***
На цьому підготовку до написання програми ми закінчили.

Само написання програми – процес творчий і неординарний, ми не на цьому зупинятимемося, а візьмемо готову програму.

```
;================ Початок LED.ASM ================
;Файл: LED.ASM

      list p=16F877       ;Використуємий процесор.
      #include p16f877.inc  ;Заголовочный файл для
                          ;мікроконтролера  PIC16F877.
                          ;Файл розташований в директорії
                          ;встановленої MPLAB-IDE.
       org 0x0000         ;Вектор скидання процесора,
                          ;після скидання програма починає
                          ;виконуватися звідси.
      nop                 ;Цей nop життєво необхідний для
                          ;коректної роботи MPLAB-ICD
      clrf INTCON         ;Перестраховуємося, забороняємо всі
                          ;переривання.
      clrf PCLATH         ;Перестраховуємося, вибираємо Bank 0
                          ;Пам'яті Програм.
      goto Start          ;Обхід вектора переривання і блоку
                          ;підпрограм.

      org 0x0004          ;Вектор переривання.

;********** Початок обробника переривань **********
      ;У цій простій програмі не використовуються переривання
;********** Кінець обробника переривань**********

;***************** Блок підпрограм ***************
      ;У цій простій програмі немає підпрограм
;************* Кінець блоку підпрограм *************
```

```
Start                       ;Тіло програми починається тут.

;********** Початок Ініціалізації процесора ********

        clrf PORTTB         ;Усі виводи PORTB переводимо в '0'

        movlw b'00100000'   ;
        movwf STATUS        ;Вибираемо Bank 1 RAM(адреси 80h - FFh)

        movlw b'00000000'    ;
        movwf TRISB          ;Усі лінії PORTB перемикаємо на вихід

        clrf STATUS    ;Повертаємося у Bank 0 RAM(адреси 00h - 7Fh)

;********** Кінець Ініціалізації процесора *********

Loop    bsf PORTB,2         ;Засвітимо світлодіод
        nop                 ;Тягнемо час...
        nop                 ;...
        nop                 ;...
        bcf PORTB,2         ;Тушимо світлодіод
        nop                 ;Тягнемо час...
        nop                 ;...
        nop                 ;...
        goto Loop           ;Нескінченній цикл.

        END                 ;Кінець початкового коду програми

;============= Кінець LED.ASM =====================
```

4.4 Компіляція початкового файлу

Компіляція початкового файлу може бути виконана декількома способами. Описаний тут метод використовує пункт меню Project > Build . Після вибору вказаного пункту меню початковий текст програми зберігається і запускається програма MPASM. Як тільки компіляція буде завершена, на екрані з'явиться вікно результатів роботи MPASM (Рисунок 4.1).

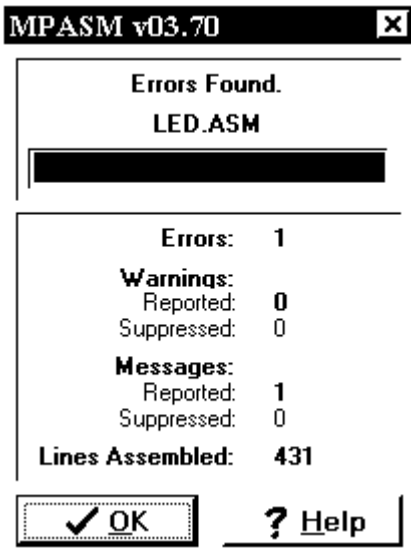


Рисунок 4.1 - Вікно результатів роботи MPASM

Угорі перший рядок указує чи є помилки. У нашому випадку вони є, тому з'являється наступне повідомлення:

- Errors Found і червона лінія 100%;
- Кількість помилок - Errors: 1;
- Кількість описок - Warnings:
  - REPORTED: 0;
  - SUPPRESSED: 0;
- Кількість повідомлень – Messages:
  - REPORTED: 1;
  - SUPPRESSED: 0;
- Усього рядків – (LINES ASSEMBLED): 432.

В вікні результатів компіляції з'явиться текст:

"Clean: Deleting intermediary and output files.
Clean: Deleted file "C:\work\led.HEX".
Clean: Done.
Executing: "C:\Program Files\MPLAB IDE\MCHIP_Tools\Mpasmwin.exe" led.asm /e+ /l+ /x- /p16F877 /c+
Error[113] C:\work\LED.ASM 35 : Symbol not previously defined (PORTTB)
Message[302] C:\work\LED.ASM 41 : Register in operand not in bank 0. Ensure that bank bits are correct.
Halting build on first failed translation as user preferences indicate.
BUILD FAILED: Sun Feb 04 12:49:37 2007"

В програмі навмисне була допущена помилка "PORTTB" (Error[113] C:\work\LED.ASM 35 : Symbol not previously defined (PORTTB)). Повідомлення "Message[302] C:\work\LED.ASM 41 : Register in operand not in bank 0. Ensure that bank bits are correct." нагадує, що регістр TRISB знаходиться не в нульовому банку і попередньо необхідно встановити перший банк.

Подвійне клацання "мишею" на повідомленні про помилку перенесе курсор на рядок в початковому тексті, де була зроблена помилка.

Після виправлення всіх помилок і повторній компіляції на екрані з'явиться вікно результатів з повідомленням про успішну компіляцію (рисунок 4.2) з зеленою лінією.

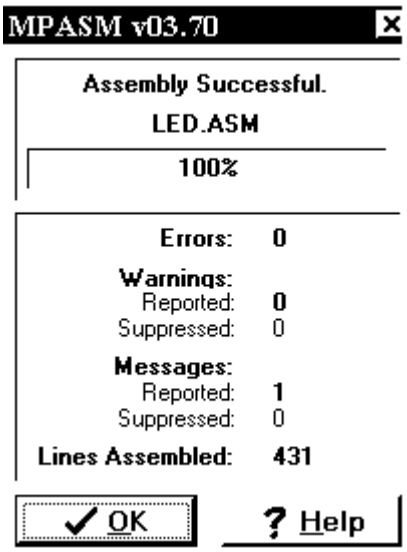


Рисунок 4.2 - Вікно з повідомленням про успішну компіляцію

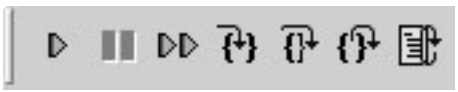
В вікні “Output” останнє повідомлення буде “BUILD SUCCEEDED”.

Тепер можна використовувати симулятор для перевірки роботи програми.

Примітка. При старті компіляції відкриті початкові файли зберігаються на диску.

4.5 Відладка програми

Проте треба переконатися, що програма працює. Як це зробити?
Для цього в MPLAB IDE передбачений процес відладки.
Для його активації необхідно вибрати “Debugger/Select Tool/MPLAB SIM”.
Після цього на панелі інструментів з'явиться така група кнопок:



Призначення кнопок наступне (зліва на право):

- RUN - Запуск програми. Програма працює з великою швидкістю, не відбувається оновлення вікон Watch. Якщо використовується MPLAB ICD2, то дана кнопка запускає програму в реальному часі.
- Halt - Пауза.
- Animate - Анімація. Програма працює швидко, але в покроковому режимі. Швидкість роботи нижча, ніж в першому випадку, проте відбувається новлення всіх вікон.
- Step Into – Крок. Програма виконується в покроковому режимі. Виконується КОЖНА команда. Гаряча клавіша F7.

- Step Over – Крок без підпрограми. Якщо наступною командою є виклик підпрограми, то Ви отримаєте результат роботи цієї підпрограми, а сам процес робота підпрограми залишиться «за кадром». Тобто Ви не потрапите «всередину» підпрограми.
- Step Out – Вихід з підпрограми. Ви вийдете з підпрограми, в якій знаходитеся в даний момент. Ви отримаєте дані роботи підпрограми.
- Reset – Скидання мікроконтролера.

Для зручності користувача передбачено меню View. З його допомогою Ви можете спостерігати:

1. File Registers – регістри ОЗП мікроконтролера.
2. Program Memory – пам'ять програм мікроконтролера.
3. EEPROM – пам'ять ПЗП мікроконтролера (EEPROM).
4. Special Function Registers – регістри спеціального призначення.
5. Watch – спеціальне вікно, в якому можна проглядати будь-які дані, розташовані в ОЗП мікроконтролера. На цьому вікні ми зупинимося трохи нижче.
6. Hardware Stack – стек мікроконтролера. У даному вікні Ви побачите адреси повернення з підпрограм і переривань.
7. Disassembly Listing – Код програми на мові Асемблер. Вельми корисна функція для людей, що пишуть на мовах високого рівня.

Отже, приступимо до відладки.

Нас, перш за все, цікавлять такі регістри:

WREG, PORTB, TRISB.

Додамо їх у вікно WATCH для спостереження за їх станом.

Для цього необхідно:

1. Відкрити вікно Watch – View/Watch
2. У ЛІВОМУ випадному списку вибрати необхідний Вам регістр і натиснути кнопку Add SFR
3. Якщо Ви хочете додати свій власний символ, наприклад регістр counter, то Вам необхідно вибрати в ПРАВОМУ випадному списку і натиснути кнопку “Add Symbol” .
4. Зручнішим і зрозумілішим способом поміщення регістра в watch є «Drag and Drop», тобто просто виділяємо регістр і перетягуємо його у вікно watch.
5. Далі необхідно змінити параметри даних у вікні Watch, для цього необхідно виконати такі дії:
    a. Лівою кнопкою мишки виділити регістр, що цікавить Вас.
    b. Натиснути праву кнопку миші і вибрати «Properties»

На екрані з'явиться вікно «властивості». В результаті цих дій ми повинні отримати всі регістри, що цікавлять нас при відладці програми.

Відкомпілюємо наш проект, для цього необхідно натиснути <F10>

Після компіляції лічильник команд встановиться в 0x00.

Натискаємо <F7> і стежимо за станом мікроконтролера у вікні watch.

Вам може захотітися дізнатися інтервали між якими-небудь подіями. Для цього в середовищі розробки MPLAB IDE існує спеціальне вікно – stopwatch.
Викликати це вікно можна так:
Debugger/Stopwatch

Працювати з даним вікном дуже просто – обнулили, запустили, подивилися. Даним інструментом зручно користуватися у поєднанні з іншою функцією MPLAB IDE – точками останову (breakpoints).
Точка останову, це крапка в якій відбудеться останов процесу симуляції.

Після перевірки працездатності програми необхідно якимсь чином «запихнути» цю програму в мікроконтролер.
Дану роботу виконує пристрій під назвою **Програматор**.
Програматори бувають **фірмові, типу фірмових** і **саморобні**.

Фірмові програматори – це програматори, що випускаються фірмою Microchip.

Вони повністю відповідає специфікації програмування всіх мікроконтролерів, що випускаються. Тривалість, скважність і інші характеристики програмування контролюються процесором програматора, а не комп'ютером. Тобто програматор буде однаково добре працювати на комп'ютерах різної продуктивності.

Фірмові програматори, які випускає фірма Microchip:

Програматор розробника **PICSTART Plus**

Цей програматор, що підтримує всі мікроконтролери фірми Microchip, оснащений панелькою програмування з нульовим зусиллям, що дозволяє просто встановлювати і витягувати мікроконтролер з програматора. Інтерфейс програматора інтегрований в середу розробки MPLAB IDE, що дозволяє програмувати контроллер без яких-небудь додаткових програм. Інтерфейс зв'язку з комп'ютером – RS232.

Внутрішньосхемний відладчик **MPLAB ICD2.**

Цей унікальний пристрій може стати в нагоді розробникові не тільки для відладки програм, але і для «банального» програмування мікроконтролерів. Проте його можливості як програматор не такі великі. Він не підтримує мікроконтролери серії PIC17, підтримує обмежену кількість мікроконтролерів PIC16. Проте, даний пристрій дозволяє програмувати всі мікроконтролери серії PIC18, микроконтроллеры серії PIC12 з flash пам'яттю програм, а також підтримує все нові мікроконтролери серії PIC16, наприклад PIC16F628A, PIC16F648A, PIC16F818, PIC16F819, PIC16F877 і т.д.

Інтерфейс даного налагоджувального засобу також інтегрований в середу розробки MPLAB IDE версії 6.xx і вище. Зв'язок з комп'ютером – USB, RS232.

Для 8 і 14 вивідних мікроконтролерів Flash фірма Microchip випустила недорогий модуль **PICkit1 FLASH STARTER KIT**. Він є платою з мікроконтролером, що управляє, і панелькою для 8 (14) вивідних

мікроконтролерів. Також є 8 світлодіодів, потенціометр і кнопочка. Для «продвинутих» користувачів є майданчик для макетування і друкарська плата для збору модуля інтерфейсу RS232.

Інтерфейс даного налагоджувального засобу інтегрований в середу розробки MPLAB IDE версії 6.xx і вище, але є і окрема програмна оболонка, що дозволяє проводити програмування, читання, стирання, верифікацію без участі MPLAB IDE. Також дана оболонка дозволяє відновлювати калібрувальні дані вбудованого тактового генератора.

Для промислового використання фірма Microchip випускає спеціалізовані програматори **PRO MATE II** і **PM3**. Вони призначені для серійного виробництва і розраховані на об'єми до 10 тис. мікроконтролерів в місяць.

Існують менш дорогі програматори. Назвемо їх «**третіх фірм**». Поза сумнівом, їх головний плюс - менша вартість. Вони випускаються не фірмою Microchip, а сторонніми розробниками. Випускаються вони серійно і їх не можна називати «кустарними». Часто, в них використовуються передові технології. Багато хто з них володіє хорошими показниками, і, можливо, нічим не гірше фірмових. Проте ніхто не дає гарантії, що завтра ця фірма не розчиниться в повітрі. В цьому випадку Ви залишитеся наодинці з незрозумілою залозкою, яка програмує далеко не всі мікроконтролери, до того ж Ви втратите можливість подальшого поповнення списку програмованих мікроконтролерів.

Третій клас програматорів – **саморобні.**

Найпростіші з них, в буквальному розумінні цього слова, можна «зібрати на коліні», при цьому використовуючи навісний монтаж. Більш «просунуті» вимагають деяких знань в області схемотехніки. Принципові схеми даних програматорів легко можна знайти в Інтернеті. Також в Інтернеті Ви можете знайти і програми для комп'ютера, що дозволяють працювати з цими програматорами.

Найбільш популярною безкоштовною програмою для програмування мікроконтролерів фірми Microchip (і не тільки мікроконтролерів) є програма ic-prog. Сайт цієї програми **www.ic-prog.com**
На цьому сайті Ви знайдете не тільки дану програму, але і величезну кількість принципових схем програматорів. Найбільш прийнятним програматором є Propic II. Проте, його схему важко назвати простою. Найбільш популярною зв'язкою є програма IC-PROG + «залізо» ProPic II.

4.6 Внутрішньосхемний відладчик MPLAB ICD2

Внутрішньосхемний відладчик (дебаггер) служить для внутрішньосхемної відладки мікроконтролерів. Відладка здійснюється на штатному серійному мікроконтролері, при цьому відладжувана програма записується в штатну FLASH програмну пам'ять мікроконтролера. Для того, щоб функціонував режим внутрішньосхемної відладки, в серійні

зразки мікро контролерів вбудовують спеціальний механізм – ICD (In_Circuit Debugger). Для відладки цей механізм включається, а для серійних виробів жорстко вимкнений в конфігураційному слові мікроконтролера.

Принцип роботи механізму наступний: під час роботи мікроконтролера досягши точки останову або при покроковій відладці відпрацьовується технологічне немасковане переривання і управління передається підпрограмі відладчика (вона непомітно для користувача дописується в останні елементи програмної пам'яті).

Ця підпрограма виконує функцію передачі через дебаггер в комп'ютер стан елементів пам'яті мікроконтролера, а так само змінює їх стан і перемикає режими роботи по команді з комп'ютера. В результаті частина комірок пам’яті програм і регістрів ОЗП стає недоступною для відладжуваної програми і резервується для роботи підпрограми відладчика.

Крім цього, при відладці так само стають недоступними:

– 2 рівні стека (з 31);

– біти порту RB6 і RB7 (для програмування мікроконтролера і управлінням режимами відладки);

– виведення MCLR/Vpp (використовується для програмування)

– режим Low Voltage ICSP Programming примусово вимикається.

Слід зазначити, що обмеження вносяться лише при включеному режимі відладки. Якщо ж програма дуже велика, в цьому випадку можна порекомендувати відладжувати програму частинами, а потім відключити режим відладки і повністю запрограмувати використовуваний мікроконтролер. При цьому знімаються всі обмеження, що накладаються, MPLAB_ICD2, він працює в режимі звичайного програматора.

Таким чином, за допомогою MPLAB_ICD2 можна не тільки написати програму, відладити її на демонстраційній платі або ж пристрої, що розробляється, але і використовувати ICD2 як серійного програматора (у тому числі і внутрішньосхемного). А невисока вартість дебаггера робить його вельми привабливим багатофункціональним налагоджувальним засобом.

Працює MPLAB ICD2 під управлінням безкоштовного універсального середовища розробника MPLAB IDE, яке періодично оновлюється для підтримки нових мікроконтролерів і перевидається на CD_ROM, а так само доступна на сайті www.microchip.com. Причому ICD2 працює як із старими 16_битными версіями MPLAB IDE 5.xx (підтримується робота тільки через RS_232), так і з новими 32_bit версіями MPLAB IDE 6.xx (підтримується як RS_232, так і USB).

Підтримується установка точок останову, перегляд і зміна пам'яті даних і EEPROM. Крім того, ICD2 можна використовувати як внутрішньосхемний програматор, при цьому вбудований захист від перевантажень по струму і напрузі, є діагностичні світлодіоди контролю стану.

Якщо дебаггер підключається до комп'ютера через USB, то немає необхідності використовувати додаткове джерело живлення. Зрозуміло, що для живлення відладжуваної плати він все-таки необхідний. Від відладжуваної схеми дебаггер живитися не може. У разі живлення дебаггера від зовнішнього джерела живлення (підключення через RS_232 або USB) дебаггер здатний видавати на вихід 5В 150мА, що дозволяє живити малопотужні відладжувані схеми.

4.7 Демонстраційно – відладочна плата PICDEM 2 Plus.

Дуже часто основним чинником, що впливає на успіх розробки, є швидкість виходу виробу на ринок. Тому так важливо почати розробку програми якомога раніше, ще до того, як буде розроблене і виготовлене «залізо» макетного зразка. Так само на початку проектування деколи буває неясно, який варіант реалізації того або іншого вузла буде кращім, якому інтерфейсу передачі даних віддати перевгу, і т.п. У цих і інших подібних ситуаціях на допомогу програмістові мікроконтролерних систем приходять налагоджувальні плати. Це вироби, що містять всі необхідні компоненти для роботи мікроконтролера (кола живлення, скидання, тактового генератора). Крім того, на платі містяться периферійні схеми і пристрої (ЖКІ індикатор, світлодіоди, клавіатура, годинник реального часу, периферійні мікросхеми (CAN, RS_232, I2C, SPI і т.д.)), а так само макетне поле, де при необхідності можна спаяти свою частину схеми. Іншими словами, маючи таку плату, програміст має все необхідне для початку розробки і внутрішньосхемної відладки програми.
Налагоджувальна плата PICDEM 2 Plus, демонстраційна плата від Microchip, має ICD_роз’єм, ЖКІ індикатор, звуковий випромінювач і температурний датчик. PICDEM 2 Plus дозволяє розробникові швидко приступити до створення і відладки програм для 18-, 28- і 40 pin FLASH-мікроконтролерів PICmicro.

Електрична схема плати PICDEM 2 Plus приведена в додатку Г.

4.8 Програмування контролера

При програмуванні за допомогою MPLAB ICD2 порядок дій такий:

1. Вибираємо програматор MPLAB ICD2, після чого з’явиться меню:

Призначення кнопок наступне: програмування мікроконтролера, читання, перевірка, стирання, перевірка «чистий?», зв'язок з MPLAB ICD2.

2. З'єднання з MPLAB ICD2 відбувається автоматично, проте якщо воно не відбулося, необхідно натиснути кнопку, MPLAB ICD2.

Клацаємо мишею по першій кнопці і завантажуємо програму, що відкомпілювалася, у відладжуваний контролер. Можливо, потрібно буде на

вкладці параметрів програмування поміняти значення бітів конфігурації (WDT, CP, і ін.). Після успішного запису і звірки можна кликнути по іконі скидання, при цьому на початковому тексті програми з'явиться сіра смужка покажчика поточної команди. Тепер можна запустити покрокову відладку, виконання в реальному часі, спробувати змінити вміст ОЗП/EEPROM, встановити точки останову.

Отже, Ви записали програму в контроллер, але у Вас взагалі нічого не працює або працює з перебоями? Що робити?

1. Можливо, Ви десь допустили помилку в програмному коді.
2. Можливо, Ви допустили помилку в слові конфігурації. Це найбільш типова помилка. Перевірте правильність установки виду тактового генератора (HS, XT, RC). Також, якщо Ви не передбачали використання WDT, відключите його і в слові конфігурації. Перша ознака включеного WDT – постійний перезапуск мікроконтролера.

Декілька коротких рекомендацій:

– Якщо дебаггер успішно програмує кристал, але при спробі скидання і покрокової відладки видає помилку зв'язку, перевірте відповідність типу генератора в конфігураційному слові (HS,XT, RC і тд) реально використовуваному на відладжуваній платі;

– Швидкість роботи покрокової відладки можна збільшити, обмеживши кількість оновлюваних регістрів ОЗП при відладці;

– Якщо Ваша програма невелика, є сенс змінити кінцеву адресу програмної пам'яті на вкладці опцій програмування ICD2. В цьому випадку кожного разу при зміні програми переписуватиметься не вся програмна пам'ять, а лише її частина, що зменшить час програмування мікросхеми.

5 ЛАБОРАТОРНИЙ ПРАКТИКУМ

Лабораторна робота 1

Робота з портами введення/виводу. Вивчення пакета MPLAB

1 Мета

На прикладі мікроконтролера PIC16F877 вивчити режими роботи портів введення/виводу, способи й особливості їх ініціалізації. Розглянути введення/вивід дискретних сигналів. Набути практичні уміння роботи з пакетом MPLAB, засвоїти способи задання зовнішніх впливів на виводи портів мікроконтролера.

2 Завдання по лабораторній

1) Ініціалізація портів введення/виводу.

- Настроїти порт A на вивід (регістр TRISA).

- Настроїти порт B на введення (регістр TRISB).
- Ввід інформації з датчиків дискретних сигналів.
- Увести через порт B число M, відповідно до варіанта завдання.
- Записати введене число в комірку пам'яті даних з адресою A, відповідно до варіанта завдання.

2) Вивід дискретних сигналів із мікроконтролера.

- Вивести молодшу тетраду введеного числа в порт A.
- Вивести старшу тетраду введеного числа в порт A.

Таблиця 5.1 – Варіанти завдань

Варіант		Номер у журналі групи																	
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	M	121	80	54	112	98	100	34	135	211	240	167	99	56	63	73	23	231	35
	A	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
2	M	151	57	213	89	81	190	119	78	83	67	34	144	132	139	21	141	252	84
	A	29	30	31	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
3	M	50	93	43	140	248	97	75	52	69	61	80	139	169	232	210	109	204	45
	A	26	27	28	29	30	31	9	10	11	12	13	14	15	16	17	18	19	20
4	M	37	65	93	86	180	34	48	201	74	80	207	176	18	55	156	238	49	50
	A	23	24	25	26	27	28	29	30	31	9	10	11	12	13	14	15	16	17

Приклад програми. Виконання конфігурації розрядів 7 – 4 порту B на вивід, а розрядів 3 – 0 – на введення інформації.

```
    …
TRISB  EQU   0x86          ; адреса регістру керування портом B
                           ;
       movlw  b’00001111’
              ; | || || || |
              ;       └┘└┘└┘──── розряди 3-0 порту B на введення
              ; └┘└┘└┘──────── розряди 7-4 порту B на вивод
       movf   TRISB        ; конфігурування порту B
       …                   ; продовження програми
```

Приклад програми. Вводу інформації із порту A та запис її в комірку пам'яті даних з адресою 0x10.

```
    …
PORTA  EQU      0x05; адреса регістру порту введення/виводу
MEM    EQU      0x10; адреса регістру у пам'яті даних
     movf  PORTA, 0    ; введення із порту A в регістр W
     movwf   MEM       ; запис в комірку пам'яті даних
     …                 ; продовження програми
```

Приклад програми. Виводу молодшої тетради із комірки пам'яті даних з адресою 0x10 в старші розряди порту B.

```
    …
PORTB  EQU      0x06; адреса регістру порту введення/виводу
MEM    EQU      0x10; адреса регістру у пам'яті даних
      swapf  MEM, 0      ; обмін тетрад місцями та запис у  W
      movwf PORTB        ; вивод із регістру W у порт B
      …                  ; продовження програми
```

3 Порядок виконання роботи

- Запуск пакета MPLAB.
- File –> New – створення нового вікна для набору тексту програми.
- Набір тексту програми.
- File –> Save As ... – збереження тексту програми у файлі з розширенням .asm.
- Options –> Development Mode – вибір типу мікроконтролера (PIC16F877) і режиму (Simulator).
- Project –> Build –.
- Якщо є помилки – виправлення помилок і перехід до компіляції програми.
- Debug –> Simulator Stimulus –> Clock Stimulus ... – задання зовнішніх сигналів на входи портів введення/виводу.
- Window –> Special Function Registers – відкриття вікна регістрів спеціального призначення.
- Debug –> Run –> Reset – скидання мікроконтролера.
- Debug –> Run –> Step – покрокове виконання програми.
- File –> Exit – вихід із програми.

4 Зміст звіту

- Тема
- Мета.
- Індивідуальне завдання.
- Лістинг програми (файл .lst) із докладним коментуванням виконання програми.
- Короткий опис програми.
- Методика тестування програми і вміст .sti файлу.
- Результати виконання програми.
- Висновки.

Лабораторна робота 2

Режими роботи таймера. Сторожовий таймер (WDT)

1 Мета

Вивчити основні режими функціонування таймера, способи й особливості його ініціалізації, варіанти використання і настроювання попереднього дільника, функціонування сторожового таймера.

2 Завдання по лабораторний роботі

2.1 Зробити ініціалізацію таймера, зробити конфігурацію елементів схеми таймера відповідно до варіанта завдання.

2.1.1 Визначити значення біта T0CS (біт 5 у регістрі OPTION).

2.1.2 Визначити значення біта T0SE (біт 4 у регістрі OPTION).

2.1.3 Визначити значення біта PSA (біт 3 у регістрі OPTION).

2.1.4 Визначити значення бітів PS2 – PS0 (біти 2 – 0 у регістрі OPTION).

2.2 Зробити ділення зовнішньої частоти F, що надходить на вивід RA4/T0CKI мікроконтролера PIC16F877 з коефіцієнтом ділення К1. Отриману частоту вивести на вивід RB0 мікроконтролера.

2.3 Оцінити максимально можливу вхідну частоту F.

2.4 Визначити період скидання сторожового таймера при використанні попереднього дільника з коефіцієнтом ділення К2.

2.5 Проаналізувати функціонування сторожового таймера в режимі SLEEP.

3 Варіанти завдань

3.1. Для парних номерів за списком журналу групи рахунковий перепад – фронт імпульсу.

3.2. Для непарних номерів за списком журналу групи рахунковий перепад – спад імпульсу.

3.3. Коефіцієнт розподілу частоти К1 = С + І + N, де С – номер групи, І – індекс групи (1 для «1», 2 для «2» і т.д.), N – номер студента за списком журналу групи.

3.4. Коефіцієнт ділення попереднього дільника К2 = 2n, при n = N mod 8, де N – номер студента за списком журналу групи.

Приклад програми. Конфігурація схеми таймера за наступними вихідними даними: тактування таймера виробляється від зовнішнього джерела тактових імпульсів по задньому фронту із використанням попереднього дільника з коефіцієнтом ділення 1:64.

```
…
movlw       b’00110101’
      ;       | |||||| |
      ;       | ||||└┴┴──── коефіцієнт поперед.дільника 1:64
      ;       | |||└──────── попередній дільник підключений до
      ;       | ||  |          каналу таймера
      ;       | ||  └──────── тактування по задньому фронту
```

```
        ;                 └──── тактування від зовнішнього генератора
        ;               └└──────────── у даній роботі не мають значення
   movwf    OPTION_REG     ; конфігурування схеми таймера
            …              ; продовження програми
```

Приклад програми. Підрахунок 10 тактових імпульсів за допомогою таймера, за умови, що таймер сконфігурований заздалегідь.

```
   …
   bcf    INTCON, GIE   ; заборона переривань
   bcf    INTCON, T0IF  ; скидання прапорця переповнення
                        ; таймера
   movlw          .256-10   ; формування константи для
   movwf          TMR0      ; ініціалізації таймера
m1       btfss        INTCON, T0IF ; очікування переповнення таймера
   goto   m1           ; перехід, якщо таймер не переповнений
   …                   ; продовження програми
```

Зауваження. Імена регістрів TMR0 (адреса 0x01), INTCON (адреса 0x0B), і біт T0IF(2), GIE(7), доступні при підключенні заголовного файлу P16F877.INC, або повинні бути описані в описовій секції програми.

Приклад програми. Визначення скидання мікроконтролера за сигналом сторожового таймера.

```
   …
  btfsc  STATUS,-TO     ; перевірка прапорця спрацьовування WDT
  goto   res            ; перехід, якщо не скидання від WDT
  brfss  STATUS,-PD     ; перевірка прапорця спрацьовування WDT
  goto   WDT_RES    ; перехід на підпрограму обробки
                        ; скидання за сигналом WDT
res
  …                     ; продовження програми
```

Зауваження. Для можливості функціонування сторожового таймера в пакеті MPLAB, необхідно зробити його настроювання: меню Options–> Processor Setup–>Hardware–>WDT Chip Reset Enable (для пакета MPLAB ver. 4.12).

Імена регістру STATUS (адреса 0x03), і біт -TO(4), -PD(3) доступні при підключенні заголовного файлу P16F877.INC, або повинні бути описані в описовій секції програми.

4 Зміст звіту

4.1 Тема.

4.2 Мета.

4.3 Індивідуальне завдання.

4.4 Структурна схема таймера та сторожового таймера.

4.5 Пояснення до схеми.

4.6 Лістинг програми (файл .lst) із докладним коментуванням виконання програми.

4.7 Короткий опис програми.

4.8 Методика тестування програми і вміст .sti файлу.

4.9 Результати виконання програми.

4.10 Висновки.

Лабораторна робота 3

Формування часових інтервалів
Сторінкова організація пам'яті

1 Мета

Вивчити способи формування часових інтервалів різної тривалості, організацію сторінкової пам'яті програм і даних.

2 Завдання по лабораторній роботі

2.1 На виводі RB1 мікроконтролера сформувати імпульс тривалістю t1.

2.1.1 Реалізувати часову затримку заданої тривалості.

2.1.2 Спираючись на знання, отримані в лабораторній роботі 1 на виводі RB1 мікроконтролера сформувати імпульс заданої тривалості.

2.2 На виводі RB2 мікроконтролера сформувати імпульс тривалістю t2.

2.2.1 Реалізувати часову затримку заданої тривалості без використання таймера у вигляді підпрограми і розташувати її за адресою 0x123 у пам'яті програм.

2.2.2 Спираючись на знання, отримані в лабораторній роботі 1 на виводі RB2 мікроконтролера, сформувати імпульс заданої тривалості. Виклик підпрограм робити явно.

2.3 На виводі RB3 мікроконтролера сформувати імпульс тривалістю t3.

2.3.1 Реалізувати часову затримку заданої тривалості з використанням таймера у вигляді підпрограми і розташувати її за адресою 0x234 у пам'яті програм.

2.3.2 Спираючись на знання, отримані в лабораторній роботі 1 на виводі RB3 мікроконтролера сформувати імпульс заданої тривалості. Виклик підпрограм робити неявним чином.

3 Варіанти завдань

3.1 $t1 = [(N \bmod 5) + 1] \bullet t_ц$, де N – номер студента за списком журналу групи, $t_ц$ – тривалість командного циклу.

3.2 $t2 = (C + I \bullet N) \bullet t_ц$, де C – номер групи, I – індекс групи, N – номер студента за списком журналу групи, $t_ц$ – тривалість командного циклу.

3.3 $t3 = 20 \bullet C \bullet I \bullet N \bullet t_ц$, де C – номер групи, I – індекс групи, N – номер студента за списком журналу групи, $t_ц$ – тривалість командного циклу.

3.4 Для парних номерів за списком журналу групи формувати позитивний імпульс на виводах мікроконтролера (_⎾⎿_), а для непарних номерів – негативний імпульс (⎿⎾).

Приклад програми. Виконання затримки без використання таймера:

```
  …
  movlw       CDEL   ;[1] запис константи
  movwf       VDEL   ;[1] у лічильник
m1      decfsz  VDEL, 1     ;[1] зменшення значення лічильника
                            ;[2] якщо VDEL = 0 пропуск наступної команди
  goto  m1      ;[2] перехід, якщо VDEL ≠ 0
  …             ;    продовження програми
```

Зауваження. У квадратних дужках зазначений час виконання кожної команди у машинних циклах. Час виконання фрагмента програми tв при різних значеннях константи CDEL обчислюється таким чином:

при CDEL = 1 $t_в = 1 + 1 + 2 = 4$ такти,

при CDEL = 2 $t_в = 1 + 1 + 1 + 2 + 2 = 7$ тактів.

Таким чином, у загальному випадку затримка буде дорівнювати

$t_з = (3 \bullet CDEL + 1) \bullet t_ц$.

Приклад програми. Виконання затримки із використанням таймера:

```
  …
  ; ініціалізація таймера
  movlw           b'00000000' ;тактування від внутрішнього генератора
                              ;з попереднім дільником при К = 2
  bsf       STATUS, RP0       ;вибір банку 1
  movwf     OPTION_REG        ;запис у регістр OPTION
  bcf       STATUS, RP0       ;вибір банку 0
  …
  bcf       INTCON, T0IF      ;[1] скидання прапорця переповнення
                              ;    таймера
  movlw           CDEL        ;[1] запис константи для
  movwf           TMR0        ;[1] ініціалізації лічильника
m1
  btfss   INTCON, T0IF        ;[1] перевірка біта T0IF у регістрі
                              ;    INTCON
                              ;[2] якщо T0IF = 1 пропуск наступної
                              ;    команди
  goto    m1                  ;[2] перехід, якщо T0IF = 0
```

```
…                       ;  продовження програми
```

Зауваження. У загальному випадку затримка буде приблизно дорівнювати $t_з = [3 + (256 – CDEL) \bullet K] \bullet t_ц$, де K – коефіцієнт ділення попереднього дільника (1, 2, 4, 8, …, 256).

Приклад програми. Неявний виклик підпрограми з адресою 0x345:

```
  …              ; основна програма
  call   proc1   ; виклик підпрограми proc1
  …              ; продовження основної програми

proc1            ; підпрограма proc1
  movlw  0x03    ; запис старших розрядів
  movwf  PCLATH  ; у регістр PCLATH
  movlw  0x45    ; запис молодших розрядів
  movwf  PCL     ; у регістр PCL і перехід за адресою 0x345
  …
  org    0x345   ; розміщення підпрограми з адреси 0x345
  …              ; команди підпрограми
  return         ; повернення з підпрограми в основну
                 ; програму
```

4 Зміст звіту

4.1 Тема.

4.2.Мета.

4.3 Індивідуальне завдання.

4.4 Алгоритми програм і підпрограм.

4.5 Лістинг програми (файл .lst) із докладним коментуванням виконання програми.

4.6 Короткий опис програми.

4.7 Результати виконання програми.

4.8 Висновки.

Лабораторна робота 4

Організація і використання пам'яті даних

1 Мета

Вивчити сторінкову організацію пам'яті даних. Навчитися використовувати режим непрямої адресації комірок пам'яті даних. Вивчити організацію і способи доступу до енергонезалежної пам'яті даних (EEPROM).

2 Завдання по лабораторній роботі

2.1 Використовуючи метод прямої адресації записати в пам'ять даних мікроконтролера PIC16F877 своє прізвище, ім'я та по батькові.

2.2 Використовуючи метод непрямої адресації переписати анкетні дані з пам'яті даних в енергонезалежну пам'ять (EEPROM).

2.3 Виконати запис в комірку енергонезалежної пам’яті даних згідно з варіантом.

2.4 Визначити час запису одного байта в EEPROM.

2.5 Визначити можливість читання даних з EEPROM відразу після початку циклу запису.

3 Варіанти завдань

Варіант 1. В останню комірку EEPROM записати контрольну суму всіх інших комірок пам'яті, що обчислюється шляхом додавання за модулем 2.

Варіант 2. В останню комірку EEPROM записати контрольну суму всіх інших комірок пам'яті, що обчислюється шляхом додавання по модулю 256.

Варіант 3. В останню комірку EEPROM записати максимальне значення з всіх інших комірок пам'яті.

Варіант 4. В останню комірку EEPROM записати мінімальне значення з всіх інших комірок пам'яті.

Варіант 5. В останню комірку EEPROM записати середнє значення з всіх інших комірок пам'яті.

Приклад програми. Очищення 10 байтів пам'яті даних, починаючи з адреси 0x0C, використовуючи метод непрямої адресації:

```
    …
    movlw  0x0C       ; встановлення початкової адреси
    movwf  FSR   ; ініціалізація покажчика
    movlw  10    ; ініціалізація лічильника CTR значенням 10
    movwf  CTR
m1  clrf   INDF  ; очищення комірки пам'яті даних
    incf   FSR   ; збільшення значення покажчика
    decfsz CTR, 1 ; зменшення значення лічильника
    goto   m1    ; перехід, якщо не остання комірка
    …            ; продовження програми
```

Зауваження. Лічильник CTR не повинен розташовуватися в комірках пам'яті, що очищаються. Імена регістрів FSR (адреса 0x00) і INDF (адреса 0x04) доступні при підключенні заголовного файлу P16F877.INC або повинні бути описані в описовій секції програми.

Приклад програми. Читання даних з комірки пам'яті EEPROM з адресою 0x10 у регістр W:

```
    …
    bcf    STATUS, RP0       ; вибір банку 0
    movlw 0x10          ; визначення адреси комірки пам'яті
```

```
                    ; EEPROM
    movwf EEADR
    bsf     STATUS, RP0     ; вибір банку 1
    bsf     EECON1, RD      ; строб читання
    bcf     STATUS, RP0     ; вибір банку 0
    movf    EEDATA, W ; запис у регістр W результату
                    ; читання
    …
```

Приклад програми. Запис значення регістру W у комірку пам'яті EEPROM з адресою 0x10:

```
    …
    bcf     STATUS, RP0     ; вибір банку 0
    movwfEEDATA     ; дані для запису
    movlw 0x10      ; визначення адреси комірки пам'яті
                    ; EEPROM
    movwf EEADR
    bsf     STATUS, RP0     ; вибір банку 1
    bsf     EECON1, WREN    ; дозвіл запису
    bcf     EECON1, EEIF    ;- скидання прапорця закінчення
                    ;  запису в EEPROM
    bcf     INTCON, GIE     ;+ заборона переривань
    movlw   0x55    ;+ обов'язкова послідовність команд
    movwf   EECON2  ;+
    movlw  0xAA     ;+
    movwf   EECON2  ;+
    bsf     EECON1, WR      ;+ строб запису
m1  btfss   EECON1, EEIF    ;- очікування закінчення запису в
                    ;  EEPROM
    goto    m1      ;-
    bcf     STATUS, RP0     ; вибір банку 0
    …
```

Зауваження. Послідовність команд, що відзначені символом “+” є обов'язковою. Команди, що відзначені символом “–” є необов'язковими у випадку однократного запису. Імена регістрів STATUS (адреса 0x03), EEDATA (адреса 0x08), EEADR (адреса 0x09), INTCON (адреса 0x0B), EECON1 (адреса 0x88), EECON2 (адреса 0x89), і біт RP0(5), RD(0), WR(1), WREN(2), EEIF(4), GIE(7) доступні при підключенні заголовного файлу P16F84.INC або повинні бути описані в описовій секції програми.

4. ЗМІСТ ЗВІТУ

4.1. Тема.

4.2. Мета.

4.3. Індивідуальне завдання.

4.4. Алгоритми програм і підпрограм.

4.5. Лістинг програми (файл .lst) із докладним коментуванням виконання програми.

4.6. Короткий опис програми.

4.7. Результати виконання програми.

4.8. Висновки.

Лабораторна робота 5

Система переривань мікроконтролера PIC16F84.
Власні оброблювачі переривань

1 Мета

Вивчити систему переривань мікроконтролера PIC16F84, способи формування переривань, використання оброблювачів декількох переривань.

2 Завдання по лабораторній роботі

2.1 Написати оброблювачі переривань відповідно до варіанта завдання і номера студента за списком журналу групи.

3 Варіанти завдань

3.1 Визначте джерело переривання відповідно до варіанта завдання і номера студента за списком журналу групи за таблицею.

Джерело переривання	Номер у журналі групи																				
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Переповнення таймера	*	*	*	*													*	*	*		
Передній фронт сигналу на виводі RB0/INT					*	*	*	*									*			*	
Задній фронт сигналу на виводі RB0/INT									*	*	*	*						*			*
Закінчення запису в EEPROM													*	*	*	*			*	*	*
Зміна рівня сигналу на виводі RB4	*				*				*				*								
Зміна рівня сигналу на виводі RB5		*				*				*				*							
Зміна рівня сигналу на виводі RB6			*				*				*				*						
Зміна рівня сигналу на виводі RB7				*				*				*				*					

3.2 Оброблювач переривання по переповненню таймера повинен виконувати.

Варіант 1. Запис у таймер нового значення, інвертування рівня сигналу на виводі RA0.

Варіант 2. Запис у таймер нового значення, інкремент комірки пам'яті даних.

Варіант 3. Запис у таймер нового значення з енергонезалежної пам'яті даних.

Варіант 4. Запис у таймер нового значення, запис цього значення в енергонезалежну пам'ять даних.

3.3 Оброблювач переривання по передньому фронту сигналу на виводі RB0/INT повинен виконувати.

Варіант 1. Інкремент 16-розрядного лічильника в пам'яті даних.

Варіант 2. Інвертування рівня сигналу на виводі RA1.

Варіант 3. Запис поточного значення таймера в комірку пам'яті даних.

Варіант 4. Прийом дискретного сигналу з входу RA2 і видачу його на вивід RA3.

3.4 Оброблювач переривання по задньому фронту сигналу на виводі RB0/INT повинен виконувати.

Варіант 1. Читання 4-розрядного числа з порту A, запис його в комірку пам'яті даних.

Варіант 2. Запис поточного значення таймера в комірку енергонезалежної пам'яті даних.

Варіант 3. Декремент 16-розрядного лічильника в пам'яті даних.

Варіант 4. Інвертування рівня сигналу на виводі RA2.

3.5 Оброблювач переривання по закінченню запису в енергонезалежну пам'ять даних (EEPROM) повинен виконувати.

Варіант 1. Запис поточного значення таймера в комірку пам'яті даних.

Варіант 2. Установку прапорця дозволу запису в EEPROM.

Варіант 3. Інкремент комірки пам'яті даних.

Варіант 4. Декремент комірки пам'яті даних.

3.6 Оброблювач переривання за зміною рівня сигналу на виводі RB4 повинен виконувати.

Варіант 1. Формування позитивного імпульсу тривалістю $ti = N \bullet 10 \bullet t_ц$ на виводі RA0.

Варіант 2. Формування позитивного імпульсу тривалістю $ti = [(N \bmod 10) + 2] \bullet t_ц$ на виводі RA1.

Варіант 3. Формування негативного імпульсу тривалістю $ti = N \bullet 10 \bullet t_ц$ на виводі RA2.

Варіант 4. Формування негативного імпульсу тривалістю $ti = [(N \bmod 10) + 2] \bullet t_ц$ на виводі RA1.

Де N – номер студента за списком журналу групи, $t_ц$ – тривалість командного циклу.

3.7 Оброблювач переривання за зміною рівня сигналу на виводі RB5 повинен виконувати.

Варіант 1. Формування на виводі RA3 імпульсної послідовності з трьох імпульсів негативної полярності, тривалістю і паузою, рівною ti = 5 • $t_ц$ з обов'язковим використанням виклику підпрограми для формування часового інтервалу.

Варіант 2. Формування на виводі RA2 імпульсної послідовності з двох імпульсів позитивної полярності, тривалістю і паузою, рівною ti = 5 • $t_ц$.

Варіант 3. Формування на виводі RA1 імпульсної послідовності з двох імпульсів негативної полярності, тривалістю імпульсів ti = 3 • $t_ц$ і паузою, рівною tп = 7 • $t_ц$.

Варіант 4. Формування на виводі RA0 імпульсної послідовності з двох імпульсів позитивної полярності, тривалістю імпульсів ti = 7 • $t_ц$, і паузою рівною tп = 3 • $t_ц$.

3.8 Оброблювач переривання за зміною рівня сигналу на виводі RB6 повинен виконувати.

Варіант 1. Одночасне інвертування сигналів на виводах RA0 і RA1.

Варіант 2. Обмін рівнів сигналів на виводах RA1 і RA2.

Варіант 3. Прийом дискретного сигналу з виводу RA3 і видачу його на вивід RA0.

Варіант 4. Обмін рівнів сигналів на виводах RA3 і RA2.

3.9 Оброблювач переривання за зміною рівня сигналу на виводі RB7 повинен виконувати.

Варіант 1. Формування на виводі RA0 імпульсної послідовності з трьох імпульсів негативної полярності, тривалістю і паузою, рівною ti = 3 • $t_ц$.

Варіант 2. Формування на виводі RA1 імпульсної послідовності з двох імпульсів позитивної полярності, тривалістю і паузою, рівною ti = 7 • $t_ц$ з обов'язковим використанням виклику підпрограми для формування часового інтервалу.

Варіант 3. Формування на виводі RA2 імпульсної послідовності з трьох імпульсів позитивної полярності, тривалістю імпульсів ti = 7 • $t_ц$ і паузою, рівною tп = 3 • $t_ц$.

Варіант 4. Формування на виводі RA3 імпульсної послідовності з чотирьох імпульсів негативної полярності, тривалістю імпульсів ti = 3 • $t_ц$ і паузою, рівною tп = 5 • $t_ц$.

Приклад програми. Збереження регістрів при виклику переривання і виклик оброблювача переривання за зміною рівня сигналу на виводі RB7.

```
   …
STATUS   EQU   0x03 ; адреса регістру STATUS
PORTB    EQU   0x06 ; адреса регістру PORTB
RBIF                EQU   0x07   ; номер розряду прапорцевого біта
INTCON   EQU   0x0B ; адреса регістру INTCON
W_TEMP              EQU   0x0C   ; адреса регістру збереження W
```

```
STATUS_TEMP         EQU   0X0D ; адреса регістру збереження STATUS
oldportb    EQU   0x0E ; адреса регістру збереження стану
                       ; порту B

int_point   org   0x04   ; вектор переривань
    movwf W_TEMP   ; збереження W
    swapf  STATUS,w  ; збереження STATUS
    movwf STATUS_TEMP
    btfsc   INTCON,RBIF       ; переривання при зміні RB4-RB7?
    call    INTRB7      ; виклик оброблювача переривання
                        ; при зміні сигналу на виводах
                        ; RB4-RB7
    swapf  STATUS_TEMP,W ; відновлення регістру STATUS
    movwf STATUS
    swapf  W_TEMP, F  ; відновлення регістру W
    swapf  W_TEMP, W
    retfie              ; повернення з переривання

INTRB7                  ; оброблювач переривання
                        ; при зміні сигналу на виводах
                        ; RB4-RB7
    movf   PORTB, W  ; читання стану порту B
    xorwf  oldportb, F   ; порівняння з попереднім значенням
    btfss   oldportb, 7   ; зміна RB7?
    goto    NOTRB7    ; немає змін
    …                   ; необхідні дії при зміні RB7
NOTRB7
    movf   PORTB, W  ; читання стану порту B
    movwf  oldportb     ; збереження для наступного
                        ; порівняння
    bcf     INTCON,RBIF       ; очищення прапорця переривання
    return
    …                   ; продовження програми
```

Зауваження. Імена регістрів спеціального призначення доступні при підключенні заголовного файлу P16F84.INC або повинні бути описані в описовій секції програми.

4 Порядок виконання роботи

4.1 Дозволити необхідні переривання, замаскувати інші.

4.2 Сформувати умови для формування відповідних переривань.

4.3 Визначити максимальний час обробки переривання в тактах і в одиницях часу, при тактуванні мікроконтролера від тактового генератора з частотою Fosc = 8 МГц.

5 Зміст звіту

5.1 Тема.

5.2 Мета.

5.3 Індивідуальне завдання.

5.4 Структурна схема системи переривань.

5.5 Пояснення до схеми.

5.6 Лістинг програми (файл .lst) із докладним коментуванням виконання програми.

5.7 Короткий опис програми.

5.8 Результати виконання програми.

5.9 Висновки.

Лабораторна робота 6

Ведення і вивод аналогових сигналів

1 Мета

Вивчити принцип роботи аналого-цифрового перетворювача на прикладі мікроконтролера PIC16C71. Навчитися вводити аналогові сигнали. Розглянути способи виводу аналогових сигналів. Навчитися формувати широтно-модульовані сигнали.

2 Короткі теоретичні відомості

2.1 **Введення аналогових сигналів** виконується за допомогою аналогово-цифрових перетворювачів (АЦП). На рисунку 5.1 наведена залежність одержуваного цифрового коду від вхідного аналогового сигналу для 8–розрядного АЦП.

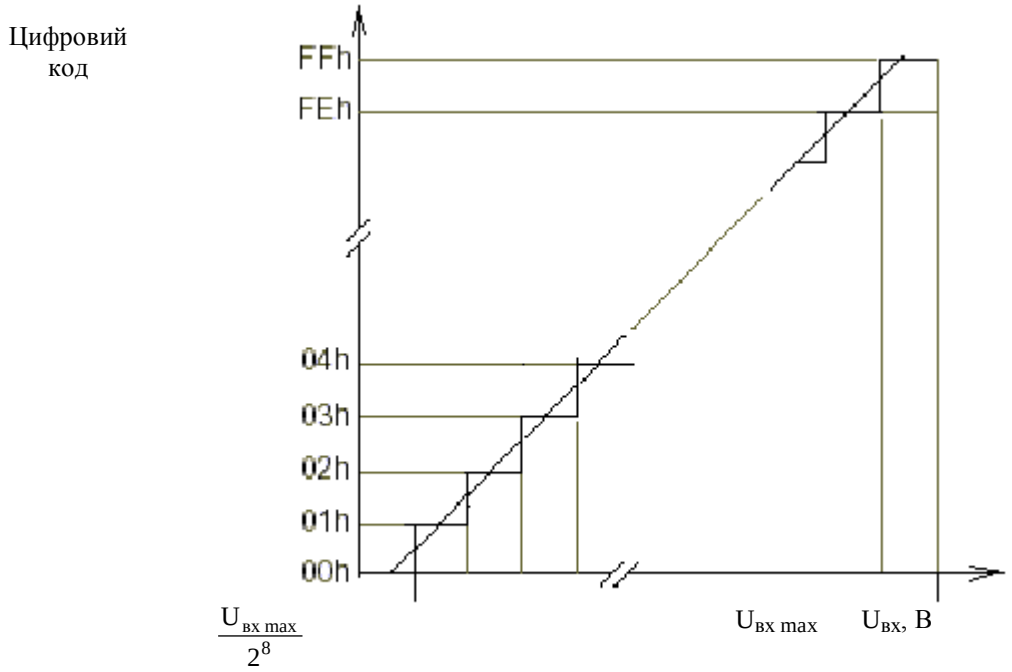


Рисунок 5.1 - Функція перетворення АЦП

Для управління АЦП у мікроконтролері PIC16F877 необхідно використовувати наступні регістри:

- регістр результату (ADRES);
- регістр управління 0 (ADCON0);
- регістр управління 1 (ADCON1);
- регістр управління перериваннями (INTCON).

У регістрі ADRES зберігається результат аналого-цифрового перетворення. Коли перетворення завершене, результат перетворення записується в регістр ADRES, після цього скидається прапорець GO/DONE (ADCON0<2>) і встановлюється прапорець переривання ADIF (ADCON0<1>). Потім необхідно витримати паузу 20 – 30 мкс для розряду ємності схеми вибірки-збереження АЦП (причому, це необхідно повторювати після кожного наступного циклу перетворення). І тільки після цього потрібно установити біт GO/DONE (ADCON0<2>) для початку перетворення.

Алгоритм дій для виконання аналого-цифрового перетворення слідуючий:

1. Здійснити конфігурацію (ініціалізацію) модуля АЦП:
   - вибрати аналогові входи та джерело опорної напруги (біти PCFG2 – PCFG0 у регістрі ADCON1);

- вибрати вхідний канал АЦП (біти CHS2 – CHS0 у регістрі ADCON0);
- вибрати джерело та частоту імпульсів перетворення (біти ADCS1, ADCS0 у регістрі ADCON0);
- включити модуль АЦП (біт ADON у регістрі ADCON0).

2. Настроїти переривання від модуля АЦП (при необхідності):
   - скинути прапор закінчення переривання (біт ADIF у регістрі ADCON0);
   - дозволити переривання від АЦП (біт ADIE у регістрі INTCON);
   - дозволити всі переривання (біт GIE у регістрі INTCON).
3. Витримати паузу 20 – 30 мкс.
4. Почати аналого-цифрове перетворення:
   - установити біт запуску АЦП (біт GO/DONE у регістрі ADCON0).
5. Очікувати закінчення перетворення, наприкінці якого автоматично виконується:
   - скидання біта закінчення перетворення (біт GO/DONE у регістрі ADCON0);
   - формування сигналу (установка біта ADIF у регістрі ADCON0).
6. Отримати результати перетворення:
   - прочитати результат перетворення (регістр ADRES);
   - скинути, якщо необхідно, прапорець закінчення переривання (біт ADIF у регістрі ADCON0).
7. Виконати дії, починаючи з пункту 1 або пункту 2, для виконання наступного перетворення.

2.2 **Вивод аналогових сигналів** виконується за допомогою цифро-аналогових перетворювачів. Сигнал з амплітудою, пропорційною цифровому коду, можна також одержати, використовуючи широтно-імпульсну модуляцію (ШІМ), що представляє собою сигнали змінної тривалості при постійному періоді. Амплітуда такого сигналу, що пройшов через інтегруючий ланцюжок або через фільтр низької частоти (ФНЧ), пропорційна тривалості ШІМ сигналу.

На рисунку 5.2 наведений приклад формування ШІМ сигналу з різною тривалістю $t_{и1}$, $t_{и2}$, $t_{и3}$ і постійним періодом T.

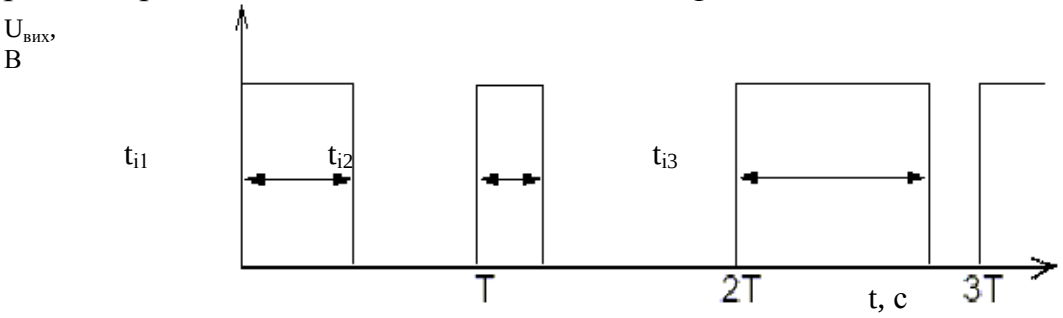


Рисунок 5.2- Форма ШІМ сигналу

На рисунку 5.3 наведений приклад схеми формування аналогового сигналу з ШІМ сигналу із використанням RC ланцюжка у якості ФНЧ першого порядку і повторювача на операційному підсилювачі.

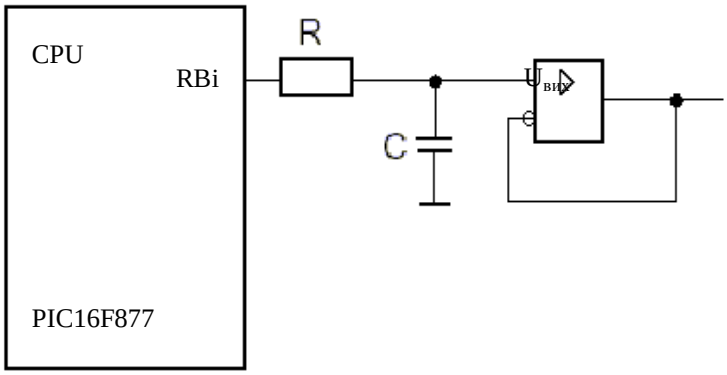


Рисунок 5.3 - Формування аналогового сигналу

Для розрахунку значень R і C звернемося до амплітудно-частотної характеристики (АЧХ) ФНЧ, що представлена на рисунок 5.4.

Ширина спектра вихідного сигналу без використання ФНЧ дорівнює f_{PWM} = 1/T, а при використанні ФНЧ – $f_{ГР}$. Частота $f_{ГР}$ повинна бути значно менше (у 5 і більш разів) частоти f_{PWM}. Тоді, задавши одне зі значень R або C, за формулою RC = 1/(2 π f), де $f = f_{ГР}$ – частота зрізу фільтра, можна знайти інше значення.

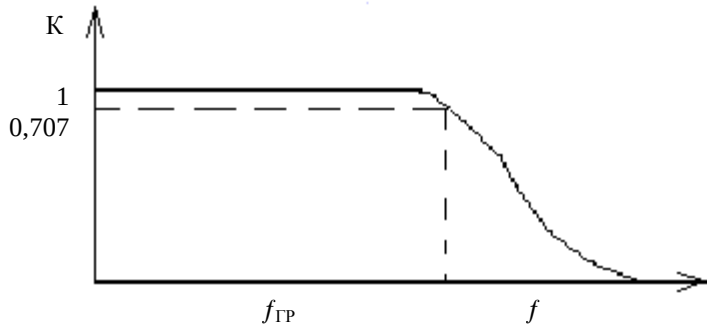


Рисунок 5.4 - Амплітудно-частотна характеристика ФНЧ

3 Завдання по лабораторній роботі

3.1 Введення аналогових сигналів.

3.1.1 Виконати конфігурацію мікроконтролера на вводу аналогових сигналів.

3.1.2 Ввести сигнали з двох аналогових датчиків U1 і U2.

3.1.3 Зробити порівняння введених сигналів.

3.1.4 За результатами порівняння виконати підпрограму згідно варіанту.

3.1.5 Визначити час виконання перетворення АЦП.

3.2 Вивід аналогових сигналів.

3.2.1 На заданому виводі мікроконтролера сформувати сигнал із заданим періодом повторення T і числом рівнів квантування N.

4 **Варіанти завдання 4.1.**

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
U1 < U2	A	B	C	D	E	F	G	H	I	J	A	B	C	D	E	F	D	C	I	J
U1 = U2	I	J	G	H	E	F	D	C	B	J	A	I	C	G	F	E	G	H	I	A
U1 > U2	A	J	C	H	B	D	G	C	I	E	J	B	H	D	E	E	D	H	B	J

Дії підпрограм:

A – інкремент комірки пам'яті даних;

B – декремент 16-розрядного лічильника в пам'яті даних;

C – читання 8-розрядного числа з порту B і запис його в комірку пам'яті даних;

D – запис поточного значення таймера (регістр TMR0) в комірку пам'яті даних;

E – формування позитивного імпульсу тривалістю $t_i = 5 \bullet t_ц$ на виводі RB0, де $t_ц$ – тривалість командного циклу;

F – інкремент 16-розрядного лічильника в пам'яті даних;

G – одночасне інвертування сигналів на виводах RB0 і RB1;

H – декремент комірки пам'яті даних;

I – обмін рівнів сигналів на виводах RB1 і RB2;

J – прийом дискретного сигналу з виводу RB3 і видача його на вивід RB0.

Варіанти завдання 4.2

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Вивід RB	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3
T, мс	2	4	8	16	32	64	2	4	8	16	64	2	4	8	16	32	64	2	4	8
N	256	128	64	32	256	128	64	32	256	128	64	32	256	128	64	32	256	128	64	32

Тактування мікроконтролера виконати від тактового генератора з частотою fOSC = 4 МГц.

Приклад програми. Фрагмент програми, що виконує настроювання АЦП і аналого-цифрове перетворення вхідного сигналу з виводу RA2/AN2.

```
; підключення файлу з описом стандартних констант і значень
    #include p16c71.inc
    …
; Настроювання АЦП
initAD
    bsf    STATUS, RP0   ; вибір банку 1
    movlw                b’00000000’      ; вив.RA3-RA0–аналогові входи
    movwf                ADCON1
    bcf    STATUS, RP0   ; вибір банку 0
```

```
    movlw       b'11010001'     ; RC-генератор для АЦП,
    movwf       ADCON0  ; канал 2, дозвіл АЦП

; Перетворення
Convert
    call    Delay30us       ; затримка 30 мкс
    bsf     ADCON0, GO_DONE     ; запуск АЦП
loop
    btfsc   ADCON0, GO_DONE     ; перетворення закінчене?
    goto    loop            ; продовжити чекання
    movf    ADRES, W        ; результат перетворення в W
    …                       ; продовження програми
```

5 **Моделювання роботи АЦП у пакеті MPLAB**

5.1 Створити текстовий файл із передбачуваними значеннями результатів перетворення АЦП, наприклад adcres.reg

5.2 Відкрити вікно Register Stimulus через меню: Debug->Simulator Stimulus->Register Stimulus->Enable...

5.3 У полі Program Memory Address вказати адресу команди в пам'яті програм (дозволяється і рекомендується вказувати мітку), наприклад, ares.

5.4 У полі Register Address вкажіть адресу або символьне ім'я регістра, наприклад, ADRES.

5.5 Натисніть кнопку Browse... для вказівки файлу стимулу, наприклад, adcres.reg.

5.6 Після скидання мікроконтролера, щораз, коли значення лічильника команд PC співпаде з указаною адресою ares, у регістр ADRES буде занесене значення з файлу adcres.reg. При досягненні кінця файлу adcres.reg підстановка почнеться з початку файлу, тобто буде виконуватися циклічно.

6 Зміст звіту

6.1 Тема.

6.2 Мета.

6.3 Індивідуальне завдання.

6.4 Лістинг програми із докладним коментуванням виконання програми.

6.5 Короткий опис програм.

6.6 Результати виконання програм.

6.7 Висновки.

Література

1. Предко М. Справочник по PIC-микроконтроллерам: Пер. с англ. – М.: ДМК Пресс, 2004.-512с.

Додаток А – Регістри спеціального призначення

Адреса	Им'я	Біт7	Біт6	Біт5	Біт4	Біт3	Біт2	Біт1	Біт0	FOR, BOR
Банк 0										
00h	INDF	Звертання до регістра, адреса якого записана в FSR (не фізичний регістр)								0000 0000
01h	TMRO	Регістр таймера 0								xxxx xxxx
02h	PCL	Молодші біти лічильника команд PC								0000 0000
03h	STATUS	IRP	RP1	RPO	-TO	-PD	Z	DC	C	0001 1xxx
04h	FSR	Регістр адреси при непрямій адресації								xxxx xxxx
05h	PORTA	-	-	Зап. у вих. защіпку PORTA, читання стану вив.PORTA						—0x 0000
06h	PORTB	Запис у вихідну защіпку PORTB, читання стану виводів PORTB								xxxx xxxx
07h	PORTC	Запис у вихідну защіпку PORTC, читання стану виводів PORTC								xxxx xxxx
08h	PORTD	Запис у вихідну защіпку PORTD, читання стану виводів PORTD								xxxx xxxx
09h	PORTE	-	-	-	-	-	RE2	RE1	REO	----- -xxx
OAh	PCLATH	-	-	-	Старші біти лічильника команд PC					---- 0 0000
OBh	INTCON	GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF	0000 000x
0Ch	PIR1	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1I	TMR2I	TMR1I	0000 0000
0Dh	PIR2	-	(5)	-	EEIF	BCLIF	-	-	CCP2I	-r-0 0 — 0
0Eh	TMR1L	Молодший байт 16-розрядного таймера 1								xxxx xxxx
0Fh	TMR1H	Старший байт 16-розрядного таймера 1								xxxx xxxx
10h	T1CON	-	-	T1CKP	T1CKP	T1OSC	T1SYN	TMR1C	TMR1O	—00 0000
11h	TMR2	Регістр таймера 2								0000 0000
12h	T2CON	-	TOUTP	TOUTP	TOUTP	TOUTP	TMR2O	T2CKP	T2CKP	-000 0000
13h	SSPBUF	Буфер приймача MSSP / регістр передавача								xxxx xxxx
14h	SSPCON	WCOL	SSPOV	SSPEN	CKP	SSPM3		SSPM1	SSPMO	0000 0000
15h	CCPR1L	Молодший байт захвату/порівняння /ШІМ CCP1								xxxx xxxx
16h	CCPR1 H	Старший байт захвату/порівняння /ШІМ CCP1								xxxx xxxx
17h	CCP1CON	-	-	CCP1X	CCP1Y	CCP1M	CCP1M	CCP1M	CCP1M	—00 0000
18h	RCSTA	SPEN	RX9	SREN	CREN	ADDE	FERR	OERR	RX9D	0000 OOOx
19h	TXREG	Регистр данных передатчика USART								0000 0000
1Ah	RCREG	Регистр данных приемника USART								0000 0000
1Bh	CCPR2L	Молодший байт захвату/порівняння /ШІМ CCP2								
1Ch	CCPR2H	Старший байт захвату/порівняння /ШІМ CCP2								xxxx xxxx
1Dh	CCP2CON	-	-	CCP2X	CCP2Y	CCP2M3	CCP2M	CCP2M	CCP2M	—00 0000
1Eh	ADRESH	Старший байт результату перетворення АЦП								xxxx xxxx
1Fh	ADCONO	ADCS1	ADCS	CHS2	CHS1	CHSO	GO/-	-	ADON	0000 00-0
Банк 1										
80h	INDF	Звертання до регістра, адреса якого записана в FSR (не фізичний регістр)								0000 0000
81h	OPTION_RE	-RBPU	INTED	TOCS	TOSE	PSA	PS2	PS1	PSO	1111 1111
82h	PCL	Молодші біти лічильника команд PC								0000 0000
83h	STATUS	IRP	RP1	RPO	-TO	-PD	Z	DC	c	0001 1xxx
84h	FSR	Регістр адреси при непрямій адресації								xxxx xxxx
85h	TRISA	-	-	Напрям виводів PORTA						— 11 1111
86h	TRISB	Напрям виводів PORTB								1111 1111
87h	TRISC	Напрям виводів PORTC								1111 1111
88h	TRISD	Напрям виводів PORTD								1111 1111
89h	TRISE	IBF	OBF	IBOV	PSPMODE	Напрям виводів PORTE				0000 -111
8Ah	PCLATH	-	-	-	Старші біти лічильника команд PC					------- 0 0000
8Bh	INTCON	GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF	0000 OOOx
8Ch	PIE1	PSPIE(2)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2I	TMR1I	0000 0000
8Dh	PIE2	-	(5)	-	EEIE	BCLIE	-	-	CCP2IE	-r-0 0 — 0
8Eh	PCON	-	-	-	-	-	-	-POR	-BOR	
8Fh	-	Не реалізовано								-
90h	-	Не реалізовано								-

Продовження додатка А

Адреса	Им'я	Біт7	Біт6	Біт5	Біт4	Біт3	Біт2	Біт1	Біт0	FOR, BOR
91h	SSPCON2	GCEN	ACKSTAT	IACKDT	ACKE	RCEN	PEN	RSEN	SEN	0000 0000
92h	PR2	Регістр періоду таймера 2								1111 1111
93h	SSPADD	Регістр адреси / Регістр генератора швидкості обміну								0000 0000
94h	SSPSTAT	SMP	CKE	D/-A	P	S	R/-W	UA	BF	0000 0000
95h	-	Не реалізовано								-
96h	-	Не реалізовано								-
97h	-	Не реалізовано								-
98h	TXSTA	CSRC	TX9	TXEN	SYNC	-	BRGH	TRMT	TX9D	0000 -010
99h	SPBRG	Регістр генератора швидкості USART								0000 0000
9Ah	-	Не реалізовано								-
9Bh	-	Не реалізовано								-
9Ch	-	Не реалізовано								-
9Dh	-	Не реалізовано								-
9Eh	ADRESL	Молодший байт результату перетворення АЦП								xxxx xxxx
9Fh	ADCON1	ADFM	-	-	-	PCFG3	PCFG2	PCFG1	PCFGO	0 ----- 0000
Банк 2										
100h	INDF	Звертання до регістра, адреса якого записана в FSR (не фізичний регістр)								0000 0000
101h	TMR0	Регістр таймера 0								xxxx xxxx
102h	PCL	Молодші біти лічильника команд PC								0000 0000
103h	STATUS	IRP	RP1	RPO	-TO	-PD	Z	DC	C	0001 Ixxx
104h	FSR	Регістр адреси при непрямій адресації								xxxx xxxx
105h	-	Не реалізовано								-
106h	PORTB	Запис у вихідну защіпку PORTB, читання стану виводів PORTB								xxxx xxxx
107h	-	Не реалізовано								-
108h	-	Не реалізовано								-
109h	-	Не реалізовано								-
10Ah	PCLATH	-	-	-	Старші біти лічильника команд PC					------ 0 0000
10Bh	INTCON	GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF	0000 OOOx
10Ch	EEDATA	Регістр даних, молодший байт								xxxx xxxx
10Dh	EEADR	Регістр адреси, молодший байт								xxxx xxxx
10Eh	EEDATH	-	-	Регістр даних, старший байт						xxxx xxxx
10Fh	EEADRH	-	-	| Регістр адреси, старший байт						xxxx xxxx
Банк 3										
180h	INDF	Звертання до регістра, адреса якого записана в FSR (не фізичний регістр)								0000 0000
181h	OPTION_RE	-RBPU	INTED	TOCS	TOSE	PSA	PS2	PS1	PSO	1111 1111
182h	PCL	Молодші біти лічильника команд PC								0000 0000
183h	STATUS	IRP	RP1	RPO	-TO	-PD	Z	DC	C	0001 Ixxx
184h	FSR	Регістр адреси при непрямій адресації								xxxx xxxx
185h	-	Не реалізовано								-
186h	TRISB	Напрям виводів PORTB								1111 1111
187h	-	Не реалізовано								-
188h	-	Не реалізовано								-
189h	-	Не реалізовано								-
18Ah	PCLATH	-	-	-	Старші біти лічильника команд PC					------ 0 0000
18Bh	INTCON	GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF	0000 000x
18Ch	EECON1	EEPGD	-	-	-	WREE	WREN	WR	RD	x ------ x000
18Dh	EECON2	Регістр керування 2 (фізично не реалізований)								
18Eh	-	Резерв								-
18Fh	-	Резерв								-

Позначення: - = не використовується, читається як 0; x = не відомо.

Додаток Б– Список команд мікроконтролерів PIC16F87X

Мнемоніка команди	Опис	Циклів	14-разрядний код Біт 13…… …….Біт0	Зм. прапори	Прим.
1	2	3	4	5	6
Байт орієнтовані команди					
ADDWF f,d	**Складання W і f**	**1**	**00 0111 dfff ffff**	**C,DC,Z**	**1,2**
ANDWF f,d	**Побитне ' I ' W та f**	**1**	**00 0101 dfff ffff**	**z**	**1,2**
CLRF f	**Очистити f**	**1**	**00 0001 1fff ffff**	**z**	**2**
CLRW	**Очистити W**	**1**	**00 0001 0xxx xxxx**	**z**	
COMF f,d	**Інвертувати f**	**1**	**00 1001 dfff ffff**	**z**	**1,2**
DECF f,d	**Відняти 1 з f**	**1**	**00 0011 dfff ffff**	**z**	**1,2**
DECFSZ f,d	**Відняти 1 з f і пропустити якщо 0**	**1(2)**	**00 1011 dfff ffff**		**1,2,3**
INCF f,d	**Додати 1 до f**	**1**	**00 1010 dfff ffff**	**z**	**1,2**
INCFSZ f,d	**Додати 1 до f і пропустити якщо 0**	**1(2)**	**00 1111 dfff ffff**		**1,2,3**
IORWF f,d	**Побітне 'АБО' W та f**	**1**	**00 0100 dfff ffff**	**z**	**1,2**
MOVF f,d	**Переслати f**	**1**	**00 1000 dfff ffff**	**z**	**1,2**
MOVWF f	**Переслати W у f**	**1**	**00 0000 1fff ffff**	**z**	**1,2**
NOP	**Немає операції**	**1**	**00 0000 0xx0 0000**		
RLF f,d	**Циклічне зрушення f вліво через перенесення**	**1**	**00 1101 dfff ffff**	**c**	**1,2**
RRF f,d	**Циклічне зрушення f управо через перенесення**	**1**	**00 1100 dfff ffff**	**c**	**1,2**
SUBWF f,d	**Відняти W з f**	**1**	**00 0010 dfff ffff**	**C,DC,Z**	**1,2**
SWAPF f,d	**Поміняти місцями півбайти в регістрі f**	**1**	**00 1110 dfff ffff**		**1,2**
XORWF f,d	**Побітне "виключаюче АБО" W та f**	**1**	**00 0110 dfff ffff**	**z**	**1,2**
Біт орієнтовані команди					
BCF f,b	**Очистити біт b в регістрі f**	**1**	**01 00bb bfff ffff**		**1,2**
BSF f,b	**Встановити біт b в регістрі f**	**1**	**01 01bb bfff ffff**		**1,2**
BTFSC f,b	**Перевірити біт b в**	**1(2)**	**01 10bb**		**3**

	регістрі f, пропустити якщо 0		**bfff ffff**		

Продовження додатка Б

1	2	3	4	5	6
BTFSS f,b	**Перевірити біт b в регістрі f, пропустити якщо 1**	**1(2)**	**01 11bb bfff ffff**		**3**
Команди управління і операцій з константами					
ADDLW к	**Скласти константу з W**	**1**	**11 111x kkkk kkkk**	**C,DC,Z**	
ANDLW k	**Побітне ' I ' константи та W**	**1**	**11 1001 kkkk kkkk**	**Z**	
CALL k	**Виклик підпрограми**	**2**	**10 0kkk kkkk kkkk**		
CLRWDT	**Очистити WDT**	**1**	**00 0000 0110 0100**	**-TO.- PD**	
GOTO k	**Безумовний перехід**	**2**	**10 1kkk kkkk kkkk**		
IORLW k	**Побітне 'АБО' константи та W**	**1**	**11 1000 kkkk kkkk**	**Z**	
MOVLW k	**Переслати константу в W**	**1**	**11 00xx kkkk kkkk**		
RETFIE	**Повернення з підпрограми з дозволом переривань**	**2**	**00 0000 0000 1001**		
RETLW k	**Повернення з підпрограми із завантаженням константи в W**	**2**	**11 01xx kkkk kkkk**		
RETURN	**Повернення з підпрограми**	**2**	**00 0000 0000 1000**		
SLEEP	**Перейти в режим SLEEP**	**1**	**00 0000 0110 0011**	**-TO.- PD**	
SUBLW k	**Відняти W з константи**	**1**	**11 110x kkkk kkkk**	**C,DC,Z**	
XORLW k	**Побітне "виключаюче АБО" константи та W**	**1**	**11 1010 kkkk kkkk**	**Z**	

Примітки:

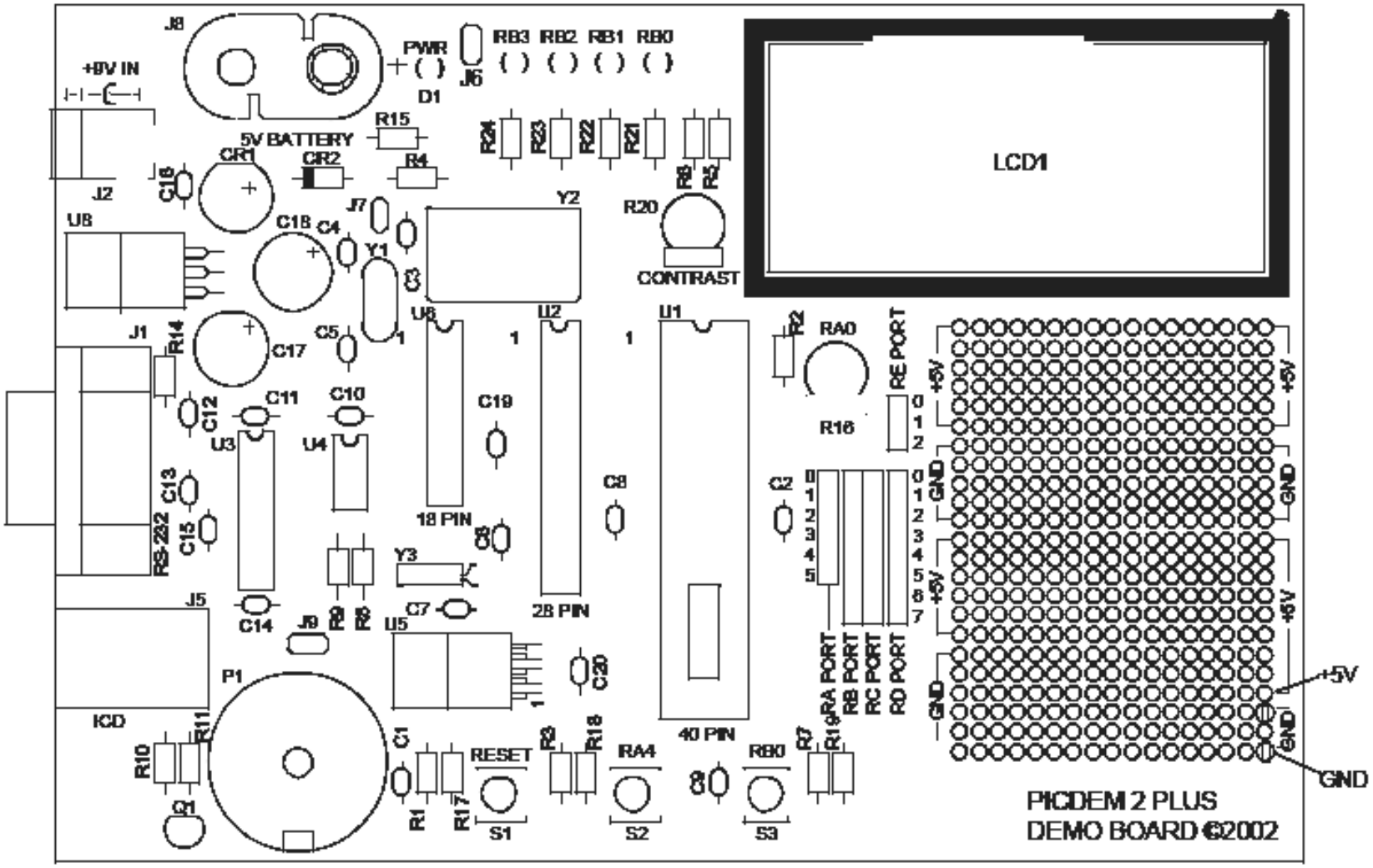
1. При виконанні операції "читання - модифікація - запис" з портом введення/виводу початкові значення прочитуються з виводів порту, а не з вихідних защіпок. Наприклад, якщо у вихідній защіпці було записана '1', а на відповідному виході низький рівень сигналу, то назад буде записано значення '0'.
2. При виконанні запису в TMR0 (і d=1) переддільник TMR0 скидається, якщо він підключений до модуля TMR0.

3. Якщо умова істинна або змінюється значення лічильника команд PC, то інструкція виконується за два цикли. У другому циклі виконується команда NOP.

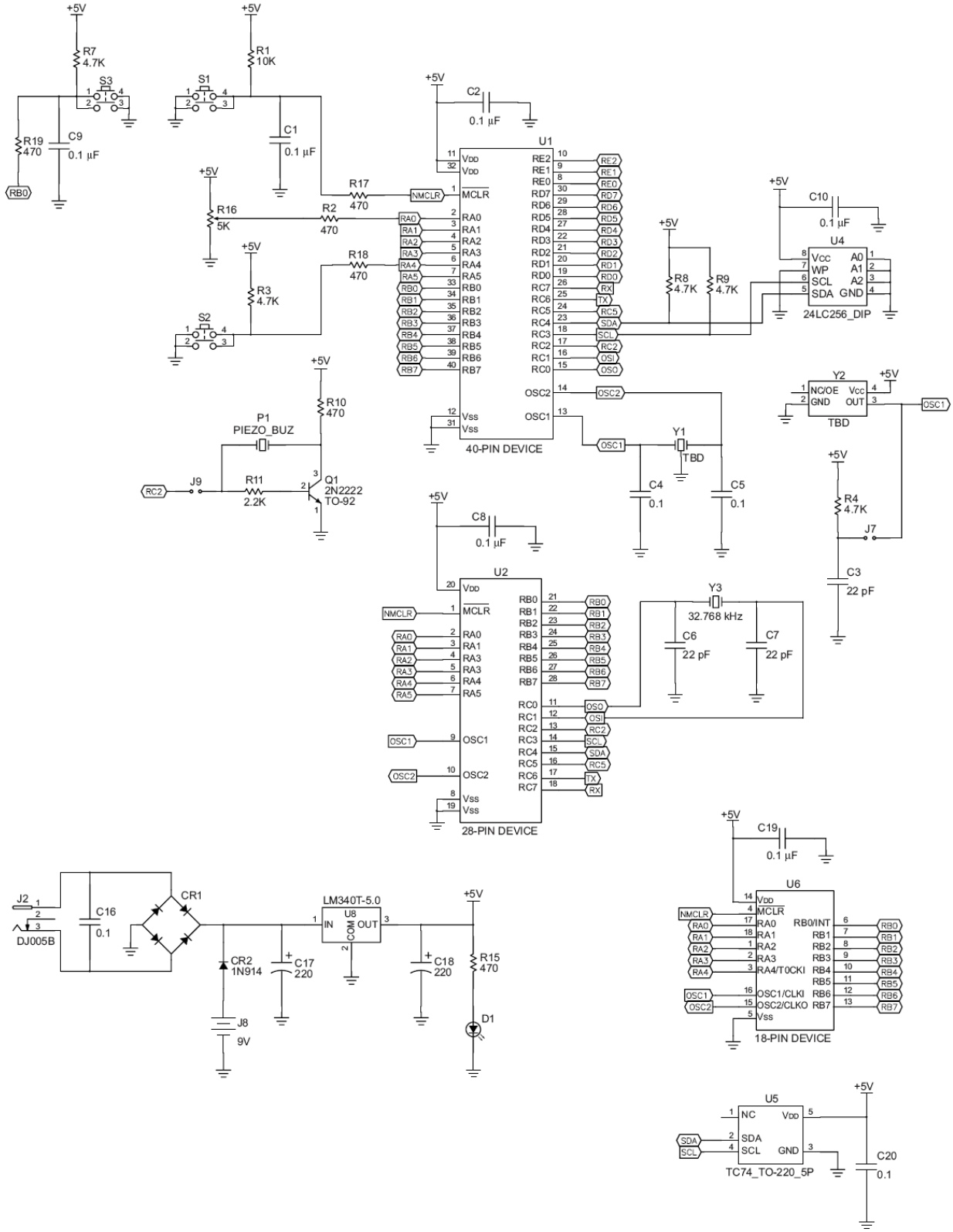
4. Опис поля операнда приведено в таблиці:

Поле	Опис
f	Адреса регістра (від 0x00 до 0x7F)
w	Робочий регістр (акумулятор)
b	Номер біта у 8-разрядному регістрі
k	Константа (дані або мітка)
X	Не має значення (0 або1). Асемблер генерує x=0
d	Показник адреси результата операції: d = 0 - результат зберігається у регістрі w;
-TO	Флаг переповнювання WDT
-PD	Флаг скида по включенню живлення

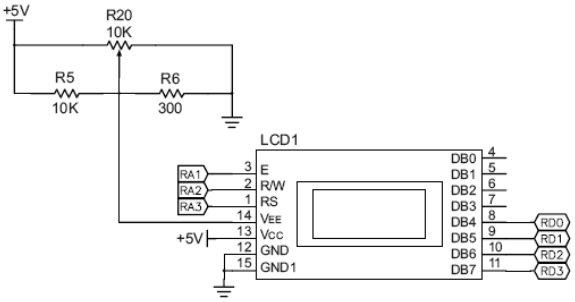
Додаток В–Розміщення елементів на платі PICDEM 2 PLUS

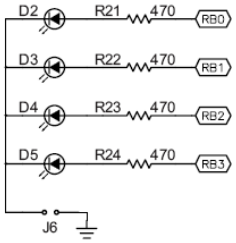


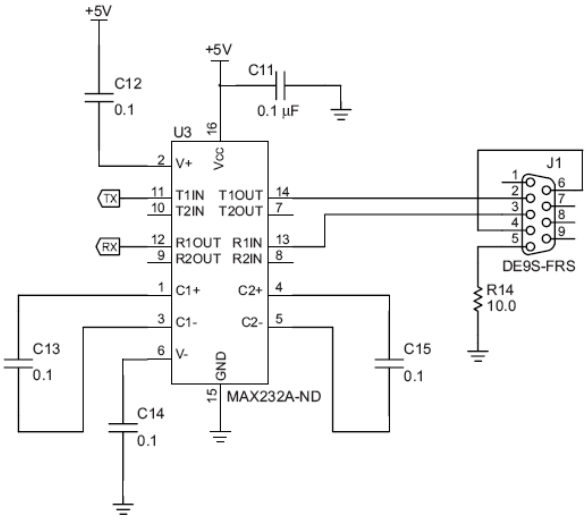
Додаток Г–Електрична схема плати PICDEM 2 PLUS

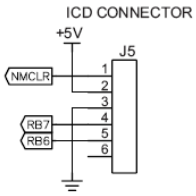


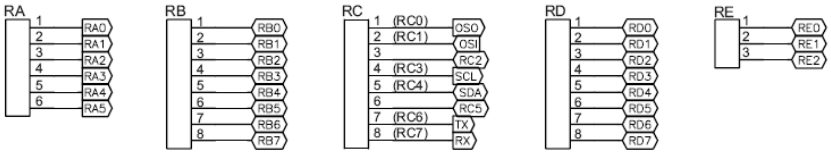
Продовження додатка Г











Навчальне видання

Методичні вказівки
до лабораторного практикуму
з дисципліни “Мікропроцесорні пристрої”
(для спеціальності 7.092203 “Електромеханічні системи автоматизації та електропривод”)
Частина 2

Укладач Олександр Михайлович Наливайко

Редактор

Підписано до друку _____ Формат 60х84/16.
Папір офсетний Ум.др. ар._____ Обл.-вид.арк.
Тираж Прим. Зам. №

Видавець і виготівник
“Донбаська державна машинобудівна академія”
84313, м.Краматорськ, вул. Шкадінова, 72
Свідоцтво про внесення до Державного реєстру
Серія ДК №1633 від 24.12.03