Міністерство освіти і науки України

Донбаська державна машинобудівна академія

МІКРОПРОЦЕСОРНІ ПРИСТРОЇ

Методичні вказівки

до самостійної роботи

для студентів спеціальності 7.092203

«Електромеханічні системи автоматизації»

усіх форм навчання

Перезатверджено
на засіданні кафедри ЕСА
Протокол №1 від 21.08.12

Краматорськ 2012

УДК

Методичні вказівки до самостійної роботи з дисципліни «Мікропроцесорні пристрої» для студентів спеціальності 7.092203 «Електромеханічні системи автоматизації» усіх форм навчання / укл.: А. М. Наливайко, Д. С. Пономарьов. – Краматорськ: ДДМА, 2008. – 132 с.

У методичних вказівках викладений короткий опис мікроконтролерів I8051, PIC16F877, основні методи й прийоми програмування на мові асемблеру сімейств MCS-51 та Picmicro. Наведені приклади складання програм і завдання для самостійної роботи студентів.

Укладачі: О. М. Наливайко, доц.,
Д. С. Пономарьов, асис.

Відп. за випуск: О. М. Наливайко, доц.

ЗМІСТ

Вступ

Застосування мікроконтролерів, що починалося в минулому сторіччі із цифрових годинників і мікрокалькуляторів, у цей час продовжує розширюватися й виправдане не тільки малими розмірами, вагою й енергоспоживанням при високій надійності й низької вартості цих мікросхем. Головною причиною ефективності використання мікроконтролерів є їхня функціональна універсальність, яка обумовлена можливістю їх програмування.

Програмування нерозривно пов'язане з комп'ютерами, хоча за останні десятиліття цей зв'язок став більш опосередкованим. Прагнення до переносності й зручності користувацького інтерфейсу підносить прикладні програми до усіх більш високі шари математичного забезпечення, відокремлювані від апаратно-залежних програм зростаючою кількістю оболонок. Таке ускладнення математичного забезпечення у свою чергу вимагає все більших апаратних ресурсів. Прогрес технічних характеристик мікропроцесорів і пов'язаних з ними наборів мікросхем викликає з однієї сторони замилування, а з іншого – тривогу. Моральне старіння персональних комп'ютерів значно випереджає їхнє фізичне зношування. Добре ще, що завдяки мікромініатюризації цей прогрес не вимагає надмірних витрат матеріальних ресурсів, але фінансові кошти, що утягуються, величезні. Експонентний ріст не може тривати безмежно. Можна тільки ворожити, які будуть результати зіткнення зростаючих апетитів споживачів з неминучими технологічними обмеженнями прогресу в розробці комп'ютерів.

У порівнянні з персональними комп'ютерами мікроконтролери не так відомі, тому що скромно ховаються усередині найрізноманітніших пристроїв, у тому числі й у комп'ютерах, у яких без мікроконтролерів не могло б працювати жодне з периферійних пристроїв, таких як клавіатура, миша, дисковід, принтер, сканер і т.д. Та й програмування для мікроконтролерів не є престижним у порівнянні із програмуванням для персональних комп'ютерів. Але ця робота не менш важлива й цікава, хоча

й не настільки помітна широкій публіці. Особливості програмування для мікроконтролерів пов'язані з обмеженнями обчислювальних ресурсів.

1 ОПИС МІКРОКОНТРОЛЕРА I8051

1.1 Організація пам'яті мікроконтролера

Мікроконтролери сімейства Intel MCS-51 виконані за гарвардською архітектурою й мають роздільну пам'ять команд і даних. Така організація пам'яті визначає деякі особливості при програмуванні мікроконтролерів даного типу. Крім цього до мікроконтролера можна підключити зовнішню пам'ять команд і даних. Внутрішня пам'ять команд і даних організована безпосередньо в самому мікроконтролері на одному кристалі.

На рисунку 1 зображена карта пам'яті команд мікроконтролера. Обсяг внутрішньої (резидентної) пам'яті програм (ROM, EPROM або OTP ROM), розташованій на кристалі, залежно від типу мікросхеми може становити 0 (Romless), 4К (базовий кристал), 8К, 16К або 32К. При необхідності користувач може розширювати пам'ять програм установкою зовнішнього ПЗУ. Дозвіл роботи із внутрішнім або зовнішнім ПЗУ визначається значенням логічного сигналу на виводі EA (External Access):

EA=1 – доступ до внутрішнього ПЗУ;

EA=0 – доступ до зовнішнього ПЗУ.

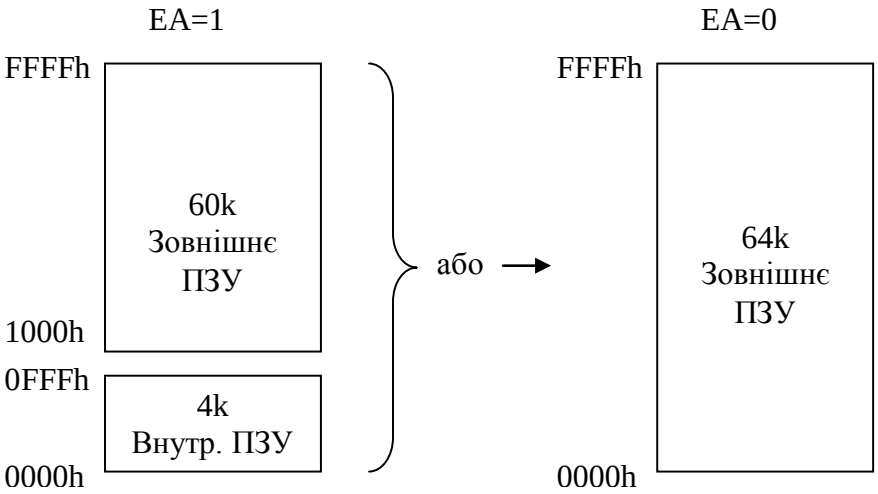


Рисунок 1 – Організація пам'яті команд

Область нижніх адрес пам'яті програм використовується системою переривань. Архітектура мікросхеми I8051забезпечує підтримку п'яти джерел переривань:

– двох зовнішніх переривань;

– двох переривань від таймерів;

– переривання від послідовного порту.

Розподіл нижніх адрес пам'яті команд зазначено на рисунку 2. Дані адреси називають «Векторами переривань». При подачі сигналу скидання RESET мікроконтролер виконує початкову ініціалізацію й починає витягувати команди за адресою 0000h. Тому, якщо передбачається застосовувати систему переривань, за даніею адресою слід записувати команду безумовного переходу на адресу 002Bh або вище. Це необхідно для обходу основною програмою таблиці векторів переривань.

З появою відповідної події, наприклад переповнення таймера 0, і умові дозволу переривання для даної події, основна програма переривається й відбувається добування команд по відповідному до адреси (для таймера 0 – 000Bh). Більш докладно із системою переривань можна познайомитися в підрозділі 1.3 «Організація системи переривань».

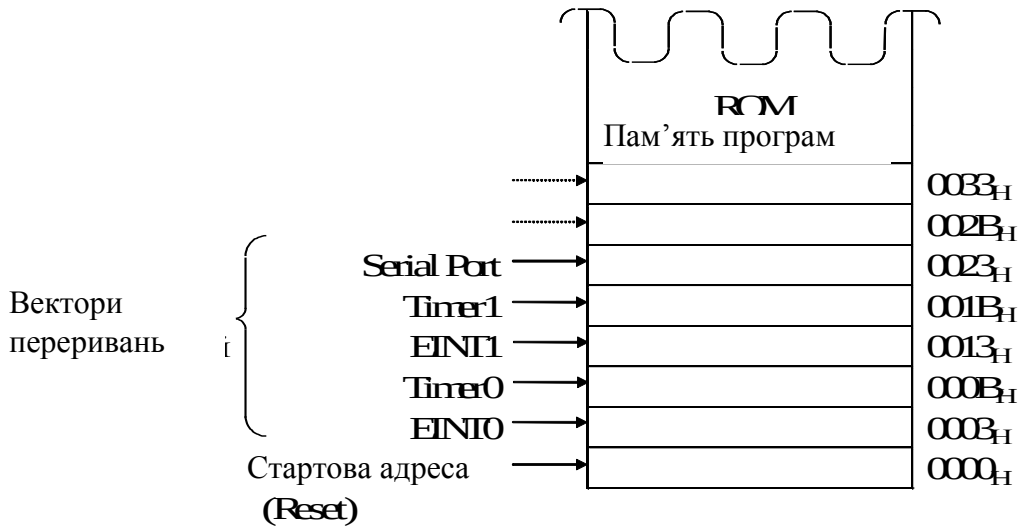


Рисунок 2 – Нижня область пам'яті команд

Пам'ять даних відділена від пам'яті програм. У цій області можлива адресація 64 К зовнішнього ОЗУ. При звертанні до зовнішньої пам'яті даних ЦП мікроконтролера генерує відповідні сигнали читання $\overline{RD}$ або запису $\overline{WR}$. Взаємодія із внутрішньою пам'яттю даних здійснюється на командному рівні, при цьому сигнали $\overline{RD}$ й $\overline{WR}$ не виробляються.

Зовнішня пам'ять програм і зовнішня пам'ять даних можуть комбінуватися шляхом сполучення сигналів $\overline{RD}$ і $\overline{PSEN}$ за схемою «логічного І» для одержання стробу доступу до зовнішньої пам'яті (програм/даних).

Внутрішня пам'ять даних розподілена на 2 ділянки: нижня область пам'яті – lower memory (адреси 00h–7Fh) містить РЗП, які вибираються з 4 банків і область пам'яті користувача (рис. 5); верхня область пам'яті – upper memory (адреси 80h–Ffh) містить регістри спеціальних функцій (SFR). Цю область пам'яті не можна використовувати для зберігання інформації користувача, і вона використовується тільки для доступу до регістрів SFR (рис. 4). Організація пам'яті даних зазначена на рисунку 3.

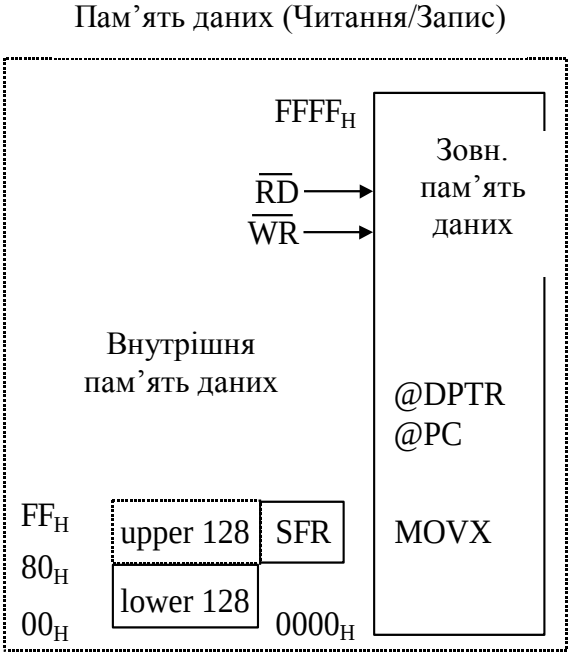


Рисунок 3 – Організація пам'яті даних

Перші 32 байти пам'яті lower memory складаються з 4 банків (Register Bank) по 8 регістрів R7...R0. Регістри R0 і R1 у кожному з банків можуть використовуватися в якості регістрів непрямої адреси.

Наступні за регістровими банками 16 байт утворюють блок побітно-адресованого простору. Набір інструкцій MCS-51 містить широкий вибір операцій над бітами, а 128 біт у цьому блоці адресуються прямо, й адреси мають значення від 00H до 7FH.

Усі байти в нижній 128-байтовій половині пам'яті можуть адресуватися як прямо, так і непрямо.

Верхня 128-байтова половина пам'яті ОЗУ (Upper 128) у мікросхемі i8051 відсутня, але є у версіях кристалів з 256 байтами ОЗУ. У цьому випадку область «Upper 128» доступна тільки при непрямій адресації. Область SFR (Special Function Register) доступна тільки при прямій адресації.

Розміщення регістрів спеціальних функцій у просторі SFR показане на рисунку 4. Вони містять у собі регістри портів, таймери, засоби керування периферією й т. п.

Бітова адресація	8 байт								
$F8_H$									FFH
$F0_H$	B								F7H
$E8_H$									EFH
$E0_H$	ACC								E7H
$D8_H$									DFH
$D0_H$	PSW								D7H
$C8_H$									CFH
$C0_H$									C7H
$B8_H$	IP								BFH
$B0_H$	P3								B7H
$A8_H$	IE								AFH
$A0_H$	P2								A7H
98_H	SCON	SBUF							9FH
90_H	P1								97H

Рисунок 4 – Розміщення регістрів спеціальних функцій у просторі SFR

Для 16 адрес у просторі SFR є можливість як байтової, так і бітової адресації. Для бітно-адресованих регістрів шістнадцатирічна адреса закінчується на «0H» або на «8H». Бітові адреси в цій області мають значення від 80H до FFH.

<table>
<tr><td>7FH</td><td colspan="8">Побайтно-адресована область ОЗП</td></tr>
<tr><td>30_H</td><td colspan="8">(direct, indirect)</td></tr>
<tr><td>2FH</td><td>7FH</td><td>7EH</td><td>7DH</td><td>7CH</td><td>7BH</td><td>7AH</td><td>79_H</td><td>78_H</td></tr>
<tr><td>2EH</td><td>77_H</td><td>76_H</td><td>75_H</td><td>74_H</td><td>73_H</td><td>72_H</td><td>71_H</td><td>70_H</td></tr>
<tr><td></td><td></td><td colspan="6">Побітно-адресована область ОЗП
(direct)</td><td></td></tr>
<tr><td>21_H</td><td>0FH</td><td>0EH</td><td>0DH</td><td>0CH</td><td>0BH</td><td>0AH</td><td>09_H</td><td>08_H</td></tr>
<tr><td>20_H</td><td>07_H</td><td>06_H</td><td>05_H</td><td>04_H</td><td>03_H</td><td>02_H</td><td>01_H</td><td>00_H</td></tr>
<tr><td>1FH
18_H</td><td colspan="8">Банк РЗП 3</td></tr>
<tr><td>17_H
10_H</td><td colspan="8">Банк РЗП 2</td></tr>
<tr><td>0FH
08_H</td><td colspan="8">Банк РЗП 1</td></tr>
<tr><td>07_H
00_H</td><td colspan="8">⇐ SP після RESET
Банк РЗП 0(R7+R0)</td></tr>
</table>

Рисунок 5 – Нижня область внутрішньої пам'яті даних

Регістри SFR мають наступне призначення:

Акумулятор A (Accumulator, адреса $E0_H$)

Команди архітектури MCS-51 використовують акумулятор як джерело і як приймач при обчисленнях і пересиланнях. Крім звертання до акумулятора командами, що використовують мнемоніку „A”, є можливість побітової або побайтової адресації, як SFR-регістр.

Регістр B (Multiplication Register, адреса $F0_H$)

Регістр B використовується як джерело і як приймач при операціях

множення й ділення, звертання до нього, як до регістру SFR, проводиться аналогічно акумулятору.

Слово стану програми PSW (Program Status Word, адреса $D0_H$)

Даний регістр містить біти, що відбивають результати виконання операцій, біти вибору регістрового банку й біт загального призначення, доступний користувачеві. PSW відображений на область SFR (рис.6).

Біти:	7	6	5	4	3	2	1	0
Позначення:	CY	AC	F0	RS1	RS0	OV	–	P

Рисунок 6 – Розташування бітів регістру PSW

Біти регістру PSW мають наступне призначення:

CY (Carry) – ознака переносу. Установлюється, якщо в старшому біті акумулятора при виконанні арифметичної операції в результаті виникає перенос біта або позика. При виконанні операцій ділення або множення біт скидається. Установка/скидання – апаратно й програмно;

AC (Auxiliary Carry) – ознака додаткового переносу. Використовується при виконанні інструкцій додавання або вирахування для вказівки переносу або позики з біта 3 при утворюванні молодшого напівбайту результату (D0-D3). Установка/скидання – апаратно й програмно;

F0 – прапор стану, обумовлений користувачем. Установка/скидання тільки програмно;

RS0, RS1 – прапори-покажчики банку робочих регістрів (РЗП). Установка/скидання тільки програмно;

OV (Overflow) – прапор переповнення. Установлюється, якщо результат операції додавання/вирахування не укладається в 7 бітах і старший восьмий біт не може бути інтерпретований як знаковий. При виконанні операції ділення прапор скидається, а у випадку ділення на 0 установлюється. При множенні прапор установлюється, якщо результат більше 255. Можливе програмне скидання/установка;

P (Parity) – прапор парності одиничних бітів у акумуляторі. Є доповненням умісту акумулятора до парності. У 9-розрядному слові, що

полягає з 8 розрядів акумулятора й біта P завжди втримується парне число одиничних бітів. Якщо всі біти акумулятора рівні 0, то й біт P скинутий. Він програмно доступний тільки для читання.

Розряд 1 PSW зарезервований і може використовуватися для програмного запису/читання.

Регістри портів P 0-P3 (адреси 80_H, 90_H, $A0_H$, $B0_H$)

Кожний порт є фіксатором-засувкою й може адресуватися як побайтно, так і побітно. Крім роботи як звичайних портів уведення/виводу, лінії портів можуть виконувати ряд альтернативних функцій:

– Через порт 0 (у мультиплексному режимі) виводиться молодший байт адреси, а також видається чи приймається в мікроконтролер байт даних при роботі із зовнішньою пам'яттю програм/даних;

– Через порт 2 виводиться старший байт адреси зовнішньої пам'яті програм і даних;

– Порт 3 має наступні альтернативні функції:

P3.7 – строб читання із зовнішньої пам'яті даних (Read Data from External Memory, $\overline{RD}$);

P3.6 – строб запису у зовнішню пам'ять даних (Write Data to External Memory, $\overline{WR}$);

P3.5 – зовнішній вхід T/C1 (Timer/Counter 1 External Input, T1);

P3.4 – зовнішній вхід T/C0 (Timer/Counter 0 External Input, T0);

P3.3 – вхід зовнішнього переривання 1 (External Interrupt 1 Input Pin, $\overline{INT1}$);

P3.2 – вхід зовнішнього переривання 0 (External Interrupt 0 Input Pin, $\overline{INT0}$);

P3.1 – вихід даних передавача послідовного порту (Serial Port Transmit Pin, Txd);

P3.0 – вхід даних приймача послідовного порту (Serial Port Receive Pin, Rxd).

Покажчик стеку SP (Stack Pointer, адреса 81_H)

Використовується для вказівки на вершину стеку в операціях запису в стек і читання з нього. Неявно використовується такими командами, як PUSH, RET, RETI, POP. За апаратним скиданням від ЦП установлюється в значення 07_H (область стеку в цьому випадку починається з адреси

внутрішньої пам'яті даних 08_H) й інкрементується при кожному записі в стек. Запис в SFR-регістр SP (з використанням байтової адресації) проводиться для вказівки положення стеку у внутрішній пам'яті даних.

Покажчик даних DPTR (Data Pointer, адреси 82_H, 83_H)

Команди архітектури MCS-51 використовують DPTR для пересилання даних, пересилання коду й для переходів (JMP@A+DPTR). DPTR складається із двох регістрів: молодшого – DPL і старшого – DPH, звертання до них – тільки байтове.

Регістр керування енергоспоживанням PCON (Power Control Register, адреса 87_H)

Для кристалів, виконаних за HMOS-технологією, даний регістр має тільки один значущий біт – SMOD, який управляє швидкістю роботи послідовного порту.

Регістри таймерів/лічильників TL0, TL1, TH0, TH1 (адреси 8AH, $8B_H$, $8C_H$, 8DH)

Утворюють 16-бітові (Low/High) регістри таймерів/лічильників T/C0 і T/C1. Звертання до регістрів тільки байтовое. Докладно описані в підрозділі «Організація таймерів/лічильників».

Регістр режиму таймерів/лічильників TMOD (Timer/Counter Mode Control Register, адреса 89_H)

Регістр керування таймерів/лічильників TCON (Timer/Counter Control Register, адреса 88_H)

Призначені для керування роботою таймерної секції мікроконтролера. Докладно описані в підрозділі «Організація таймерів/лічильників».

Буфер послідовного порту SBUF (Serial Data Buffer, адреса 99_H)

Являє собою два окремі регістри. При записі в SBUF завантажується «буфер передачі» послідовного порту, при читанні SBUF зчитується вміст «буфера приймання» послідовного порту.

Регістр керування послідовним портом SCON (Serial Port Control Register, адреса 98_H)

Призначений для керування роботою послідовного порту. Звертання до даного регістру може бути як байтовим, так і побітним.

Регістр дозволу переривань IE (Interrupt Enable Register, адреса $A8_H$)

Регістр керування пріоритетом переривання IP(Interrupt Priority Control Register, адреса $B8_H$)

Підтримують роботу системи переривань мікроконтролера. Докладний опис роботи з регістрами даний в підрозділі «Організація системи переривань».

1.2 Організація таймерів/лічильників

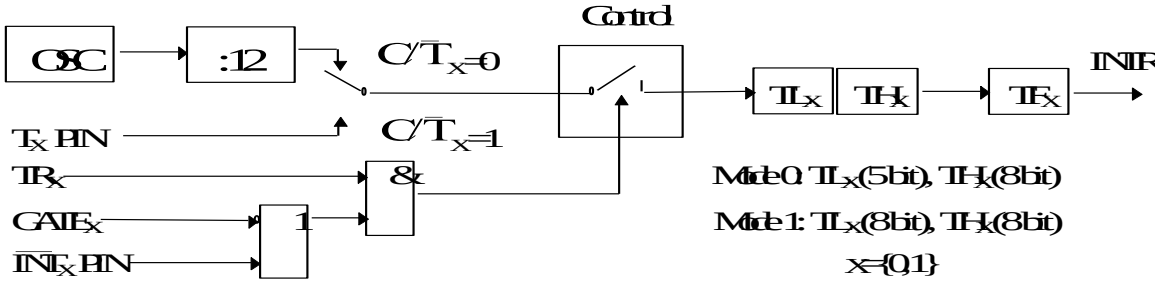
Таймери/лічильники (T/C0 і T/C1) призначені для підрахунку зовнішніх подій (виводи T0 і T1), організації програмно-керованих тимчасових затримок і виміру тимчасових інтервалів. Таймер 1 може також служити генератором швидкості передачі для послідовного порту.

Таймер/лічильник, працюючи в режимі таймера, веде підрахунок тактів діленої системної частоти (запрограмований проміжок часу) і видає запит переривання. Регістр таймера інкрементується один раз у кожному периферійному циклі. Оскільки цикл складається з 12 тактів, то швидкість рахунку таймера дорівнює $F_{OSC}/12$.

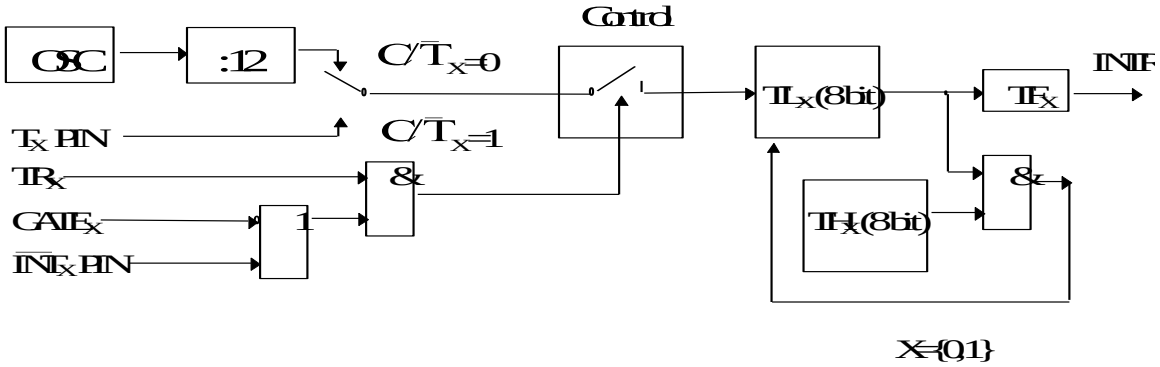
У режимі лічильника регістр таймера веде підрахунок (передустановленої кількості подій) негативних перепадів сигналу на зовнішньому вході й по закінченню рахунку видає запит переривання. Оскільки розпізнавання негативного переходу зовнішнього сигналу займає 24 періоду тактової частоти (2 циклу), т. ч. максимальна швидкість рахунку дорівнює $F_{OSC}/24$. Обмежень на робочий цикл не накладається, але щоб гарантувати опитування конкретного рівня сигналу хоча б один раз до моменту його зміни, він повинен утримуватися на вході хоча б протягом одного повного периферійного циклу.

Програмне керування функціонуванням T/C0 і T/C1 забезпечують SFR-Регістри TMOD і TCON. Можливі 4 режими роботи T/C мікроконтролера, які визначаються установкою відповідних бітів регістру TMOD. Режим 0 (13-бітовий таймер), 1 (16-бітовий таймер) і 2 (8-бітовий таймер з автозавантаженням) повністю ідентичні для обох T/C. У режимі

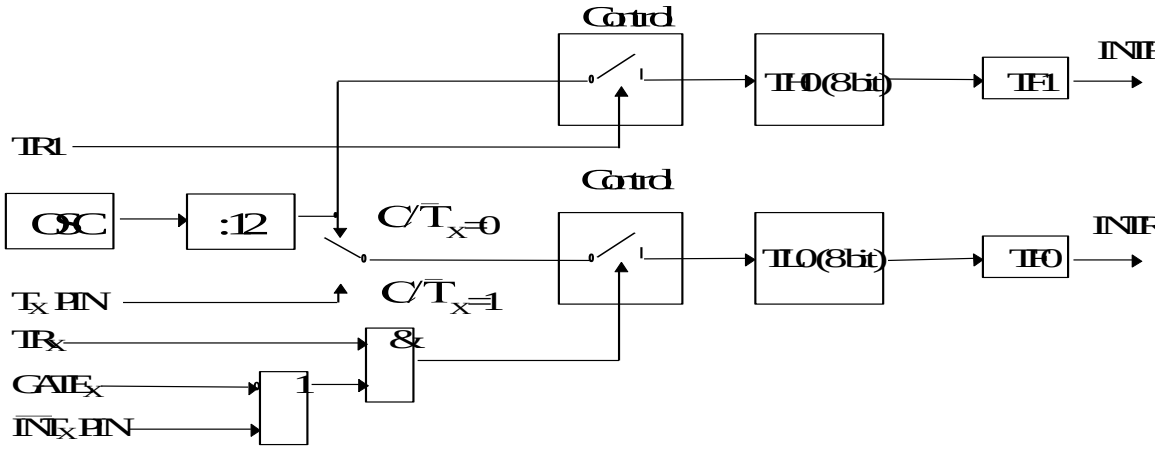
3 (два 8-бітових регістри) працює тільки T/C0, T/C1 у цьому режимі заблокований («позбавлений» біта керування запуском TR1 і прапора переповнення TF1) і зберігає вміст своїх регістрів TL1 і TH1. Логіка роботи T/C0 і T/C1 у режимах 0, 1, 2, 3 показана на рисунку 7.



А – Логіка роботи T/C0 та T/C1 у режимі 0, 1



Б – Логіка роботи T/C0 та T/C1 у режимі 2



В – Логіка роботи T/C0 та T/C1 у режимі 3

Рисунок 7 – Логіка роботи T/C0 таT/C1 у режимах 0, 1, 2 і 3

Шляхом відповідного програмування регістрів TMOD і TCON здійснюється вмикання й вимикання таймерів/лічильників, вибір джерела їх тактування й установка певного режиму їх роботи.

Функціональне призначення розрядів цих регістрів наступне (рис.8).

Біти:	7	6	5	4	3	2	1	0
Позначення:	GATE1	$C/\overline{T}1$	M1.1	M1.0	GATE0	$C/\overline{T}0$	M0.1	M0.0

Рисунок 8 – Розташування бітів регістру TMOD

Призначення бітів:

Gatex – Якщо Gatex=1 і Trx=1, то включення й вимикання відповідного таймера здійснюється зовнішнім сигналом на вході Intx. Коли Gatex=0, біт керування запуском Trx=1 дозволяє проходження вхідних сигналів від обраного джерела тактування;

C/$\overline{T}$x – вибирає функцію таймера, (підрахунок імпульсів діленої системної частоти) або вибирає функцію лічильника (підрахунок негативних переходів сигналу на зовнішньому виводі Tx);

Mx.1, Mx.0 – біти вибору режиму таймерів. Можуть мати наступні комбінації:

Mx.1	Mx.0	
0	0	Mode 0: 8-бітовий таймер/лічильник (THx) з 5-бітовим переддільником (TLx);
0	1	Mode 1: 16-бітовий таймер/лічильник;
1	0	Mode 2: 8-бітовий автоперезавантажувальний таймер/лічильник (TLx). Константа перезавантаження попередньо заноситься в THx.
1	1	Mode 3: TL0 – це 8-бітовий таймер/лічильник; TH0 – 8-бітовий таймер, що використовує біти TR1 і TF1.

Біти:	7	6	5	4	3	2	1	0
Позначення:	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Рисунок 9 – Розташування бітів регістру TCON

Призначення бітів:

TRx – біти запуску відповідного таймера/лічильника. Установлюються/скидаються тільки програмно;

TFx – біти переповнення відповідного лічильника (установлюються/скидаються програмно й апаратно при переході значення таймера з $FFFF_H$ в 0000_H), очищається апаратно, коли процесор переходить на підпрограму обробки переривання або програмно;

IEx – прапор виявлення зовнішнього переривання. Установлюється апаратно, коли виявлене зовнішнє переривання (по фронту або рівні сигналу) на виводі Intx; скидається апаратно під час обробки переривання тільки в тому випадку, коли переривання було викликано фронтом сигналу або програмно;

ITx – вибір типу сигналу для виявлення зовнішнього переривання. Визначає тип сприйманого сигналу на вході Intx; для вибору спрацьовування за фронтом сигналу (високий і низький) потрібно встановити цей біт, для спрацьовування за рівнем (активний низький рівень) потрібно скинути цей біт.

Приклад настроювання таймерів/лічильників у режим 2:

```
 mov  TMOD, #20H     ; T/C1 у режимі 2
 mov  TL1, #data8    ; константи перезавантаження
 mov  TH1, #data8    ; таймера/лічильника 1
SETB  TR1            ; запуск таймера/лічильника на рахунок
```

1.3 Організація системи переривань

Архітектура системи керування перериваннями для базової моделі I8051 показана на рисунку 10.

Кожне із зовнішніх переривань INT0, INT1 може бути активізоване за рівнем «0» або за фронтом (перехід з «1» в «0») сигналів на виводах ОМЕОМ P3.2, P3.3, що визначається станом бітів IT0 і IT1 регістру TCON. При вступі запиту зовнішнього переривання INTx установлюється прапор

IEx регістру TCON. Установка прапорів IEx у регістрі TCON викликає відповідне переривання. Очищення прапора IEx проводиться в такий спосіб. При перериванні за фронтом IEx скидається апаратно (автоматично внутрішніми засобами OMEOM) при звертанні до відповідної до підпрограми обробки переривання. При перериванні за рівнем прапор очищається при знятті запиту зовнішнього переривання, тобто в IEx відслідковується стан виводу INTx.

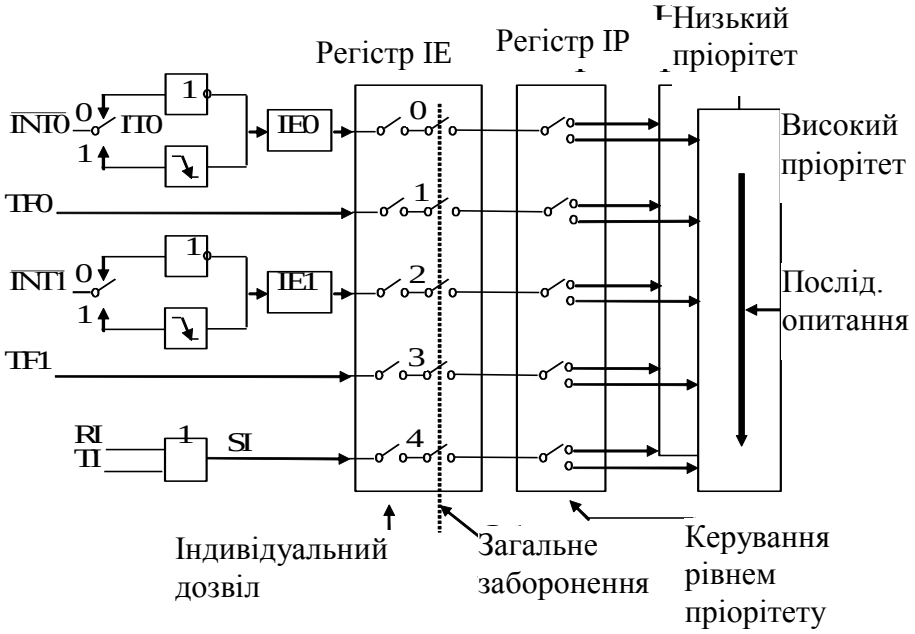


Рисунок 10 – Система переривань OMEOM

Переривання від таймерів/лічильників викликаються установкою прапорів TF0 і TF1 регістру TCON, які встановлюються при переповненні відповідних регістрів таймерів/лічильників (за винятком режиму 3, див. розділ «Організація таймерів/лічильників»). Відчищання прапорів TF0 і TF1 проводиться внутрішньою апаратурою OMEOM при перехід до підпрограми обслуговування переривання.

Переривання від послідовного порту викликається установкою прапора переривання приймача RI або прапора переривання передавача TI у регістрі SCON. На відміну від усіх інших прапорів, RI і TI скидаються тільки програмним способом звичайно в межах підпрограми обробки відповідного переривання.

Кожний з перелічених джерел переривань може бути індивідуально дозволений або заборонений установкою або скиданням відповідного біта в регістрі дозволу переривань IE. Регістр IE містить також біт EA, скидання якого в «0» забороняє відразу всі переривання. Необхідною умовою переривання є його дозвіл у регістрі IE. Формат і опис регістру дозволу переривань наведений нижче (рис. 11).

Біти:	7	6	5	4	3	2	1	0
Позначення:	EA	-	-	ES	ET1	EX1	ET0	EX0

Рисунок 11 – Розташування бітів регістру IE

Призначення бітів:

EA (Enable All) – дозвіл переривань від усіх джерел;

Біти 6, 5 – зарезервовані;

ES (Enable Serial) – дозвіл переривання від послідовного порту;

ETx (Enable T/Cx) – дозвіл переривання за переповненням від відповідного таймера;

EXx (Enable external) – дозвіл переривання за зовнішнім сигналом на вході $\overline{\text{INTx}}$;

Усі біти, які викликають переривання (IE0, IE1, TF0, TF1, RI, TI), можуть бути програмно встановлені або скинуті з тим же результатом, що й у випадку їх апаратної установки або скидання. Т. ч. переривання можуть програмно викликатися, або обслуговування переривання можуть програмно ліквідуватися. Крім того, переривання за входами $\overline{\text{INT0}}$, $\overline{\text{INT1}}$ можуть викликатися програмною установкою P3.2=0 і P3.3=0, як показано в наведеному нижче прикладі:

```
MAIN:     MOV  IE, #00000101B   ; Дозвіл переривання від INT0,INT1.
          MOV  IP, #04H         ; Присвоєння INT1 старшого пріоритету.
         SETB  EA               ; Загальний дозвіл переривань.
          M0V  P3, #11110011B   ; Імітація зовнішніх переривань.
SUBR:                           ; Перехід до підпрограми обслуговування
  ORG 013H                      ; INT1.
```

У запропонованому прикладі запити переривання INT0 і INT1, що мають різний пріоритет, надходять одночасно. При цьому обслуговується переривання з вищим пріоритетом.

У випадку, коли переривання за $\overline{INTx}$ (x=0,1) викликається рівнем сигналу на відповідному вході ОМЕОМ, прапор IEx при перехід до підпрограми обробки переривання автоматично скидається, а потім, якщо відповідний вивід ОМЕОМ P3.2 або P3.3 усе ще перебуває в стані логічного «0», знову встановлюється. Тому, у випадку, коли переривання за входами $\overline{INT0}$, $\overline{INT1}$ викликається рівнем, програмна установка в «1» прапорів IE0, IE1 викличе переривання, після чого відповідний прапор IEx (x=0,1) буде автоматично скинутий при перехід до підпрограми обробки переривання.

Прапори IE0, IE1, TF0, TF1, RI, TI установлюються незалежно від того дозволене чи ні відповідне переривання в регістрі IE.

Структура пріоритетів переривань є двоступінчастою. Кожному джерелу переривання може бути індивідуально привласнено один із двох рівнів пріоритету: високий або низький. Виконується це установкою (високий рівень пріоритету) або скиданням (низький рівень пріоритету) відповідного біта в регістрі пріоритетів переривань IP (рис. 12).

Біти:	7	6	5	4	3	2	1	0
Позначення:	-	-	-	PS	PT1	PX1	PT0	PX0

Рисунок 12 – Розташування бітів регістру IP

Призначення бітів:

Біти 7, 6, 5 – зарезервовані;

PS (Priority of Serial) – Установка рівня пріоритету від послідовного порту;

PTx (Priority of T/Cx) – Установка рівня пріоритету від відповідного T/C;

PXx (Priority of External) – Установка рівня пріоритету від відповідного зовнішнього джерела переривань.

Низькопріоритетне переривання може перериватися високопріоритетним, але ніколи не переривається запитом того ж рівня пріоритету. Тому, якщо одночасно виникають два переривання з різним рівнем пріоритету, то спочатку виконується високопріоритетне. Якщо ж

подібна ситуація складається для переривань із однаковим рівнем пріоритету, то послідовність їх обробки визначається спеціальною послідовністю опитування прапорів переривань (Interrupt Polling Sequence): IE0→TF0→IE1→TF1→RI+TI. Запит IE0 – розпізнається першим, TI+RI – останнім.

При виконанні переривання основної програми для виконання підпрограми обробки переривання усередині мікропроцесора виконується апаратно-реалізована команда LCALL, що виконує перехід за фіксованою адресою, значення якої залежить від джерела переривання. Дану адресу називають «вектором переривання». Адреси векторів переривань для базової моделі I8051 зазначені на рисунку 2. При цьому підпрограма буде виконуватися до команди повернення з переривання RETI. Дана команда знімає заборону на обробку інших переривань і завантажує в лічильник команд зі стека адресу наступної команди перерваної програми.

Програмістові при використанні системи переривань необхідно стежити за тим, щоб підпрограма обробки переривань не змінювала значення регістрів і областей пам'яті, використовуваних для обчислень і зберігання даних. Звичайно це виконується збереженням робочих регістрів у стеці при початку виконання обробки переривання й відмовою використання робочих областей пам'яті в підпрограмі. Приклад програми збереження стану процесора при обробці переривань:

```
LOC     EQU     $               ; Запам'ятовування лічильника адреси.
        ORG     0003h           ; Початкова адреса програми переривання.
          LJMP  SERV            ; Перехід на оброблювач переривання.
;          ...  ...             ; Виконання підпрограми.
        ORG     LOC             ; Відновлення лічильника адреси.
SERV:     PUSH  PSW             ; Збереження регістрів у стек.
          PUSH  ACC
          PUSH  B
          PUSH  DPL
          PUSH  DPH
           MOV  PSW, #1000b     ; Вибір іншого регістрового банку.
;         ...   ...             ; Тіло оброблювача переривання.
           POP  DPH             ; Відновлення регістрів зі стека
           POP  DPL             ; у зворотному порядку.
           POP  B
           POP  ACC
           POP  PSW
          RETI                  ; Повернення з переривання.
```

2 ОСНОВИ ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРНИХ СИСТЕМ

2.1 Загальні відомості

Діапазон мов написання вихідного тексту прикладної програми простягається від машинного коду до майже природньої мови. У машинному коді або мовою асемблера програмувати важче, чим алгоритмічною мовою високого рівня. Але, незважаючи на даний недолік, програма, написана цією мовою, виконується найбільше швидко й займає найменший обсяг пам'яті, що є досить важливим аргументом при виборі мови програмування для деяких видів завдань.

Об'єктні коди, отримані шляхом трансляції вихідних програм, написаних алгоритмічною мовою високого рівня, займають у пам'яті більше місця й вимагають більшого часу на виконання. Величезна більшість прикладних завдань керування об'єктами внаслідок того, що вони повинні вирішуватися в реальному часі, висувають настільки високі вимоги щодо швидкодії, що для їхнього рішення основним язиковим засобом написання прикладних програм ще довгі роки буде залишатися мова асемблера. Це положення про переважне використання мови асемблера підкріплюється й тією обставиною, що однокристальні мікроконтролери мають обмежений обсяг резидентної пам'яті програм і, отже, критичні до довжини прикладних програм. Вибір язикових засобів складання вихідних програм у кожному конкретному випадку залежить від характеристик прикладного завдання, досвіду програміста й припустимих витрат на розробку.

Якщо завдання на розробку прикладної програми для мікроконтролера поставлена, то для одержання тексту вихідної програми необхідно виконати ряд послідовних дій:

– Докладно описати завдання.

– Проаналізувати завдання.

– Виконати інженерну інтерпретацію завдання, бажано із залученням того або іншого апарата формалізації (граф автомата, мережі Петрі, матриці станів і зв’язності й т. п.).

– Розробити загальну схему алгоритму роботи контролера.

– Розробити деталізовані схеми окремих процедур, виділених на основі модульного принципу складання програм.

– Детально проробити інтерфейс контролера й внести виправлення в загальну й деталізовані схеми алгоритмів.

– Розподілити робочі регістри й пам'ять.

– Сформувати текст вихідної програми.

У результаті роботи з трьома першими пунктами даного переліку одержують так звану функціональну специфікацію прикладної програми, у якій основна увага приділяється деталізації способів формування вхідної й вихідної інформації.

Мовою схем алгоритмів розроблювач описує метод, обраний ним для рішення поставленого завдання. Досить часто буває, що те саме завдання може бути вирішене різними методами. Спосіб рішення завдання, обраний на етапі його інженерної інтерпретації, на основі якого формується схема, визначає не тільки якість розроблювальної прикладної програми, але і якісні показники кінцевого виробу.

Алгоритм є точна процедура, що пропонує контролеру однозначно певні дії щодо перетворення вхідних даних на оброблені вихідні дані. Тому розробка схеми вимагає граничної точності й однозначності використовуваної атрибутики: символічних імен змінних, констант, підпрограм (модулів), символічних адрес таблиць, портів уведення виводу й т. п. Основну увагу при розробці слід приділити тому розділу функціональної специфікації прикладної програми, у якому наводиться опис апаратури сполучення з об'єктом керування. Цей опис повинний бути деталізований аж до електричних і тимчасових характеристик кожного вхідного й вихідного сигналу або пристрою. Даним описом є принципова електрична схема системи, у якій використовується мікроконтролер.

Успіх розробки прикладної програми полягає у використанні методу декомпозиції, при якому усе завдання послідовно розділяється на менші функціональні модулі. Кожний з модулів можна аналізувати, розробляти й налагоджувати окремо від інших. При виконанні прикладної програми в мікроконтролері керування без усяких двозначностей передається від одного функціонального модуля до іншого. Схема зв’язності цих

функціональних модулів, кожний з яких реалізує деяку процедуру, утворює загальну схему алгоритму прикладної програми. Мову графічних образів схеми алгоритму можна використовувати на будь-якому рівні деталізації опису модулів аж до того, що кожному операторові схеми буде відповідати єдина команда мікроконтролера.

Структурне програмування є процес побудови прикладної програми з набору програмних модулів, кожний з яких реалізує певну процедуру обробки даних. Програмні модулі повинні мати тільки одне місце входу й одне місце виходу. Тільки в цьому випадку окремі модулі можна розробляти й налагоджувати незалежно, а потім поєднувати в закінчену прикладну програму з мінімальними проблемами їх взаємозв'язку.

Джерелом переважної більшості помилок програмування є використання модулів, що мають один вхід і кілька виходів. При необхідності організації множинних розгалужень у програмі декомпозицію завдання виконують таким чином, щоб кожний функціональний модуль мав тільки один вхід і один вихід. Для цього умовні оператори (що мають два виходи) або включають усередину модуля, поєднуючи їх з операторами обробки, або виносять у систему між модульних зв'язків, формуючи тим самим схему алгоритму більш високого рангу.

У міжнародному стандарті на програмний продукт HIPO (Hierarchy Input Process Output) декларується аналогічний підхід до розробки прикладних програм.

Розробка схеми алгоритму функціонального модуля програми має яскраво виражений ітеративний характер, тобто вимагає багаторазових проб, перш ніж виникає впевненість, що алгоритм реалізації процедури правильний і завершений. Незалежно від функціонального призначення процедури при розробці її схеми необхідно дотримуватися наступної черговості роботи:

– Визначити, що повинен робити модуль.

– Визначити способи одержання модулем вхідних даних (від датчиків через порти введення, з таблиць у пам'яті або через робочі регістри).

– Визначити необхідність якої-небудь попередньої обробки введених вихідних даних (маскування, зрушення, масштабування, перекодування й т. п.).

– Визначити метод перетворення вхідних даних на необхідні вихідні. Використовуючи оператори процедур і умовні оператори ухвалення рішення, відобразити мовою схеми алгоритм обраного методу.

– Визначити способи видачі з модуля оброблених даних (передати у пам'ять, програму або в порти виводу).

– Визначити необхідність якої-небудь вторинної обробки виведених даних (зміна формату, перекодування, масштабування, маскування й т. п.).

– Повернутися до пункту 1 цього переліку й проаналізувати отриманий результат. Виконати ітеративне коректування схеми алгоритму з метою зробити її простою, логічною, стрункою, що володіє чітким графічним образом.

– Перевірити працездатність алгоритму на папері шляхом підстановки в нього дійсних даних. Переконатися в його збіжності й результативності.

– Розглянути граничні випадки й спробувати визначити граничні значення інформаційних об'єктів, з якими оперує алгоритм, за межами яких він втрачає властиві кінцівки, збіжності або результативності. Особливу увагу при цьому слід приділити аналізу можливих ситуацій переповнення розрядної сітки, зміни знака результату операції, діленню на змінну, яка може набути нульового значення.

– Провести уявний експеримент з визначення працездатності алгоритму в реальному масштабі часу, коли стохастичні події, що відбуваються в об'єкті керування, можуть вплинути на роботу алгоритму. При цьому самому ретельному аналізу слід піддати реакцію алгоритму на можливі переривання з метою визначення критичних операторів, які необхідно захистити від переривань. Крім того, під час цього уявного експерименту слід проаналізувати логіку алгоритму з метою визначення таких послідовностей операторів, при виконанні яких мікроконтролер може «не помітити» короткочасних подій в об'єкті керування. При виявленні таких ситуацій у логіку слід внести корективи.

Практика розробки програмного забезпечення показала, що послідовне використання описаної поетапної процедури, що становить основу методу структурного програмування, дозволяє впевнено одержувати працездатні прикладні програми.

Перетворення розробленої схеми алгоритму у вихідний текст програми – справа нескладна. Але перш ніж почати писати програми, необхідно специфікувати пам'ять і вибрати мову програмування.

Специфікація пам'яті й робочих регістрів полягає у визначенні адреси першої команди прикладної програми, дійсних початкових адрес стека, таблиць даних, буферних зон передачі параметрів між процедурами, підпрограм обслуговування переривань і т. д. При цьому слід пам'ятати, що в мікроконтролерах пам'ять програм і пам'ять даних фізично й логічно розподілені.

При написанні програм на асемблері для I8051 необхідно враховувати особливості архітектури мікроконтролера й додержуватися певних рекомендацій, які сформульовані нижче:

– При скиданні мікроконтролер починає виконувати команди з нульової адреси, де розташовується 8-байтний оброблювач переривання. Тому перша команда цього оброблювача повинна реалізувати перехід до основної програми й виконати обхід таблиці векторів переривань.

– Бажано, щоб програма працювала з одним 8-байтовим банком регістрів. Це скоротить обсяг програми й прискорить її виконання.

– Часто використовувані змінні слід розміщати в перших 256 байтах пам'яті, що також скоротить обсяг і прискорить виконання програми.

– Бажано, щоб оброблювач переривання мав обсяг не більш 8 байт, щоб його можна було розмістити у відповіднім вікні.

– При виконанні програм, які контролюють час виконання, переривання повинні бути заборонені. Наприклад, при виконанні ділянки коду, що програмно формує послідовність сигналів на якому-небудь інтерфейсі (приміром, шина I^2C), де є тверді рамки часових характеристик сигналів.

– При використанні безпосередньої адресації не забувайте ставити перед константою символ «#». Інакше константа буде інтерпретуватися як адреса одного з перших 256 байт пам'яті.

Виконання цих рекомендацій при розробці прикладних програм дозволить звести до мінімуму проблеми, які можуть виникнути надалі при налагодженні програми.

Що стосується додаткової інформації з написання програм мовою асемблера для I8051, то вона залежить від конкретного використовуваного асемблера. Формат написання програм практично однаковий для різних асемблерів, але для нормальної їхньої роботи, можливо, знадобиться ввести певні директиви. Ми будемо розглядати вільно-розповсюджуваний макро-асемблер версії 2.2 виробництва Intel Corporation.

2.2 Процедури й підпрограми

При розробці мікроконтролерних систем можуть бути використані два способи організації прикладних програм: монолітний і модульний. При першому способі вся прикладна програма розробляється як єдине ціле. При другому вона будується з окремих програмних блоків, кожний з яких реалізує деяку процедуру обробки даних або керування. Взаємозв'язок блоків визначається розроблювачем при монтажі із цих блоків закінченої прикладної програми.

Окремі фрагменти прикладної програми можуть бути отримані у вигляді лінійної послідовності блоків, інші (багаторазово використовувані) звичайно оформляються у вигляді підпрограм, до яких прикладна програма, названа основною, має можливість звернутися в міру необхідності. Підпрограма повинна мати наступні властивості: виконувати закінчену процедуру обробки даних, мати тільки один вхід і один вихід і не мати ефект післядії, при якому поточне виконання підпрограми виявляло б вплив на її наступні виконання.

Звертання до підпрограми здійснюється за командою виклику CALL MARK, де MARK – символічне ім'я процедури. Ім'я процедури використовується як мітки, що відзначає одну з команд (найчастіше першу) підпрограми. Для I8051 мнемонічне значення CALL є узагальненим і транслюється в одну з команд ACALL або LCALL залежно від адресної відстані поточної команди до викликуваної підпрограми.

За командою CALL у стеці зберігається значення лічильника команд, і повернення з підпрограми здійснюється в те місце основної програми, звідки був здійснений виклик (до команди основної програми, що випливає за командою CALL). Для цього будь-яка підпрограма повинна закінчуватися командою повернення RET, що здійснює відновлення вмісту програмного лічильника зі стека.

Досить часто виникає необхідність такої організації обчислювального процесу, при якій підпрограма викликає іншу підпрограму, та у свою чергу викликає наступну і т. д. Цей процес називається вкладенням підпрограм. Кількість підпрограм, які можуть бути викликані таким способом, (глибина вкладеності підпрограм) обмежується тільки ємністю стека.

При такому інтенсивному використанні стекової пам'яті покажчик стека (SP) настроюють якнайдалі від чарунки пам'яті, де зберігаються дані або проводяться тимчасові записи. Настроювання стекового покажчика, при цьому, повинне виконуватися на самому початку програми – при ініціалізації. Для більш точної величини використовуваної стекової пам'яті розраховують максимальну вкладеність підпрограм і збережень регістрів у стеці одночасно, а також необхідно врахувати виклик переривання й кількість пам'яті для збереження стану процесора в оброблювачі переривань, даних й інших записів у стек.

Іноді при звертанні до підпрограми виникає необхідність зберегти не тільки адресу повернення в основну програму, але й уміст окремих робочих регістрів. Зручним способом для цього є перемикання банку регістрів. Наприклад, якщо основна програма використовує банк регістрів 0, то підпрограма може використовувати банк регістрів 1. Однак перемикання банку регістрів не забезпечує збереження вмісту акумулятора, що викликає необхідність створювати в одному з робочих регістрів або в пам'яті копію акумулятора.

Для успішної роботи будь-якої підпрограми необхідно однозначно визначити спосіб передачі в неї вихідних даних і спосіб виводу результату її роботи. Підпрограма, якій потрібна додаткова інформація у вигляді параметрів її настроювання або операндів, називається параметризованою.

Одержали поширення чотири способи передачі параметрів: через пам'ять, через регістри загального призначення (РЗП), через регістр ознак і через стек.

При передачі вхідних параметрів через пам'ять основна програма обов'язково містить команди завантаження деяких чарунок пам'яті, а підпрограма – команди зчитування із цих чарунок. При передачі вхідних параметрів підпрограма повинна зчитувати деякі чарунки пам'яті, а основна програма – завантажити туди дані.

Передача параметрів через регістри здійснюється аналогічним чином.

Третій спосіб передачі параметрів – через регістр ознак. Його зручно використовувати при передачі вихідних параметрів (наприклад, у підпрограмах порівняння чисел). У цьому випадку підпрограма повинна встановити (або скинути) відповідні ознаки, а основна програма – проаналізувати їхнє значення. I8051 має більші можливості для передачі параметрів через ознаки. У ньому є 128 прапорів користувача, доступних для модифікації й аналізу.

Спосіб передачі через стек дозволяє використовувати в якості параметра вміст лічильника команд.

Використання процедур, оформлених у вигляді підпрограм, при розробці програмного забезпечення має ряд гарних властивостей. Насамперед, відносно прості модулі, виділені зі складної програми, можуть програмуватися декількома розроблювачами з метою скорочення часу проектування. Ще більш важливим є те, що будь-яка підпрограма допускає автономне налагодження. Це, як правило, багаторазово скорочує час налагодження всього прикладного програмного забезпечення. І, нарешті, механізм використання підпрограм зменшує довжину прикладної програми, що має своїм наслідком зменшення ємності, що вимагається, пам'яті програм.

Істотним є й та обставина, що налагоджені процедури організуються розроблювачами в бібліотеки параметризованих підпрограм і можуть бути багаторазово використані в проектній роботі. Бібліотека підпрограм повинна будуватися на основі «угоди про єдиний спосіб обміну параметрами».

2.3 Основні правила запису вихідного тексту програми

Вихідний текст програми являє собою послідовність операторів мови, згрупованих у сегменти й оформлених у вигляді файла.

Оператор – це базова конструкція мови програмування, що визначає дії в програмі. У мові програмування ASM-51 в одному рядку може бути записаний тільки один оператор. Максимальний розмір рядка – 255 символів. Ознакою кінця оператора є символ «повернення каретки».

Оператор складається із трьох полів:

<поле мітки> <поле операції> <поле коментаря>,

Кожне з полів, у тому числі й усі поля, можуть бути відсутніми. Оператор, у якому усі поля відсутні, називається порожнім оператором. Він використовується для збільшення наочності програми.

Приклад оператора, записаного мовою програмування ASM-51, відображено на рисунку 13.

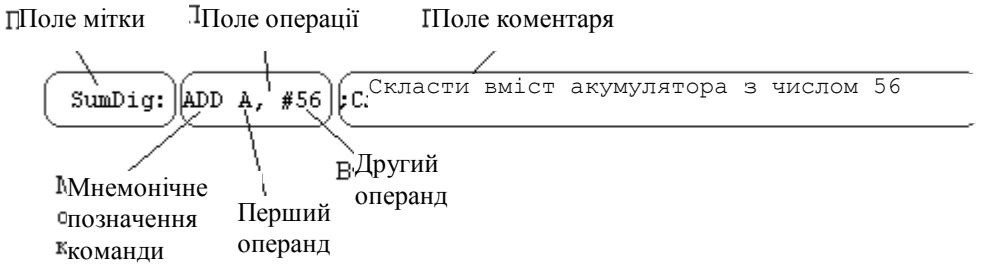


Рисунок 13 – Приклад запису оператора мовою ASM-51

Поле мітки використовується для запису міток. Мітки використовуються для організації умовних і безумовних переходів, а також для оголошення змінних і констант. Ознакою кінця поля мітки є символ «двокрапка». Однак мова програмування ASM-51, у вигляді виключення, допускає використовувати символи інтервалу, як ознаку кінця поля мітки.

Якщо в операторові присутня тільки мітка, то вона позначає найближчий наступний оператор, у якому є присутня інструкція процесора або директива асемблера. Використання оператора, що містить тільки мітку, може бути викликане або занадто великою довжиною самої мітки,

або необхідністю привласнити одному непустому операторові декілька міток.

Приклад використання оператора, що містить тільки мітку:

```
Podprogammaperedachidannix:  ; Позначається наступний оператор
                MOV  R0,A    ; <- Оператор, що позначається
                MOV  A,@R0
```

Поле операції використовується для запису директиви мови або інструкції мікроконтролера, які складаються із мнемонічного позначення команди мікроконтролера й одного або декількох операндів. У якості операндів можуть використовуватися адреси комірок пам'яті, позначення регістрів або мітки операторів. Операнди відділяються один від одного комами. Разом з комами для збільшення читаності програми допускається використання символів інтервалу.

Поле коментаря починається із символу «крапка з комою». Це поле використовується для запису пояснень до програми. Оператор, у якому є присутнім тільки поле коментаря, використовується для збільшення наочності програми.

Коментар починається із символу (;) і може містити будь-які ASCII символи. Приклади коментарів:

```
;-------------------------------------------------------------------*
; ПІДПРОГРАМА ОБЧИСЛЕННЯ ФУНКЦІЇ                                    |
;-------------------------------------------------------------------*
; X+Y*Z
```

Для набору тексту програми використовуються наступні символи:

– Символи інтервалу.

– Букви.

– Знаки.

– Цифри.

Символи інтервалу визначають один або кілька пробілів у пропозиції вихідного модуля. До цих символів відносяться «пробіл» і «табуляція».

Найменування знаків і їх позначення наведено в таблиці 1.

Таблиця 1 – Припустимі знаки

Найменування	Позначення
1	*2*

Номер	#
Знак грошової одиниці	$
Апостроф	'
Кругла дужка ліва	(
Кругла дужка права	)
Зірочка	*
Плюс	+
Кома	,

Продовження таблиці 1

Мінус	-
Крапка	.
Дробова риса	/
Двокрапка	:
Крапка з коми	;
Менше	<
Рівно	=
Більше	>
Знак питання	?
Комерційне эт	@

У якості **букв** сприймаються латинські букви верхнього й нижнього регістру.

ASCII символи, що не входять до переліку основних символів алфавіту мови, уважаються додатковими. Ці символи можуть використовуватися для пояснень у вихідному тексті програми, а також для визначення символьних констант.

Із символів формуються ідентифікатори й числа.

Ідентифікатор – це символічне позначення об'єкта програми. У якості ідентифікатора може бути використана будь-яка послідовність букв і цифр. При цьому в якості букви може бути використана будь-яка буква латинського алфавіту, а також знак питання (?) і знак «нижнє підкреслення» (_). **Ідентифікатор може починатися тільки з букви.** Це

дозволяє відрізняти його від числа. В ідентифікаторах мова програмування ASM-51 розрізняє букви верхнього й нижнього регістрів.

Кількість символів в ідентифікаторі обмежене довжиною рядка (255 символів). Транслятор розрізняє ідентифікатори за першими 31 символами.

Приклади ідентифікаторів:

ADD5, FFFFH, ALFA_1.

У мові програмування ASM-51 є три категорії ідентифікаторів: ключові слова, вбудовані імена й обумовлені імена.

Ключове слово є визначальною частиною оператора мови асемблера. Значення ключових слів мови асемблера ASM-51 не можуть бути змінені або перевизначені в програмному модулі яким-небудь чином. Ключовому слову не може бути призначене ім'я-синонім. **Ключові слова можуть бути написані буквами як верхнього, так і нижнього регістрів**.

У мові ASM-51 є наступні категорії ключових слів:

– Інструкції.

– Директиви.

– Вбудовані імена.

– Операції.

Інструкції з форми запису збігаються із мнемонічними позначеннями команд мікроконтролерів сімейства MCS-51 і разом з операндами становлять команди мікроконтролера.

Директиви разом з **допоміжними словами** визначають дії в програмі, які повинні бути виконані асемблером у процесі перетворення вихідного тексту програми в об'єктний код.

Операції виконуються асемблером у процесі обчислення виражень на етапі трансляції вихідного тексту програми для визначення конкретного числа, яке використовується в команді.

Вбудовані імена привласнені адресам регістрів спеціальних функцій, адресам прапорів спеціальних функцій, робочим регістрам R0-R7 поточного банку регістрів, а також акумулятору A і прапору переносу C.

Обумовлені імена оголошуються користувачем. У мові програмування ASM-51 є наступні категорії обумовлених ідентифікаторів:

– Мітки.

– Внутрішні й зовнішні змінні адресного типу.

– Внутрішні й зовнішні змінні числового типу.

– Імена сегментів.

– Назви програмних модулів.

У мові програмування ASM-51 використовуються цілі беззнакові числа, подані у двійковій, восьмеричній, десятковій та шістнадцятерічній формах запису. Для визначення підстави системи числення використовується суфікс (буква, що становиться за числом):

B –двійкове число;

Q або O – вісімкове число;

[D] – десяткове число (суфікс допускається пропускати);

H – шістнадцяткове.

Для десяткового числа суфікс може бути відсутнім. Кількість символів у числі обмежене розміром рядка, однак значення числа визначається за модулем 2.

Приклади запису чисел:

```
011101b, 1011100B, 735Q, 456o, 256, 0fah, 0CBH
```

Число завжди починається із цифри. Це необхідно для того, щоб відрізняти шістнадцятерічне число від ідентифікатора: ADCH – ідентифікатор; 0ADCH – число.

Часто буває зручно виконати деякі обчислення для того, щоб одержати число. Мова програмування ASM-51 дозволяє виконувати беззнакові операції над числами. У таких вираженнях припустимо використовувати арифметичні операції:

– «+» підсумовування.

– «-» вирахування.

– «*» множення.

– «/» ділення.

– «mod» обчислення залишку від цілочисельного ділення.

У мові програмування ASM-51 також визначена одномісна операція . Для неї потрібен один операнд, якому вона передує. Для зміни порядку виконання операцій можна скористатися дужками. Крім арифметичних операцій у вираженнях припустиме використання логічних операцій:

– «not» побітова інверсія операнда.

– «and» логічне «і».

– «or» логічне «або».

– «xor» виключне «або» (підсумовування за модулем два).

– Функцій виділення старшого HIGH і молодшого LOW байта шістнадцятирозрядного числа.

Приклад використання виражень для визначення числової константи показаний на рисунку 14.

```
init:
;--- Настроить Timer 0 ---------------------------------------
  mov   TMOD, #00000001b    Выражение для определения константы
                ;    ||||
                ;    ||++--Выбрать режим 16-разрядного таймера
                ;    |+----Использовать внутреннюю синхронизацию
                ;    +-----Запретить управление таймером от INT0

  mov   TL0,  #LOW(-(F_ZQ/12)*10)    ;Настроить таймер
  mov   TH0,  #HIGH(-(F_ZQ/12)*10)   ;на период 10мс

  setb TR0                  ;Включить таймер 0
;-------------------------------------------------------------
```

Рисунок 14 – Використання виражень числової константи мовою ASM-51

Часто число використовується для опису символів. У цьому випадку для визначення числа можна скористатися літеральною константою «’». Літеральна константа полягає в апострофи:

```
MOV  SBUF,#'A'
```

Для запису фраз у пам'яті програм можна скористатися літеральними рядками:

```
ERROR:          DB  'Помилка при передачі'
```

У цьому випадку кожний символ заміняється окремим байтом і запам'ятовується в ПЗУ пам'яті програм.

2.4 Директиви мови ASM-51

2.4.1 Загальні відомості

Мова програмування асемблер завжди містить у собі машинні коди мікроконтролера, але цим не обмежується набір команд цієї мови. Справа в тому, що потрібно вміти управляти самим процесом трансляції програми. З

повним списком директив можна познайомитися безпосередньо в описі самого компілятора, а тут будуть докладно розглянуті деякі з них.

Перше, що незручно при використанні тільки машинних команд – це необхідність пам'ятати, які дані в якій комірці пам'яті перебувають. При читанні програми важко відрізнити константи від змінних, адже вони відрізняються в командах тільки видом адресації. Подолати ці труднощі можна за допомогою ідентифікаторів. Можна призначити який або комірці пам'яті ідентифікатор, і тим самим працювати із цим ідентифікатором як зі змінною. Такий спосіб застосовується при розробці складних і трудомістких програм.

Асемблер підтримує ряд директив, які дозволяють дати символічне визначення змінним, резервують та ініціалізують простір пам'яті, визначають розташування згенерованого об'єктного коду в пам'яті. За винятком DB і DW директиви не роблять об'єктний код. Директиви використовуються, щоб змінити стан асемблера, визначити об'єкти й додати інформацію до об'єктного файла.

Директиви асемблера можуть бути розділені на ряд категорій:

– Символічні визначення.

– Резервування простору пам'яті.

– Ініціалізація даних.

– Керування станом асемблера.

– Вибір сегментів.

– Визначення макрокоманд.

2.4.2 Директиви символічних визначень

Директиви символічних визначень можуть бути використані для того, щоб резервувати простір пам'яті, поставити у відповідність до символічних імен певні числові значення, регістри процесора й сегменти. Ці директиви вимагають, щоб ім'я символу було визначено поряд з адресою, числовим значенням, регістром або типом сегмента:

BIT – Визначає символічне ім'я, що посилається на адресу біта.

CODE – Визначає символічне ім'я, що посилається на адресу коду (для об'єктного коду).

DATA – Визначає символічне ім'я, що посилається на адресу резидентної пам'яті даних.

EQU – Призначає символічному імені числове значення або ім'я регістру.

IDATA – Визначає символічне ім'я, що посилається на побічно-адресовану адресу резидентної пам'яті даних (для об'єктного коду).

SEGMENT – Повідомляє ім'я переміщуваного сегмента, його тип і розташування (для об'єктного коду).

SET – Призначає символічне ім'я числовому значенню або регістру. Ім'я може бути згодом змінене за допомогою директиви SET.

XDATA – Визначає символічне ім'я, що посилається на адресу зовнішньої пам'яті даних (для об'єктного коду).

2.4.3 Директиви резервування й ініціалізації пам'яті

Ці директиви використовуються для резервування й ініціалізації слів, байтів або бітів. В абсолютному сегменті зарезервований простір починається з поточного адреси. У переміщуваному сегменті зарезервований простір починається з поточного зсуву. Покажчик розташування підтримується окремо для кожного сегмента, до нього можна звертатися, використовуючи символ ($):

DB – Затягає у пам’ять програм байтову константу.

DBIT – Резервує простір у бітовому сегменті (для об'єктного коду).

DS – Резервує простір пам'яті в поточному сегменті (для об'єктного коду).

DW – Ініціалізує пам'ять значенням слова.

2.4.4 Директиви компонування програми

Ви можете використовувати директиви компонування програми для того, щоб дати об'єктному модулю ім'я й визначити загальні й зовнішні символи. Ці директиви використовуються в L51 для об'єднання окремих об'єктних модулів у єдиний абсолютний об'єктний модуль. Дані директиви не використовуються при написанні програми без об'єктних кодів:

EXTRN – Визначає символічні імена, які оголошені в інших об'єктних модулях.

NAME – Визначає ім'я об'єктного модуля.

PUBLIC – Визначає символічні імена, які можуть використовуватися в інших об'єктних модулях.

2.4.5 Директиви керування станом асемблера

Ці директиви використовуються для того, щоб повідомити про кінець трансляції програми, вибрати початкову адресу або зсув для сегмента, визначити використовуваний банк регістрів:

END – Повідомляє про кінець трансльованого модуля.

ORG – Змінює значення асемблерного лічильника адреси поточного сегмента програми.

USING – Вибирає номер банку регістрів загального призначення.

2.4.6 Директиви вибору сегмента

Наступні директиви визначають сегменти даних і коду (для об'єктного коду):

BSEG – Вибирає абсолютний бітовий сегмент.

CSEG – Вибирає сегмент програми в машинному коді.

DSEG – Вибирає абсолютний сегмент резидентної пам'яті даних.

ISEG – Вибирає абсолютний побічно адресований сегмент резидентної пам'яті даних.

RSEG – Вибирає попередньо певний переміщуваний сегмент.

XSEG – Вибирає абсолютний сегмент зовнішньої пам'яті даних.

2.4.7 Директиви макровизначень

Наступні директиви використовуються для визначення макрокоманд (для макроасемблера):

ENDM – Закінчує макровизначення.

EXITM – Змушує макророзширення негайно завершитися.

IRP – Визначає список аргументів.

IRPC – Визначає аргумент.

LOCAL – Визначає до 16 локальних символів, використовуваних усередині макрокоманди.

MACRO – Початок макровизначення, визначає ім'я макрокоманди й параметрів, які можуть бути передані макрокоманді.

REPT – Визначає кількість повторень наступних рядків.

2.5 Приклади використання директив

Директива equ дозволяє призначати імена змінних і констант. Тепер можна призначити змінній адресу в одному місці й користуватися ідентифікатором змінної у всій програмі. Правда за використання ідентифікатора саме в якості змінної відповідає програміст. Проте, якщо в процесі написання програми буде потрібно змінити адресу змінної, це можна зробити в одному місці програми, а не переглядати всю програму, розбираючись, чи є в даній конкретній команді число 10 константою, адресою чи кількістю повторів у циклі. Усі необхідні зміни зробить сам транслятор. Приклад призначення змінних наведений на прикладі опису інтерфейсу підключення рідкокристалічного індикатора (РКІ) до мікроконтролера. Гідність такого опису полягає в простому переносі програми на іншу, подібну систему, що має інше підключення індикатору:

```
Dispdat       EQU  P0     ; Шина даних РКІ на порту P0.
;
RDS           EQU  P1.2   ; Сигнал читання команди РКІ підключений до
                          ; лінії P1.2.
RW            EQU  P1.1   ; Сигнал вибору записи/читання РКІ підключений до
                          ; лінії P1.1
E             EQU  P1.0   ; Сигнал строб синхронізації РКІ підключений до
```

```
                        ; лінії P1.0
BUSY          EQU  P0.7   ; Сигнал зайнятості РКІ підключений до лінії P0.7
;
Setdd_Adr     EQU  80h    ; Адреса регістру даних/адреси усередині РКІ
Funcset       EQU  20h    ; Команда РКІ установки функцій
_8bit         EQU  10h    ; Режим 8 біт інтерфейс РКІ
_2line        EQU  8      ; Режим 2 рядкового РКІ
```

Як видно на наведеному прикладі, використання ідентифікаторів значно підвищує зрозумілість програми, тому що в назві змінної відображається функція, за яку відповідає дана змінна.

Один раз призначений ідентифікатор уже не може бути змінений надалі й при повторній спробі призначення точно такого ж імені ідентифікатора буде видане повідомлення про помилку.

Директива set. Якщо потрібно в різних місцях програми призначати тому самому ідентифікатору різні числа, то потрібно користуватися директивою set. Використання цієї директиви повністю ідентично використанню директиви equ, тому ілюструватися прикладом не буде.

Константи, призначувані директивою equ, можуть бути використані тільки в одній команді. Досить часто потрібна робота з таблицею констант, такий як таблиця перекодування, таблиці елементарних функцій або синдроми завадостійких кодів. Такі константи використовуються не на етапі трансляції, а зберігаються в пам'яті програм мікроконтролера. Для занесення констант в пам’ять програм мікроконтролера використовуються **директиви db** і **dw.**

Розглянемо застосування даних директив на прикладі. Нехай у нас є цифровий семисегментний індикатор із загальним катодом, підключений до мікропроцесорної системи згідно з рисунком 15.

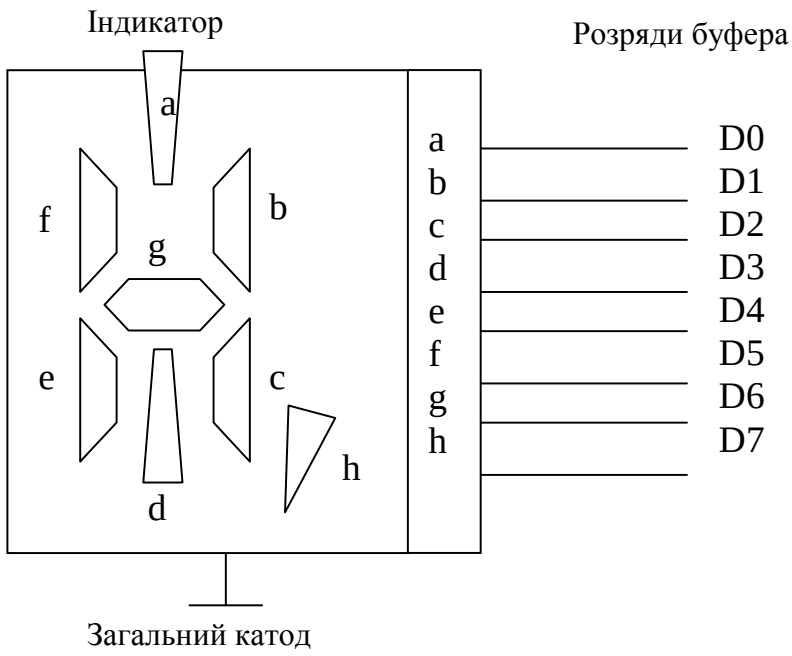


Рисунок 15 – Приклад підключення індикатору до МПС

При установці відповідного біта розряду вихідного буфера в логічну «1», запалюється відповідний сегмент. Наприклад, для відображення числа 1 необхідно встановити біти D1 і D2, відповідні до сегментів «b» і «c». Тоді, для виводу чисел на такий індикатор, необхідно виконати програмний дешифратор із числа в код семисегментного індикатору. Можна виконати табличний переклад, що нагадує пошук значення функції за таблицями Брадіса. Безпосередньо дані для світіння на індикаторі задаються директивою db:

```
;Підпрограма перекладу числа в код семисегментного індикатору
;Вхід:  A – ДД число формату 0X
;Вихід: A – семисегментний код числа
;-----------------------------------------------------------------------
Decode:
            MOV  DPTR, #table    ; Завантаження адреси таблиці.
           MOVC  A,@A+DPTR       ; Читання запису зі зсувом в A.
            RET
;
Table:        ;  hgfedcba
             DB  00111111b       ; Символ ‘0’
             DB  00000110b       ; Символ ‘1’
             DB  01011011b       ; Символ ‘2’
             DB  01001111b       ; Символ ‘3’
             DB  01100110b       ; Символ ‘4’
```

```
        DB  01101101b       ; Символ ‘5’
        DB  01111101b       ; Символ ‘6’
        DB  00000111b       ; Символ ‘7’
        DB  01111111b       ; Символ ‘8’
        DB  01101111b       ; Символ ‘9’
```

Ця ж директива дозволяє легко записувати написи, які надалі буде потрібно висвітлювати на вбудованому дисплеї або екрані дисплея універсального комп'ютера, підключеного до розроблювального пристрою через який-небудь інтерфейс. Приклад використання директиви db для занесення написів в пам’ять програм мікроконтролера наведений нижче:

```
Decode:
     MOV  R7,#endnadp-nadpsvjazust   ;В R7 завантажуємо число символів.
     MOV  DPTR,#nadpsvjazust         ;Підготувати до передачі перший символ
Putnextchar:
     CLR  A
    MOVC  A,@A+DPTR                  ;Читаємо черговий символ
     INC  DPTR
    CALL  Putchar                    ;Відправляємо символ на передачу
    DJNZ  R7,Putnextchar             ;Останній символ?
     RET                             ; Так, повернення з підпрограми

Nadpsvjazust: DB ‘Зв'язок установлений’,10,13
Endnadp:
```

Директива dw дозволяє завантажувати у пам’ять програм двобайтні числа. У цій директиві, як і в директиві db числа можна вказувати через кому. Приклад лістингу фрагмента:

```
 0023 0001              294  dw 1,2,0abh,’a’,’QW’
 0025 0002
 0027 00AB
 0029 0061
 002B 5157
```

Іноді потрібно розташувати команду за певною адресою. Найбільш часто це потрібно при використанні переривань, коли перша команда програми-оброблювача переривань повинна бути розташована точно на векторі переривання. Це можна зробити, використовуючи команду NOP для заповнення проміжків між векторами переривання, але краще скористатися директивою ORG.

Директива org призначена для запису в лічильник адреси сегмента значення свого операнду. Тобто за допомогою цієї директиви можна помістити команду (або дані) у пам'яті мікроконтролера за будь-якою

адресою. Приклад використання директиви ORG для розміщення підпрограм обробки переривань на векторах переривань показаний нижче:

```
Reset:
      LJMP Main          ; Перехід на початок основної програми.
; Переривання переповнення таймера 0. ----------------------------------
      ORG 0bh            ; Вектор переривання таймера 0.
      LJMP Intt0         ; Перехід на оброблювач переривання T/C0.
; Переривання послідовного порту. ---------------------------------------
      ORG 23h            ; Вектор переривання послідовного порту.
      LJMP Intserport
; Для частоти кварцового резонатора 12Мгц.
Intt0:
      Mov TL0, #LOW(-(Fosc/12)*10-2)      ;Настроювання таймера
      Mov TH0, #HIGH(-(Fosc/12)*10-2)     ;на 10мкс.
      Reti
;Початок основної програми мікроконтролера.
Main: Mov SP,#vershsteka                  ; Настроювання покажчика стека.
      Call Init          ; Виконати п/п ініціалізації мікроконтролера.
;-------------------------------------------------------------------------
```

Необхідно відзначити, що при використанні цієї директиви можлива ситуація, коли програміст наказує транслятору розмістити новий код програми за вже написаними місцями, що приводить до невірної трансляції програми.

Директива using. При використанні переривань критичним є час, займаний програмою, оброблювачем переривань. Цей час можна значно скоротити, виділивши для обробки переривань окремий банк регістрів. Виділити окремий банк регістрів можна за допомогою директиви USING. Номер банку використовуваних регістрів вказується в директиві в якості операнда. Приклад використання директиви USING для підпрограми обслуговування переривань від таймера 0 наведений нижче:

```
_code segment code
CSEG AT 0bh              ; Вектор переривання від T/C0.
      Jmp Tntt0

rseg _code
      USING 2
Intt0:
      Push PSW           ; Зберігаємо регістри в стек.
      Push ACC
      Mov PSW,#00010000b                  ;Включаємо банк 2 РОН.
      Mov TL0, #LOW(-(Fosc/12)*10-2)      ;Настроювання таймера
      Mov TH0, #HIGH(-(Fosc/12)*10-2)     ;на 10мкс.
      Pop ACC
```

```
Pop PSW
Reti
```

Директива CALL. У системі команд мікроконтролера MCS-51 використовується три команди безумовного переходу. Вибір конкретної команди залежить від розташування її в пам'яті програм, однак програміст звичайно цього не знає. У результаті, щоб уникнути помилок, доводиться використовувати саму довгу команду LJMP. Це призводить до більш довгих програм і до додаткового навантаження на редактор зв'язків. Транслятор сам може підібрати найкращий варіант команди безумовного переходу. Для цього замість команди мікроконтролера слід використовувати директиву call.

2.6 Реалізація підпрограм

При написанні програм часто при реалізації алгоритму роботи пристрою доводиться повторювати ті самі оператори (наприклад, оператори, що працюють із паралельним або послідовним портом). Було б непогано використовувати ту саму ділянку коду, замість того, щоб повторювати ті самі оператори кілька раз.

Ділянка програми, до якої можна звертатися з різних місць програми для виконання деяких дій, називається **підпрограмою.**

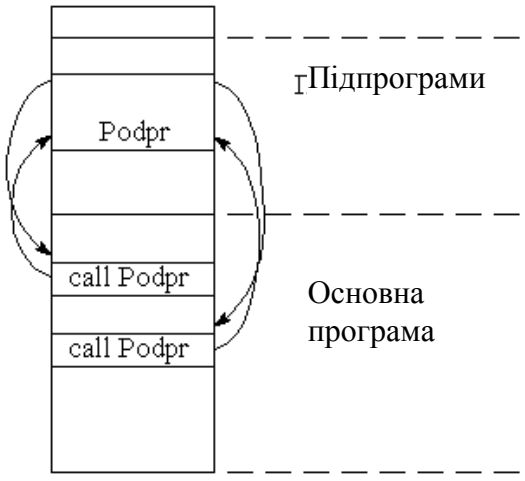


Рисунок 16 – Виклик підпрограми й повернення до виконання основної програми

Проблема, з якої доводиться зустрічатися при багаторазовому використанні ділянок кодів – це в яке місце пам'яті програм вертатися після завершення підпрограми. Звертання до підпрограми проводиться з декількох місць основної програми. Описану ситуацію ілюструє рисунок 16. На цьому рисунку зображений адресний простір мікроконтролера. Молодші адреси адресного простору на цьому рисунку перебувають у нижній частині.

Для звертання до підпрограми й повернення з неї до системи команд мікропроцесорів уводять спеціальні команди. У мікроконтролерах сімейства MCS-51 це команди LCALL, ACALL для виклику підпрограми й команда RET для повернення з підпрограми. Ці команди не тільки здійснюють передачу керування на зазначену адресу, але й запам'ятовують адресу команди, що виконується за командою виклику підпрограми в стековій пам'яті. Команда повернення з підпрограми RET передає керування команді, адреса якої була запам’ятована командою виклику підпрограми. Приклад використання підпрограми мовою програмування ASM-51 наведений нижче:

```
      ...
      Mov G_Per,#56          ; Передати 56 через послідовний порт.
      Call Peredatbyte
      ...
      Mov G_Per,#49          ; Передати 49 через послідовний порт.
      Call Peredatbyte
      ...
;-------------------------------------------------------------------------*
; Підпрограма передачі байта по послідовному портові.                     |
;-------------------------------------------------------------------------*
Peredatbyte:
      Jb TI,$           ; Очікування кінця передачі попереднього байта.
      Mov SBUF,G_Per    ; Передача байта
      Ret
```

У жодному разі не можна попадати в підпрограму будь-яким способом крім команди виклику підпрограми CALL. А якщо ні, то команда повернення з підпрограми передасть керування випадковій адресі. За цією адресою можуть бути розташовані дані, які в цьому випадку будуть інтерпретовані як програма, або звернення до зовнішньої пам'яті, звідки будуть зчитуватися випадкові числа.

Дуже часто потрібно з однієї підпрограми звертатися до іншої підпрограми. Таке звертання до підпрограми називається вкладеним. Кількість вкладених підпрограм називається рівнем вкладеності підпрограм. Максимально припустимий рівень вкладеності підпрограм визначається кількістю комірок стекової пам'яті. Логічно ці комірки пам'яті організовані так, щоб зчитування останнього записаного адреси проводилося першим, а першого записаного адреси проводилося останнім (буфер LIFO). Така логічна організація формується спеціальним лічильником. Цей лічильник, як було зазначено вище, називається покажчиком стека **SP**. Комірка пам'яті, у яку в цей момент може бути записана адреса повернення з підпрограми, називається **вершиною стека.** Кількість комірок пам'яті, призначених для організації стека, називається глибиною стека. Остання комірка пам'яті, у якій можна робити запис, називається **дном стека**. Логічна організація стека наведена на рисунку 17.

У мікроконтролерах сімейства MCS-51 при занесенні інформації в стек уміст покажчика стека збільшується (стік росте нагору), тому стік розміщається в самій верхній частині пам'яті даних. Для того, щоб установити глибину стека 28 байт, необхідно відняти з адреси максимальної комірки внутрішньої пам'яті мікроконтролера глибину стека й записати отримане значення в покажчик стека SP:

```
Dnosteka EQU 127        ; Обсяг внутрішнього ОЗУ I8051 – 128 байт.
Mov SP,#dnosteka-28     ; Установимо глибину стека 28 байт.
```

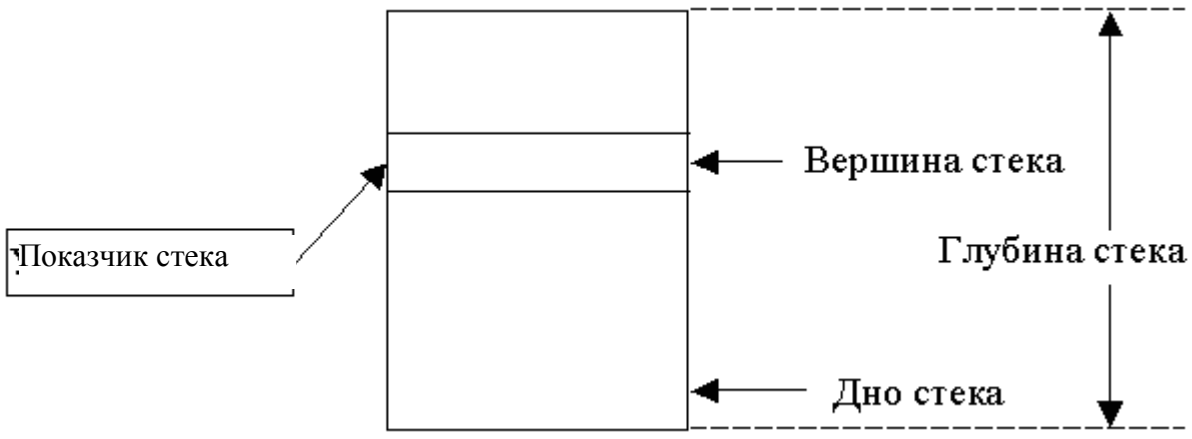


Рисунок 17 – Організація стека в пам'яті даних мікропроцесора укр.!

Крім умісту програмного лічильника часто потрібно запам'ятовувати вміст внутрішніх регістрів і прапорів процесора, локальних змінних підпрограми. Стік виявився зручним засобом і для цього завдання. Збереження локальних змінних у стеці дозволило здійснювати виклик підпрограми самої із себе (реалізовувати рекурсивні алгоритми). Це призвело до введення в систему команд спеціальних команд роботи зі стеком. У мікроконтролерах сімейства MCS-51 – це команди PUSH і POP. Використання цих команд показується на наступному прикладі:

```
Podprogramma:
      PUSH PSW          ; Зберігаємо використовувані в підпрограмі
      PUSH ACC          ; регістри в стеці.
      PUSH R0

      ...               ; Код самої підпрограми.

      POP R0            ; Витягаємо збережені регістри зі стека
      POP ACC           ; у зворотному порядку.
      POP PSW
      RET               ; Вихід з підпрограми.
```

У наведеному вище прикладі передачі байта через послідовний порт, сам байт передається в підпрограму через глобальну змінну G_Per. Однак програма буде ефективніше при використанні підпрограми з параметрами. Ми знаємо, що параметр підпрограми – це локальна змінна. У цьому випадку можуть значно знизитися вимоги до пам'яті даних. Для розміщення локальних змінних найкраще використовувати внутрішні регістри процесора. Мовою ASM51 для передачі параметра розмірністю один байт звичайно використовується акумулятор.

Якщо в підпрограму потрібно передати двобайтове значення, то в якості параметра підпрограми використовується пара регістрів (звичайно регістри R6 – старший байт і R7 – молодший байт). Приклад програми, що передає в підпрограму двобайтове число, написану мовою програмування ASM51, наведений нижче:

```
      ...
;Приклад передачі в підпрограму двобайтового числа.
      Mov R7, #56       ; Передача молодшого байта.
      Mov R6, #0        ; Передача старшого байта.
      Call Podprog      ; Викликати підпрограму.
      ...
```

Якщо в підпрограму потрібно передати чотирибайтове значення (це потрібно для змінної, відповідної до типу long або float), то використовуються регістри R4...R7 (регістр R4 – старший байт):

```
      ...
;Приклад передачі в підпрограму чотирьохбайтового числа.
      Mov R7, #56       ; Передача молодшого байта.
      Mov R6, #0
      Mov R5, #0
      Mov R4, #0        ; Передача старшого байта.
      Call Podprog      ; Викликати підпрограму.
      ...
```

Регістри R0 і R1 звичайно використовуються в якості покажчиків оброблюваних змінних, таких як рядок або масиви. Якщо потрібно, щоб підпрограма обробила значний обсяг даних, то ці дані можна передати через параметр – покажчик. У якості покажчика при звертанні до зовнішньої пам'яті даних або до пам'яті програм звичайно використовується регістр-покажчик даних DPTR. Приклад передачі як параметра рядка, написаний мовою програмування ASM51, наведений нижче:

```
      ...
;Приклад передачі в підпрограму покажчика даних.
      Mov DPTR, #stroka ; Передача покажчика на рядок.
      Call Podprog      ; Викликати підпрограму.
      ...
Stroka: DB 'Наш рядок символів.'
```

При звертанні до масивів або структур, розташованих у внутрішній пам'яті даних як покажчик адреси використовується регістр R0 або R1:

```
      ...
;Приклад передачі в підпрограму покажчика даних внутрішньої пам'яті.
      Mov R0, #array    ; Передача покажчика на дані.
      Call Obrabotka    ; Викликати підпрограму.
      ...
```

Часто потрібно передавати результат обчислень із підпрограми в основну програму. Для цього можна скористатися функцією – підпрограмою-функцією. Підпрограма-функція повертає обчислене значення:

```
      Mov A,X           ; Передати в підпрограму значення X.
      Call Sin          ; Виклик функції Y=sin(X)
      Mov Y,A           ; Збереження функції в змінній Y.
```

При цьому змінні X і Y повинні бути заздалегідь визначені директивою EQU або SET.

2.7 Реалізація багатомодульних програм

Розбивка вихідного тексту програми на кілька файлів робить цей текст більш зрозумілим для програміста або декількох програмістів, що беруть участь у створенні програмного продукту. Однак залишається невирішеними ще кілька завдань:

– Програма-транслятор працює з усім вихідним текстом цілком, адже вона з'єднує всі файли перед трансляцією разом. Тому час трансляції вихідного тексту програми залишається значним (і навіть зростає). У той ж самий час програма ніколи не листується цілком. Звичайно змінюється тільки невелика ділянка програми.

– При призначенні змінних їхня кількість обмежена програмою-транслятором і може бути вичерпана при написанні програми.

– Різні програмісти, що беруть участь в створенні програмного продукту можуть призначати однакові імена для своїх змінних і при спробі з'єднання файлів у єдину програму звичайно виникають проблеми.

Усі ці проблеми можуть бути вирішені при роздільній трансляції програми. Тобто було б непогано вміти транслювати кожний файл із вихідним текстом програми окремо й з'єднувати потім готові трансльовані ділянки програми.

Компілятори, які дозволяють транслювати окремі ділянки програми, називаються **компіляторами з роздільною трансляцією.**

Вихідний текст програми, який може бути окремо трансльований, називається **програмним модулем.**

Трансльований програмний модуль зберігається у вигляді окремого файла в об'єктному форматі, де крім машинних команд зберігається інформація про імена змінних, адреси команд, що вимагають модифікації при об'єднанні модулів у єдину програму, й відладочна інформація.

Роздільна трансляція програми можлива при використанні двох програм: **транслятора** вихідного тексту програми й **редактора зв'язків.**

На перший погляд роздільна трансляція не повинна викликати яких-небудь проблем. Однак це не так. При компіляції вихідного тексту

програми транслятор становить таблицю посилань на константи, змінні й команди. Якщо при другому перегляді вихідного тексту програми, під час якого формується об'єктний модуль, транслятор не виявить імені змінної або мітки у своїй таблиці, то буде сформоване повідомлення про помилку, й об'єктний модуль буде стертий з диска комп'ютера.

Для того, щоб транслятор замість формування повідомлення про помилку записав в об'єктний модуль інформацію, необхідну для редактора зв'язків, потрібно використовувати спеціальні директиви посилань на зовнішні змінні або мітки. Звичайно ці директиви називаються PUBLIC (загальні) і EXTRN (зовнішні). Для посилання на змінну або мітку використовується директива EXTRN. У цій директиві перелічуються через кому мітки й змінні, точне значення яких редактор зв'язків повинен одержати з іншого модуля й модифікувати всі команди, у яких ці мітки або змінні використовуються. Приклад використання директиви EXTRN мовою програмування ASM51:

```
EXTRN DATA (Bufind, ERR)
EXTRN CODE (Podprog)
```

Для того, щоб редактор зв'язків міг здійснити зв'язування модулів у єдину програму, змінні й мітки, оголошені принаймні в одному з модулів як EXTRN, в іншому модулі повинні бути оголошені як доступні для всіх модулів за допомогою директиви PUBLIC. Приклад використання директиви PUBLIC:

```
PUBLIC Bufind, Parametr
PUBLIC Podprogr, ?Podprog?Byte
```

Використання декількох модулів при написанні програми збільшує швидкість трансляції й, в остаточному підсумку, швидкість написання програми. Однак оголошення змінних і імен підпрограм зовнішніх модулів захаращують вихідний текст модуля. Крім того, при використанні чужих модулів важко оголосити змінні й підпрограми без помилок. Тому звичайно оголошення змінних, констант і попередні оголошення підпрограм зберігають у файлах, що включаються, які називаються файлами-заголовками. Правилом гарного тону вважається при розробці програмного модуля відразу ж написати файл-заголовок для цього модуля, який може бути використаний програмістами, що працюють із вашим програмним модулем.

Для об'єднання декількох модулів у програму, що виконується, імена всіх модулів передаються в редактор зв'язків rl51.exe у якості параметрів при запуску цієї програми. Приклад виклику редактора зв'язків з командного рядка DOS для об'єднання трьох модулів:

rl51.exe progr.obj, modul1.obj, modul2.obj

У результаті роботи редактора зв'язків у цьому прикладі буде створений модуль, що виконується, з іменем progr. Формат запису інформації в цьому файлі залишається колишнім – об'єктний. Це дозволяє поєднувати модулі вроздріб, тобто при бажанні можна з декількох дрібних модулів одержати один більший.

Для одержання з об'єктного файла Нех-Файл необхідно об'єктний файл обробити програмою oh.exe.

2.8 Використання сегментів у мові програмування асемблер

Необхідно відзначити, що навіть, коли ми не замислюємося про сегменти, у програмі присутні два сегменти: сегмент коду програми й сегмент даних. Якщо уважно придивитися до програми, то можна виявити, що крім кодів команд у пам'яті програм зберігаються константи, тобто в пам'яті програм мікроконтролера розташовуються, принаймні, два сегменти: програма й дані. Чергування програми й констант у досить складній програмі, може призвести до небажаних наслідків. Внаслідок яких-небудь причин дані можуть бути випадково виконані в якості програми або навпаки програма може бути сприйнята й оброблена як дані.

Перелічені вище причини приводять до того, що треба виділити, принаймні, чотири сегменти:

– Програми.

– Стека.

– Змінних.

– Констант.

Приклад розміщення сегментів в адресному просторі пам'яті програм і внутрішньої пам'яті даних наведений на рисунку 18.

Рисунок 18 – Розбивання пам'яті програм і даних на сегменти

На цьому рисунку 18 видно, що при використанні декількох сегментів змінних у внутрішній пам'яті даних, редактор зв'язків може розмістити менший з них на місці невикористаних банків регістрів. Під сегмент стека звичайно приділяється вся область внутрішньої пам'яті, не зайнята змінними. Це дозволяє створювати програми з максимальним рівнем вкладеності підпрограм.

Найбільш простий спосіб визначення сегментів – це використання абсолютних сегментів пам'яті. При цьому способі розподіл пам'яті ведеться вручну точно так само, як це робилося при використанні директиви EQU. У цьому випадку початкова адреса сегмента жорстко задається програмістом, і він же стежить за тим, щоб сегменти не перекривалися один з одним у пам'яті мікроконтролера. Використання абсолютних сегментів дозволяє більш гнучко працювати з пам'яттю даних, тому що тепер байтові змінні в пам'яті даних можуть бути призначені за допомогою директиви резервування пам'яті DS, а бітові змінні за допомогою директиви резервування бітів DBIT.

Для визначення абсолютних сегментів пам'яті використовуються наступні директиви.

Директива BSEG дозволяє визначити абсолютний сегмент у внутрішній пам'яті даних з бітовою адресацією за певною адресою. Ця директива не призначає імені сегменту, тобто об'єднання сегментів з

різних програмних модулів неможливе. Для визначення конкретної початкової адреси сегмента застосовується атрибут AT. Якщо атрибут AT не використовується, то початкова адреса сегмента дорівнює нулю. Використання бітових змінних дозволяє значно заощаджувати внутрішню пам'ять програм мікроконтролера. Приклад використання директиви BSEG для оголошення бітових змінних:

```
      BSEG AT 8         ; Сегмент починається з 8 біта.
Rejind:     DBIT 1      ; Прапор режиму індикації.
Rejpriem:   DBIT 1      ; Прапор режиму приймання.
Flag:       DBIT 1      ; Прапор загального призначення.
```

Директива CSEG дозволяє визначити абсолютний сегмент у пам'яті програм за певною адресою. Ця директива не призначає імені сегменту, тобто об'єднання сегментів з різних програмних модулів неможливе. Для визначення конкретної початкової адреси сегмента застосовується атрибут AT. Якщо атрибут AT не використовується, то початкова адреса сегмента дорівнює нулю. Приклад використання директиви CSEG для розміщення підпрограми обслуговування переривання від таймера 0:

```
CSEG AT 0bh             ; Вектор переривання від T/C0.
Intt0:
      Mov TL0, #LOW(-(Fosc/12)*10-2)      ;Настроювання таймера
      Mov TH0, #HIGH(-(Fosc/12)*10-2)     ;на 10мкс.
      Reti
```

Директива DSEG дозволяє визначити абсолютний сегмент у внутрішній пам'яті даних за певною адресою. Передбачається, що до цього сегмента будуть звертатися команди із прямою адресацією. Ця директива не призначає імені сегменту, тобто об'єднання сегментів з різних програмних модулів неможливе. Для визначення конкретної початкової адреси сегмента застосовується атрибут AT. Якщо атрибут AT не використовується, то початкова адреса сегмента дорівнює нулю. Приклад використання директиви DSEG для оголошення байтових змінних:

```
      DSEG AT 20h       ; Розмістити сегмент із адреси, де можлива
                        ; бітова адресація ( Для можливості одночасно
                        ; бітової і байтової адресації).
Rejind:     DS 1        ; Змінна, що відображає стан програм
                        ; обслуговування апаратури.
Rejim:      DS 1        ; Змінна режиму роботи.
Massiv:     DS 10       ; Десятибайтовий масив.
```

У наведеному прикладі передбачається, що він пов'язаний із прикладом, наведеним вище. Тобто команди, що змінюють бітові змінні

Rejind, Rejpriem або Flag одночасно будуть змінювати вміст змінної Rejim, і навпаки: команди, що працюють зі змінною Rejim, одночасно змінюють уміст прапорів Rejind, Rejpriem або Flag. Таке оголошення змінних дозволяє написати найбільш ефективну програму керування контролером і підключеними до нього пристроями.

Директива ISEG дозволяє визначити абсолютний сегмент у внутрішній пам'яті даних за певною адресою. Ця директива не призначає імені сегмента, тобто об'єднання сегментів з різних програмних модулів неможливе. Для визначення конкретної початкової адреси сегмента застосовується атрибут AT. Якщо атрибут AT не використовується, то початкова адреса сегмента дорівнює нулю. Приклад використання директиви ISEG для оголошення байтових змінних:

```
      ISEG AT 40h       ; Розташувати сегмент в адресах від 40h.
Buffer:     DS 10       ; Змінна Buffer обсягом 10 байт.
Stack:      DS 245      ; Глибина стека 245 байт.
```

Якщо абсолютні адреси змінних або ділянок програм нецікаві, то можна скористатися переміщуваними сегментами. Ім'я переміщуваного сегмента задається директивою segment.

Директива segment дозволяє визначити ім'я сегмента й область пам'яті, де буде розміщатися даний сегмент пам'яті. Для кожної області пам'яті визначене ключове слово:

Data – розміщає сегмент у внутрішній пам'яті даних із прямою адресацією;

Idata – розміщає сегмент у внутрішній пам'яті даних з непрямою адресацією;

Bit – розміщає сегмент у внутрішній пам'яті даних з бітовою адресацією;

Xdata – розміщає сегмент у зовнішній пам'яті даних;

Code – розміщає сегмент у пам'яті програм.

Директива rseg. Після визначення імені сегмента можна використовувати цей сегмент за допомогою директиви rseg. Використання сегмента залежить від області пам'яті, для якої він призначений. Якщо це пам'ять даних, то в сегменті оголошуються байтові або бітові змінні. Якщо це пам'ять програм, то в сегменті розміщаються константи або ділянки

коду програми. Приклад використання директив segment і rseg для оголошення бітових змінних:

```
_data segment idata
      PUBLIC Vershsteka, Bufferklav

; Визначення змінних
      rseg _data
      Bufferklav DS 8   ; Обсяг буфера клавіатури 8 байт.
      Vershsteka:       ; Стек починається тут.
```

У цьому прикладі оголошений рядок buferklav, що складається з восьми байтових змінних. Крім того, у даному прикладі оголошена змінна Vershsteka, відповідна до останньої комірки пам'яті, використовуваної для зберігання змінних. Змінна Vershsteka може бути використана для початкової ініціалізації покажчика стека для того, щоб відвести під стек максимально доступну кількість комірок внутрішньої пам'яті. Це необхідно для того, щоб уникнути переповнення стека при вкладеному виклику підпрограм.

Оголошення й використання сегментів даних в області внутрішньої або зовнішньої пам'яті даних не відрізняється від наведеного прикладу за винятком ключового слова, що визначає область пам'яті даних.

Ще один приклад використання директив segment і rseg:

```
_bits segment bit
      PUBLIC Knizm, strvv

; Визначення бітових змінних
      rseg _bits
      Knizm: DBIT 1     ; Прапор натискання кнопки.
      strvv: DBIT 1     ; Прапор уведеного рядка.
```

У цьому прикладі директива segment використовується для оголошення сегмента бітових змінних.

Найбільший ефект від застосування сегментів можна одержати при написанні основного тексту програми з використанням модулів. Звичайно кожний програмний модуль оформляється у вигляді окремого переміщуваного сегмента. Це дозволяє редактору зв'язків скомпонувати програму оптимальним образом. При використанні абсолютних сегментів пам'яті програм доводиться це робити вручну, а тому що в процесі написання програми розмір програмних модулів постійно міняється, то доводиться вводити захисні області невикористовуваної пам'яті між програмними модулями.

Приклад використання переміщуваних сегментів у вихідному тексті програми:

```
_code segment code

CSEG AT 0               ; Початок програми.
Reset:
      Jmp Main

; Початок основної програми мікроконтролера-----------------------------
rseg _code
Main:
      Movx @DPTR,A
      Mov SP,#vershsteka      ; Настроїти покажчик стека на вершину
      Call Init               ; Виконати п/п ініціалізації процесора.
;-------------------------------------------------------------------------
```

У цьому прикладі наведена початкова ділянка основної програми мікроконтролера, на яку виконується перехід з нульової комірки пам'яті програм. Використання такої структури програми дозволяє в будь-який момент часу при необхідності використовувати кожний з векторів переривання, доступний у конкретному мікроконтролері, для якого пишеться ця програма. Досить помістити визначення цього вектора з використанням директиви cseg.

У наведеному прикладі використане ім'я переміщуваного сегмента _code. Воно було оголошено в найпершому рядку вихідного тексту програми. Конкретне ім'я переміщуваного сегмента може бути будь-яким, але як уже говорилося раніше воно повинне відображати те завдання, яке вирішує даний конкретний модуль.

3 СТВОРЕННЯ ПРОГРАМ ДЛЯ МІКРОКОНТРОЛЕРІВ

3.1 Структурне програмування

Створення програм для мікроконтролерів різко відрізняється від написання програм для універсального комп'ютера. При виконанні програми на універсальному комп'ютері запуск програм, взаємодія із внутрішніми, зовнішніми пристроями або людиною бере на себе операційна система. Програма, написана для мікроконтролера, повинна вирішувати всі ці завдання. Програма, написана для комп'ютера, коли-небудь запускається й завершується. Програма, яку керує мікроконтролер, запускається при включенні пристрою й не завершує свою роботу, поки не буде виключене живлення.

Програма для мікроконтролера повинна вирішувати всі перелічені вище проблеми. Найпростішим видом програми, яка може вирішувати поставлені завдання є монітор.

Алгоритм програми-монітора наведений на рисунку 19. Після включення живлення ця програма повинна настроїти мікросхему під виконуване програмою завдання, тобто настроїти певні ніжки мікросхеми на введення або вивід інформації, включити й настроїти внутрішні таймери мікроконтролера і так далі. Цей блок алгоритму програми-монітора називається ініціалізацією процесора.

Основна частина програми починає виконуватися після настроювання мікроконтролера на виконувану роботу. При цьому необхідно розуміти, що якщо в апаратурі введення, обробка й вивід інформації проводиться різними блоками, то при виконанні програми ці ж дії проводяться послідовно тим самим пристроєм – мікропроцесором. У цьому ж циклі передбачений блок обробки помилок. Його призначення: повідомляти оператора про непередбачену ситуацію, таке як неправильне введення із клавіатури або неправильні дані з підключеного до мікроконтролера пристрою.

При використанні декількох підпрограм виникає проблема обміну інформацією між цими підпрограмами. Як уже розглядалося раніше,

інформація в підпрограму може бути передана через параметри підпрограми або через глобальні змінні. При створенні програми-монітора може знадобиться передавати ту саму інформацію декільким підпрограмам, тому в моніторах інформація звичайно передається через глобальні змінні.

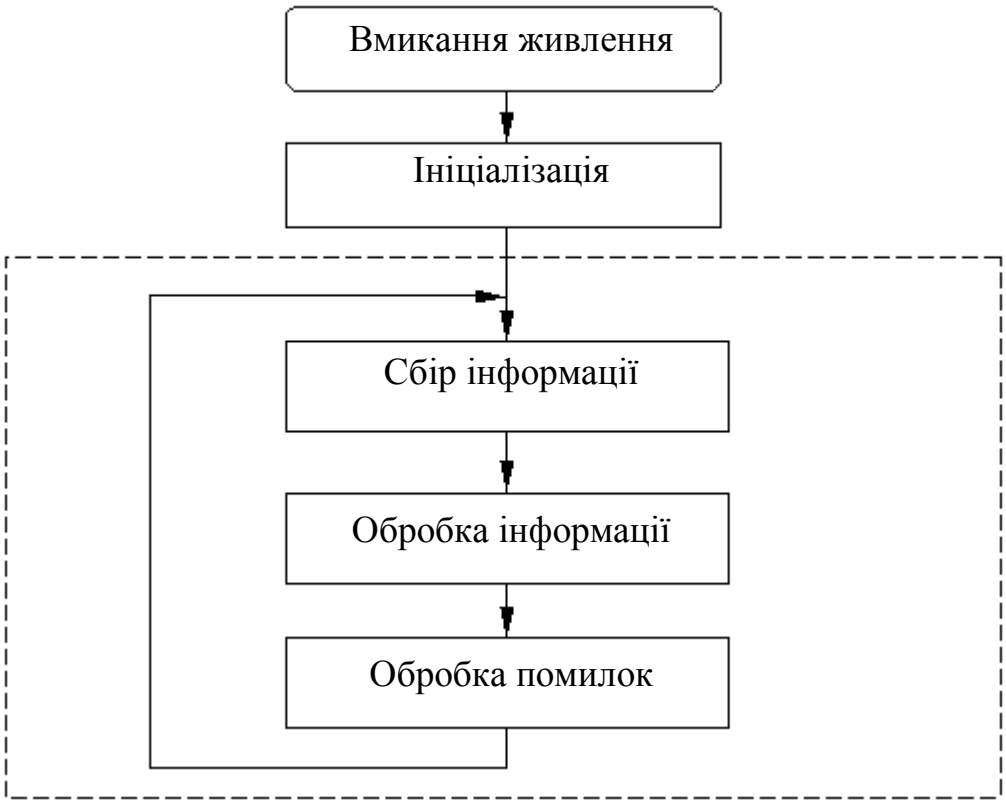


Рисунок 19 – Алгоритм програми-монітора

У цей час існує два способи написання програм: знизу нагору й згори вниз. При написанні програми знизу нагору почати налагодження програми неможливо, не написавши повністю всю програму. При написанні програми згори вниз на будь-якому етапі написання програми вона може бути відтрансльована й виконана, при цьому можна відстежити всі алгоритмічні дії програми, написані до цього часу. Процес написання програми не відрізняється від процесу створення алгоритму. Більше того, ці етапи створення програми можна об'єднати. Виконувана алгоритмічна дія відображається в назві підпрограми. Наприклад:

```
Call Readport          ; Прочитати порт
Call Indoff            ; Вимкнути Індикатор
```

Основна ідея структурного програмування полягає у тому, що існує тільки чотири структурні оператори. Використовуючи ці оператори, можна побудувати як завгодно складну програму.

Перший структурний оператор називається **лінійний ланцюжок операторів**. Будь-яке завдання може бути розбите на декілька підзавдань. Виконання підзавдань може бути доручене підпрограмі, у назві якої можна (і потрібно) відбити підзавдання, яке повинне вирішувати ця підпрограма. На момент написання алгоритму (і програми) верхнього рівня нас не цікавить, як буде вирішуватися це завдання, тому замість справжньої підпрограми поставимо підпрограму-заглушку (Підпрограма-заглушка це підпрограма, яка нічого не виконує, а тільки повертає керування головній програмі; дія, яку надалі повинна виконувати ця програма, відображається в назві підпрограми-заглушки).

Алгоритмічно такий ланцюжок операторів зображений на рисунку 20.

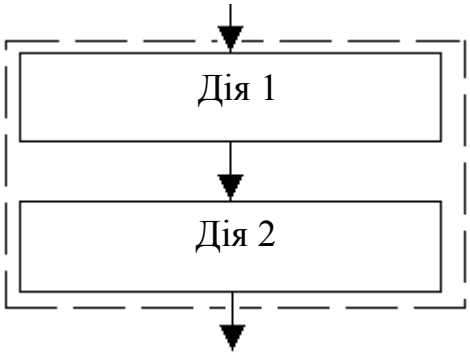


Рисунок 20 – Алгоритмічне зображення лінійного ланцюжка операторів

Другий структурний оператор називається **умовний оператор (оператор розгалуження)**. Досить часто одне або інше завдання повинні виконуватися залежно від певної умови, яка залежить від результатів виконання попередньої програми або від зовнішніх пристроїв. Кожне з таких завдань називається плечем умовного оператора й зображене на рисунку 21.

Умовний оператор може використовуватися в неповному виді, коли відсутнє одне із плечей оператора.

Третій структурний оператор – це **оператор циклу з перевіркою умови після тіла циклу**. Такий оператор легко реалізується мовою

програмування асемблер за допомогою команди умовного або безумовного переходу. Відмінність від умовного оператора полягає в тому, що передача керування здійснюється не вперед, а назад. На мовах програмування високого рівня такий оператор входить до складу мови (оператор do..while у мові програмування C++, або оператор repeat..until у мові програмування PASCAL). У мові асемблер MCS-51 – це команда DJNZ, перед якою пишеться тіло циклу.

Алгоритмічно такий оператор зображений на рисунку 22.

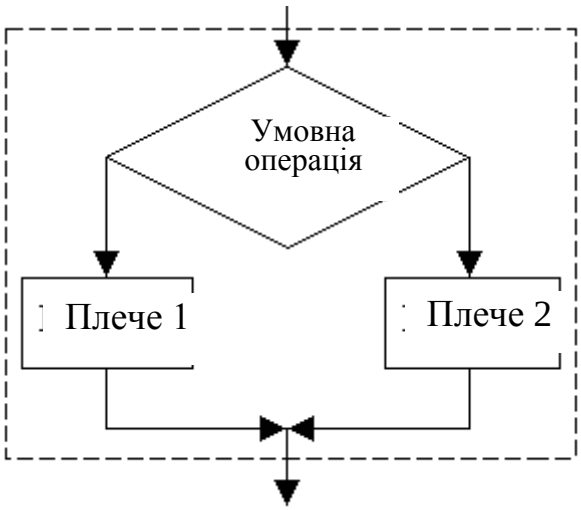


Рисунок 21 – Алгоритмічне зображення умовного оператора

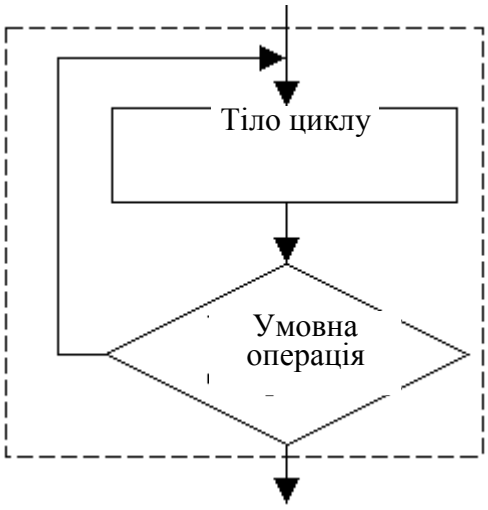


Рисунок 22 – Алгоритмічне зображення оператора циклу з перевіркою умови після тіла циклу

Четвертий структурний оператор – це **оператор циклу з перевіркою умови до тіла циклу**. На відміну від попереднього оператора тіло циклу в цьому операторові може жодного разу не виконатися, якщо умова циклу відразу ж виконана. Цей оператор, як і умовний оператор, неможливо реалізувати на одній машинній команді (рис. 23).

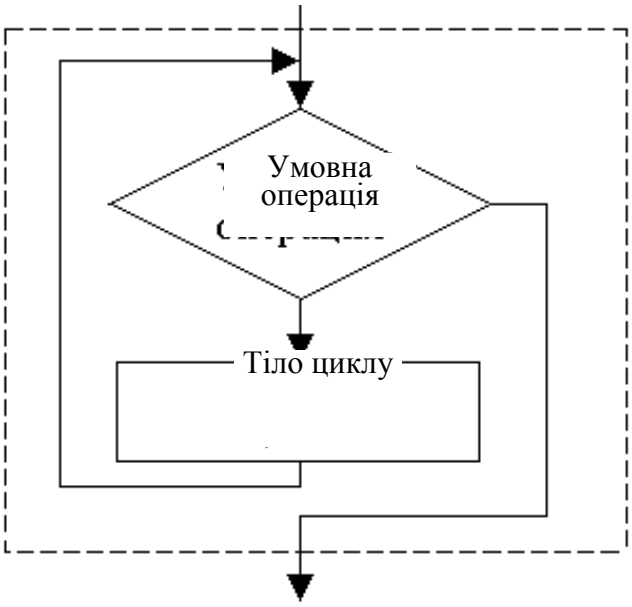


Рисунок 23 – Алгоритмічне зображення оператора циклу з перевіркою умови до тіла циклу

Умовний оператор виконується шляхом порівняння величин. Якщо це бітова величина, то в такому випадку все просто – якщо біт установлений, виконується одне плече, якщо ні – інше. При порівнянні байтових і більш розміром величин зазначають наступні умови:

E (equal) – рівно;

NE (Not Equal) – нерівно;

GE (Great or Equal) – більше або рівно;

LE (Less or Equal) – менше або рівно;

GT (Great Than) – більше;

LT (Less Than) – менше.

У деяких мікропроцесорів операції даних умовних переходів присутні. Для мікропроцесорів, у яких немає таких операцій (у тому числі й I8051), дані умови будуються на аналізі результатів регістру PSW після виконання порівняння чисел.

Розглянемо приклад порівняння двох байтових величин на мікроконтролері I8051. Так як даний контролер має команду порівняння й переходу, якщо нерівно (CJNE), то даний варіант ми розглядати не будемо.

Нехай є 2 байта, один в акумуляторі, іншийї – у регістрі R1. Потрібно їх зрівняти й виконати плече умови №1, якщо вони рівні. Інакше – плече №2:

```
      Clr C             ; Відчистимо ознаку переносу перед порівнянням.
      Subb A,R1         ; Порівняння вирахуванням
      Jnz Neravno       ; Перехід на плече №2.
      ...               ; Тут виконується плече №1 рівності.
      Jmp Dalee
Neravno:
      ...               ; Тут виконується плече №2 нерівне.
Dalee:                  ; Продовження програми.
```

При виконанні м'яких умов (GE і LE) завдання ускладнюється через контроль двох ознак. Нехай нам необхідно перевірити умову (A)≤(R1). Якщо воно дотримується, то виконуємо плече №1, інакше – №2:

```
      Clr C             ; Відчистимо ознаку переносу перед порівнянням.
      Subb A,R1         ; Порівняння вирахуванням.
      Jb ACC.7,Menwe    ; Перехід на плече №1. за умовою менше (A<0).
      Jz Menwe          ; Перехід на плече №1 за умовою рівності (A=0).
      ...               ; Тут виконується плече №2.
      Jmp Dalee
Menwe:
      ...               ; Тут виконується плече №1.
Dalee:                  ; Продовження програми.
```

За таким принципом можна побудувати конструкції й за допомогою команди додавання. Приклад контролю ознак результату порівняння залежно від різних умов при виконанні вирахування наведено в таблиці 2.

Таблиця 2 – Розгалуження і їх команди після операції вирахування

Умова	Назва умови	Виконання розгалужень
A=R1	E	JZ
A≠R1	NE	JNZ
A<R1	LT	JC або JB ACC.7
A>R1	GT	JNC або JNB ACC.7
A≤R1	LE	(JC+JZ)

A≥R1	GE	(JNC+JZ)

У процесі побудови програми слід пам'ятати, що умовні переходи в мові асемблер для мікроконтролера I8051 можуть виконати тільки відносний перехід адресним простором у межах -128…+127 адрес щодо самої команди. Тому бажано розташовувати плечі умов, які виконуються переходом, якнайближче до ділянки програми, де виконується порівняння. Якщо такої можливості немає і у процесі компіляції програми, компілятор видає помилку про неможливість виконати перехід, то наведені вище конструкції умовних операторів потрібно доповнити «довгими стрибками» LJMP до плеча оператора.

3.2 Створення програми мікропроцесорного пристрою

Розглянемо приклад написання програми для мікроконтролера. Насамперед, не потрібно забувати, що програма не може існувати незалежно від схеми пристрою. Якщо при написанні програми для універсального комп'ютера, такого як IBM PC можна не замислюватися про схему, тому що вона стандартна, то перед написанням програми для мікроконтролера необхідно розробити схему пристрою, або вивчити, якщо остання є в наявності, до складу якого буде входити мікроконтролер.

Для програмування будемо застосовувати мікропроцесорну систему, виконану на основі навчально-відладжувального стенду EV8031. Перше завдання, яке потрібно виконати – визначення устаткування системи, способи адресації до пристроїв і розподіл адресного простору.

Розподіл адресного простору зовнішньої пам'яті в стенді виконує дешифратор адреси DD7 (рис. 24). Своїми входами він підключений до старшої тетради 16-розрядної зовнішньої адреси. На вході керування E3 буде логічна «1», що дозволяє роботу дешифратора, якщо адресований пристрій має адресу більше 7FFFh. Т. ч. дешифратор розподіляє адресний простір старшої половини зовнішніх адрес мікроконтролера. Залежно від комбінації рівнів на входах 0, 1 і 2 дешифратора, підключених до шини адреси, з'являється низький рівень тільки на одному виході, номер якого у двійковій формі заданий на входах. Сигнали з виходів дешифратора називаються $\overline{CS}$ (Chip Select) і мають свій номер. Кожний сигнал $\overline{CS}$

відповідає тільки своєму адресному простору. Низький рівень цього сигналу вказує на адресацію до пристрою, який підключається до шин мікропроцесора даним сигналом. Адресні простори, відповідні до сигналів $\overline{CS}$ зазначені на рисунку 24.

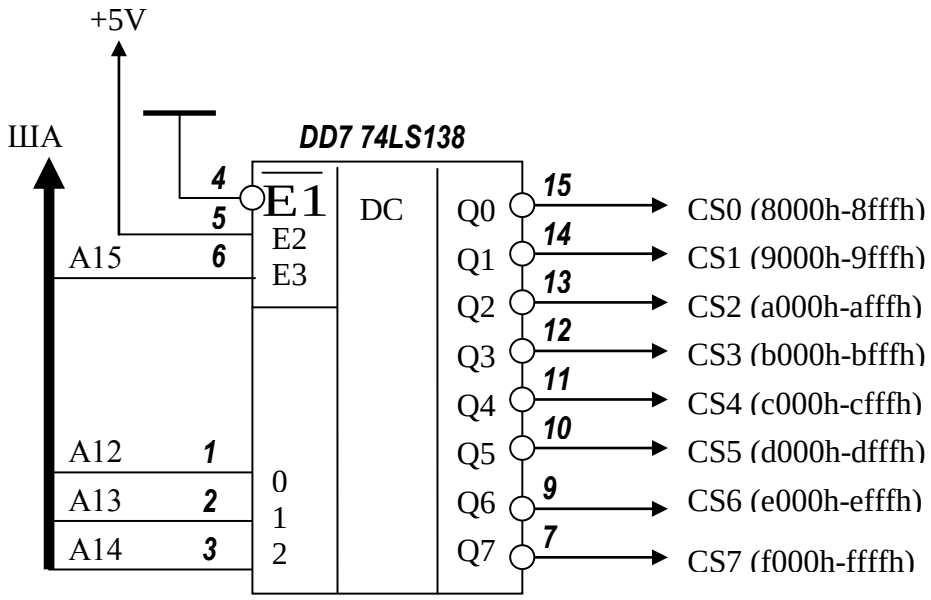


Рисунок 24 – Дешифратор адресного простору зовнішньої пам'яті стенда

Аналізуючи принципову схему стенда, можна побачити, що до сигналу $\overline{CS0}$ підключена мікросхема паралельного периферійного адаптера КР580ВВ55; до $\overline{CS1}$ – буфер читання клавіатури; $\overline{CS2}$ – старша пара статичних індикаторів; $\overline{CS3}$ – молодша пара статичних індикаторів.

Для простоти схеми дешифрації адреси (у цьому випадку вона побудована тільки на мікросхемі DD7), дешифратор вибирає з адресної шини деякий проміжок адрес, у якому можна помістити безліч пристроїв. Фізично в такому діапазоні перебуває тільки одна (у деяких випадках – декілька) комірка. Тому для адрес звернення при програмуванні ухвалюють тільки першу (декілька перших) комірку. Наприклад, для запису числа в регістр-засувку статичного індикатора можна звернутися на адресу зовнішньої пам'яті 0A000h, 0A001h і т.д. до 0Afffh. Тоді при програмуванні ухвалюють тільки адресу 0A000h.

Розглянемо кілька прикладів. Нехай нам необхідно засвітити число на статичних індикаторах HG3 і HG4. Для цього нам необхідно завантажити двійково-десяткове число в акумулятор і записати ці дані в комірку зовнішньої пам'яті даних за адресою 0B000h згідно з аналізом принципової схеми. Розташування статичних індикаторів на стенді і їх нумерація зазначена на рисунку 25.

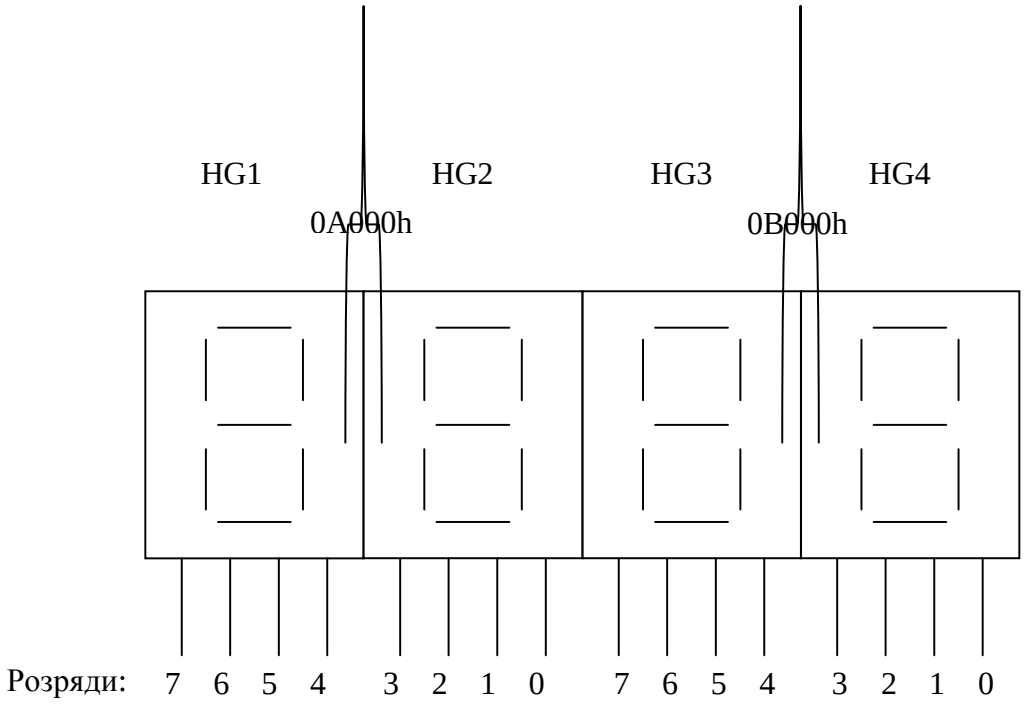


Рисунок 25 – Статичні індикатори

Приклад ділянки програми:

```
Mov A,#56              ; Число, яке виводимо на індикатор.
Mov DPTR,#0b000h       ; Адреса регістру індикатора.
Movx @DPTR,A           ; Вивід на індикатор.
```

Дані, виведені на ці індикатори, повинні бути подані у двійково-десятковій формі (тетради мають значення 0-9). При виводі шістнадцяткового числа, код тетради якої більше 9 (Ah-Fh), дешифратор індикатора не зможе перетворити це число на правильну комбінацію сигналів для відображення цифри й сегменти індикатора будуть погашені.

На основі попереднього прикладу можна змінити умови. Нехай нам тепер необхідно вивести теж число, тільки індикатор HG3 повинен не світитися. Для виконання цього завдання необхідно, щоб старша тетрада числа дорівнювала Fh. Цього можна добитися шляхом виконання операції

«логічне АБО» над числом і константою F0h. Такий спосіб у програмуванні називається **маскуванням**:

```
Mov A,#56          ; Число, яке виводимо на індикатор.
Orl A,#0F0h        ; Налагоджуємо маску – гасимо старшу тетраду.
Mov DPTR,#0b000h   ; Адреса регістру індикатора.
Movx @DPTR,A       ; Вивід на індикатор.
```

За допомогою операції «логічне І» можна виконувати скидання бітів числа або тетради, а операцією «що виключає АБО» – інверсію бітів.

Тепер наведемо приклад написання програми секундоміра. Індикатори HG1 і HG2 відображають секунди, HG3 і HG4 – десяті й соті секунди. Кнопки клавіатури виконують керування: «1» – старт; «2» – стоп; «3» – скидання показань.

Секундомір обов'язково повинен містити пристрій виміру часу, який у свою чергу завжди складається з генератора еталонних інтервалів часу й лічильника цих інтервалів. Схема пристрою виміру часу наведена на рисунку 26.

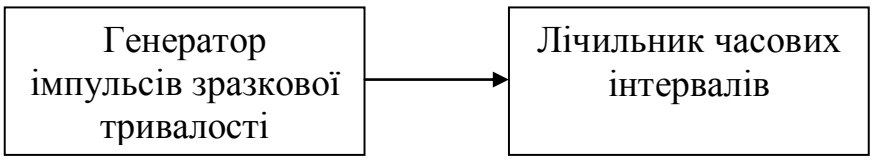


Рисунок 26 – Структурна схема пристрою виміру часу

Генератор імпульсів зразкової тривалості повинен виробляти імпульси із частотою 100 Гц, що відповідає дискретності часу 0,01 сек. Частота тактового генератора стенда становить 7 372 800 Гц. Усередині мікроконтролера є вбудований дільник імпульсів на 12, який тактує рахунковий вхід таймерів. Для визначення константи, що завантажується в таймер для формування рахункового імпульсу можна скористатися формулою

$$K = 65535 - \frac{F_{OSC}}{12 \cdot f},$$

де F_{OSC} – частота кварцового генератора;

F_T – необхідна частота після розподілу таймером.

Підставивши числові значення, одержимо:

Враховуючи це, структурна схема пристрою набуде виду, зазначеного на рисунку 27. Усі елементи даного пристрою, крім дешифратора й індикаторів, необхідно виконати програмно. Для такого виконання розіб'ємо пристрій на складові частини. Генератор зразкових імпульсів (ГЗІ) являє собою таймер мікроконтролера, що генерує імпульси із частотою 100 Гц. Ці імпульси ми будемо подавати на програмний лічильник з корекцією показань секунд (після 59 секунди набуває значення 0). ГЗІ й схему сканування клавіатури виконаємо в одному потоці – підпрограмі переривання від переповнення T/C0. Це дозволить позбутися алгоритму усунення шурудіння контактів. Лічильник і схему керування – в іншому потоці: це буде основна програма.

Для такої організації програми необхідні комірки пам'яті для надання необхідної інформації між потоками (синхронізація потоків). Така комірка в нас буде називатися STATUS. Вона буде являти собою набір бітів, що встановлюються у певних ситуаціях та скидаються програмою. Необхідні наступні біти:

Count – Біт, який встановлює п/п обробки переривань переповнення таймера, для відліку сотих часток секунди; Скидає основна програма після виконання інкременту лічильника;

Keyrun – виявлене натискання кнопки «1». П/п переривання повідомляє основну програму про натискання кнопки.

Keystop – виявлене натискання кнопки «2». П/п переривання повідомляє основну програму про натискання кнопки.

Keyreset – виявлене натискання кнопки «3». П/п переривання повідомляє основну програму про натискання кнопки.

Runprocessed – прапор обробки кнопки «1». Необхідний для однократного виконання події натискання при втриманні натиснутої кнопки. Скидається, коли Keyrun=0 (кнопка віджалася).

Stopprocessed – прапор обробки кнопки «2». Скидається, коли Keystop=0.

Resetprocessed – прапор обробки кнопки «3». Скидається, коли Keyreset=0.

Run – прапор, що дозволяє рахунок часу.

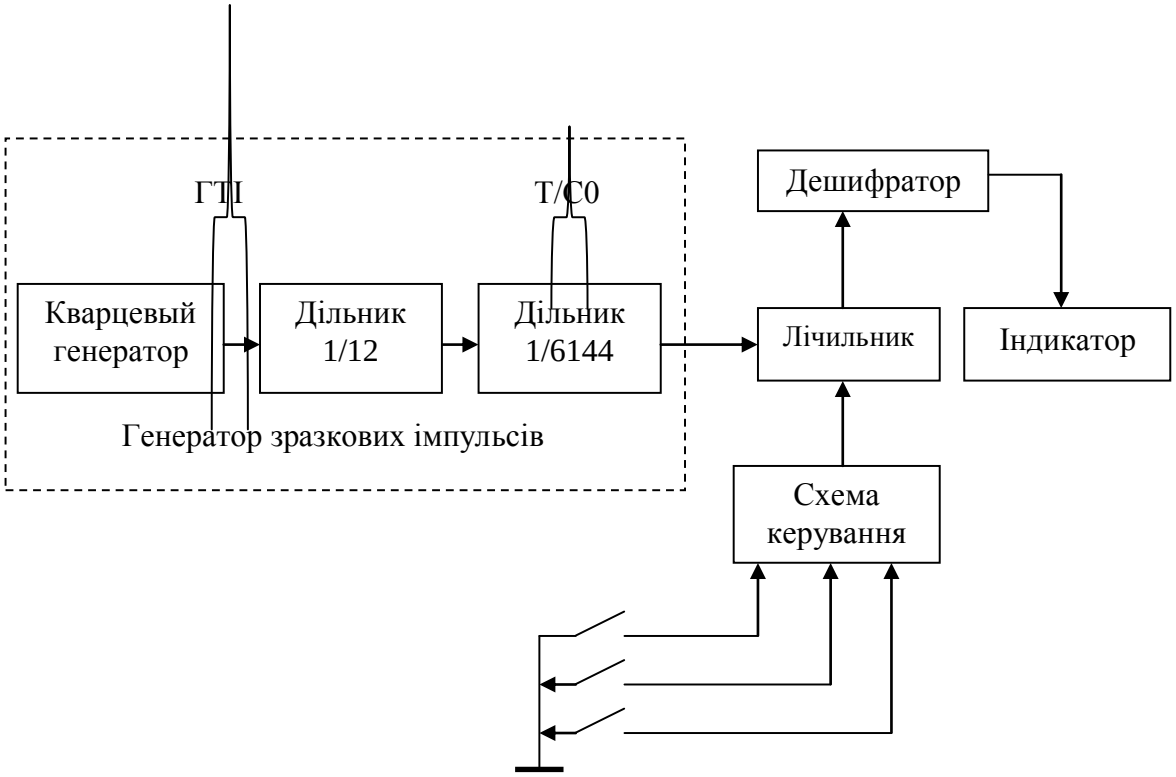


Рисунок 27 – Уточнена структурна схема пристрою виміру часу

Крім регістру STATUS нам необхідно визначити, де будуть зберігатися числа, виведені на індикатор. Приймемо регістр R7 для рахунку часток секунд, R6 – для рахунку секунд. Уміст цих регістрів будемо виводити на індикатори. Для подальшого написання програми нам потрібний алгоритм.

Алгоритм підпрограми обслуговування переривання наводити не будемо. Він будується на аналогічних принципах. Ця підпрограма повинна виконати перезавантаження лічильника Т/С0, установити прапор Count, просканувати клавіатуру й, якщо натиснута яка-небудь клавіша, установити відповідний біт у комірці STATUS; якщо клавіша не натиснута – скинути відповідний біт. За допомогою цієї комірки виконується своєрідний «місток» для обміну інформацією між потоком у

перериванні й основною програмою (рис. 29). Іноді такі комірки називають **регістрами подій**, а біти – **прапорами подій.**

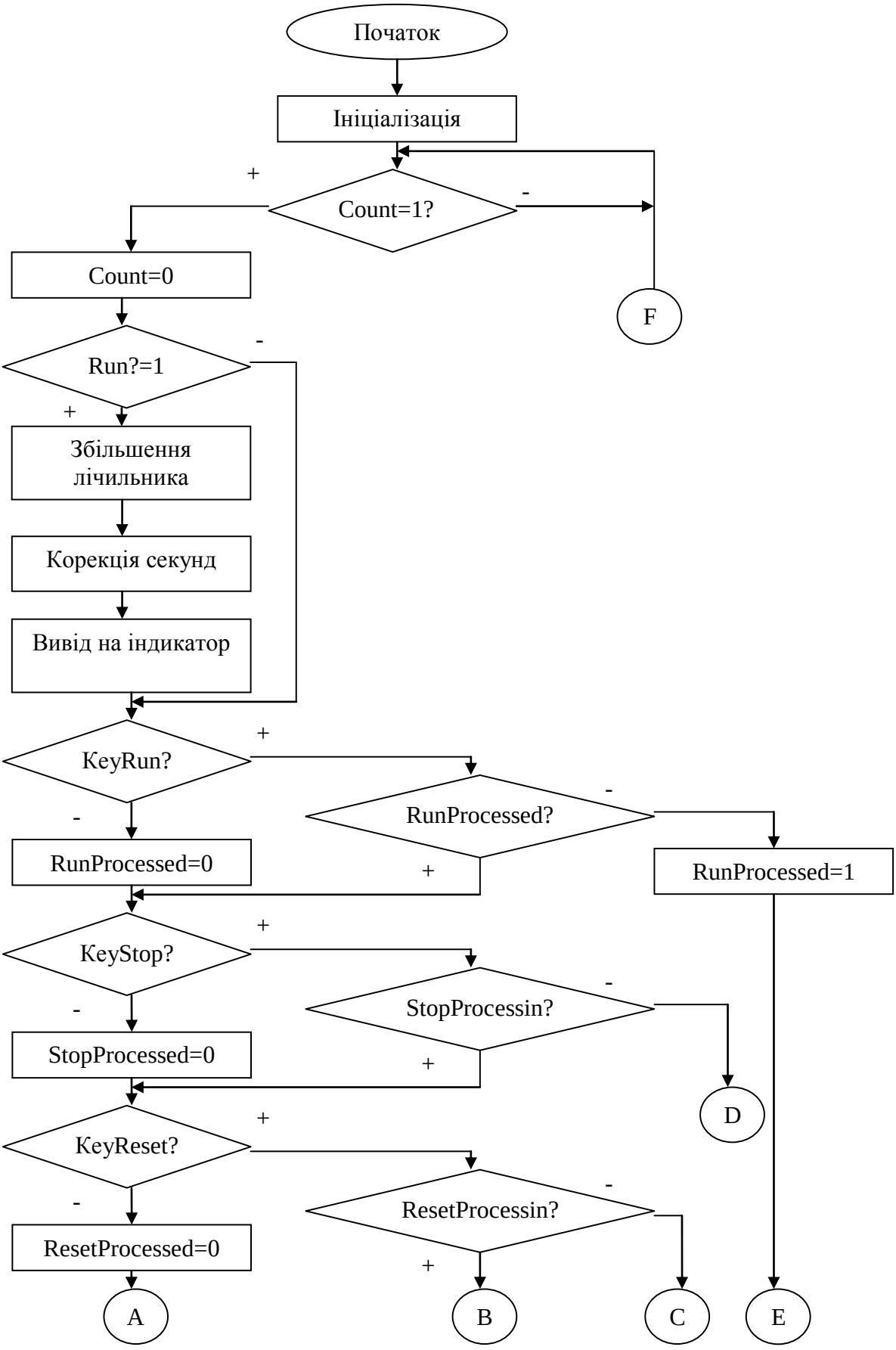


Рисунок 28 – Блок-схема алгоритму основної програми секундоміра

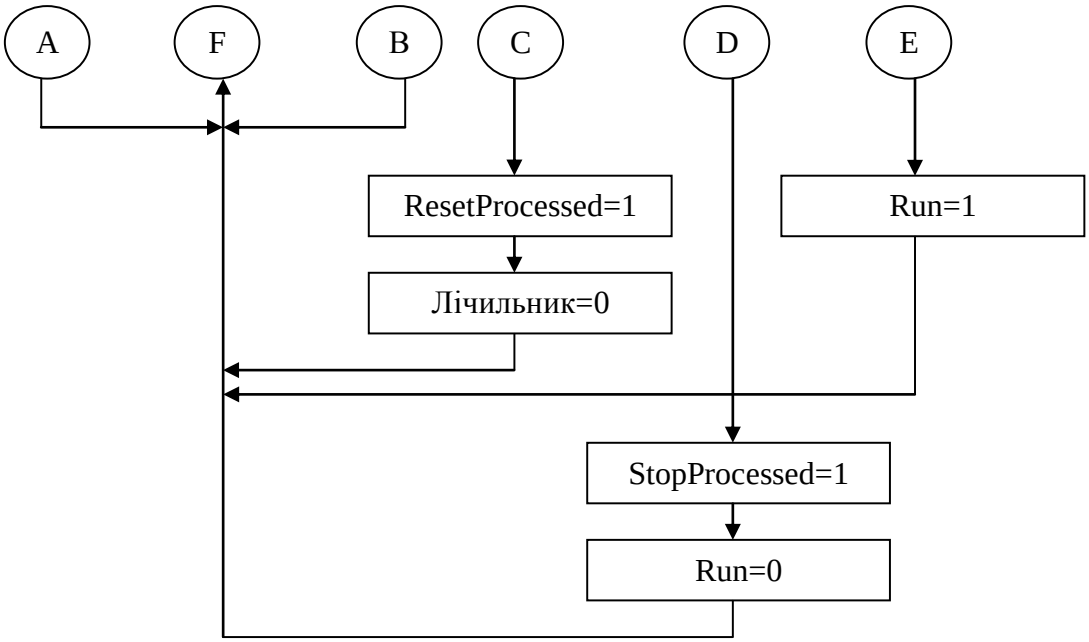


Рисунок 28, аркуш 2

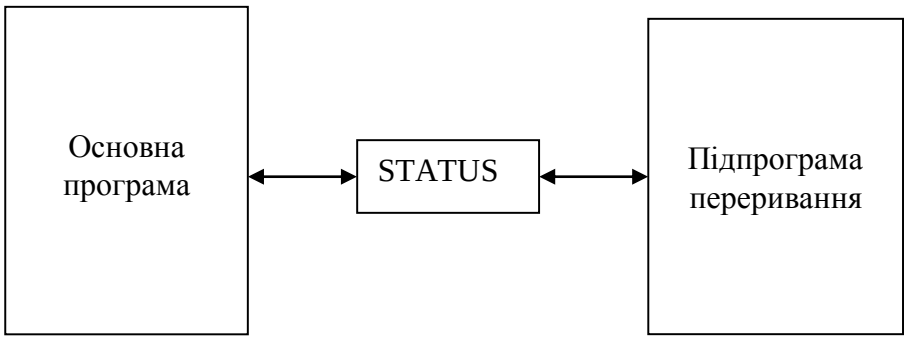


Рисунок 29 – Зв'язок між потоком основної програми й потоком переривання

При цьому в кожного потоку можуть бути свої незалежні змінні. У нашому випадку це регістри лічильника R6 і R7 для основної програми й регістри таймера TH0 і TL0 для потоку переривання.

Пишемо програму секундоміра мовою асемблер MCS-51:

```
$nolist
$include(mod51)
$list
;Програма реалізації секундоміра на стенді EV8031
;-------------------------------------------------------------------------
-
```

```
;Визначення змінних
Tmconst           set   6144  ;Загальний коефіцієнт розподілу тактової
                              ;частоти Fosc/(12*Ft)
                              ;Fosc=7372800 Гц, Ft=100 Гц
L_IND             equ   0a000h;Адреси З_Інд. у зовнішній пам'яті
R_IND             equ   0b000h
Low_Counter       equ   R7    ;Призначення імен регістрам
High_Counter      equ   R6
But1              equ   9006h ;Адреса звертання до кнопки 1
But2              equ   But1-1;Адреса звертання до кн.2=Адр.Кн1-1
But3              equ   But1-3;Адреса звертання до кн.3
Downstek          EQU   127   ;Визначаємо дно стеку
;Бітова адресація до прапорів у комірці РПД 20h STATUS:
Count             bit   0     ;Прапор рахунку
Keyrun            bit   1     ;Прапор натискання кнопки запуск
Keystop           bit   2     ;Прапор натискання кнопки останов
Keyreset          bit   3     ;Прапор натискання кнопки скидання
Runprocessed      bit   4     ;Прапор обробки події запуску
Stopprocessed     bit   5     ;Прапор обробки події останову
Resetprocessed    bit   6     ;Прапор обробки події скидання
Run               bit   7     ;Прапор дозволу рахунку часу
;-------------------------------------------------------------------------
-
;Місце входу в програму  вектором скидання процесора
      Org 0
      jmp Main
;-------------------------------------------------------------------------
-
;Місце входу в переривання процесора по переповненню T/C0
      Org 0bh
TMR0_Int:                   ;Вхід в п/п переривання T/C0
      Push PSW              ;Збереження стану програми
      Push ACC
      Push DPL
      Push DPH
      CLR TR0               ;Зупиняємо таймер
      Mov TL0,#LOW(NOT(Tmconst+1))  ;завантажуємо константи
      Mov TH0,#HIGH(NOT(Tmconst+1))
      SETB TR0              ;Запускаємо таймер
      SETB Count            ; Повідомляємо, що закінчилася 1/100 секунди
      Mov DPTR,#but1        ;Адреса звертання до буфера клавіатури
      Movx A,@DPTR
      Jb ACC.0, Norun       ; Кн.1 натиснута?
      SETB Keyrun           ; Так, установимо прапор
      sjmp KEY2
Norun:CLR Keyrun            ; Ні, скинемо прапор
KEY2: Mov DPTR,#but2        ;Адреса звертання до буфера клавіатури
      Movx A,@DPTR
```

```
      Jb ACC.0, Nostop        ; Кн.2 натиснута?
      SETB Keystop            ; Так, установимо прапор
      sjmp KEY3
Nostop:CLR Keystop            ; Ні, скинемо прапор
KEY3: Mov DPTR,#but3          ;Адреса звертання до буфера клавіатури
      Movx A,@DPTR
      Jb ACC.0, Noreset             ; Кн.3 натиснута?
      SETB Keyreset                 ; Так, установимо прапор
      sjmp Donekey
Noreset:CLR Keyreset                ; Ні, скинемо прапор
Donekey:Pop DPH                     ;Повернення стану програми
      Pop DPL
      Pop ACC
      Pop PSW
      RETI
;-------------------------------------------------------------------------
-
;Основна програма
;Тут починається ініціалізація
Main: Mov SP,#downstek-28           ;Настроювання покажчика стека
      Mov TMOD,#00000001b           ;T/C0, режим 1 (16-розрядів)
      Mov TL0,#LOW(NOT(Tmconst+1))  ;завантажуємо константи
      Mov TH0,#HIGH(NOT(Tmconst+1))
      SETB IT0                      ;Дозвіл переривань від T/C0
      SETB TR0                      ;Запуск таймера
      SETB EA                       ;Дозвіл переривань
      CLR A
      Mov 20h,A                     ;Відчищання керуючих бітів STATUS
Clear:Mov High_Counter,A            ;Відчищання регістрів лічильника
      Mov Low_Counter,A
      Mov DPTR, #L_IND              ;Вивід на індикатори 0
      Movx @DPTR,A
      Mov DPTR,#R_IND
      Movx @DPTR,A
;Кінець ініціалізації. Початок основної програми
Wait: Jnb Count,$                   ;Очікування 1/100 сек.
      CLR Count
      Jnb Run, Keyprocessed         ; Рахунок дозволений?
;Тут починається лічильник часток і секунд
      Inc Low_Counter               ; Так, збільшуємо соті сек.
      Mov A,Low_Counter
      DA A                          ;Двійково-десяткова корекція
      Mov Low_Counter,A
      JNC Nocarry                   ; Число більше 99?
      Mov A, High_Counter           ; Так, збільшимо лічильник секунд
      INC A
      DA A                          ;Двійково-десяткова корекція
      CJNE A,#60h,Less60            ;Секунд 60?
```

```
      CLR A                          ; Так, скинемо старший розряд лічильника
Less60:Mov High_Counter,A            ; Ні, збережемо результати
Nocarry:Mov A,High_Counter
      Mov DPTR, #L_IND               ;Вивід на індикатори секунд
      Movx @DPTR,A
      Mov A, Low_Counter             ;і часток секунд
      Mov DPTR,#R_IND
      Movx @DPTR,A
;Обробка натискань кнопок
Keyprocessed:
Isrun?:Jb Keyrun, Setrun             ;Натиснута кнопка запуску?
      CLR Runprocessed               ; Ні, відчистити прапор обробки
Isstop?:Jb Keystop, Setstop          ;Натиснута кнопка стоп?
      CLR Stopprocessed              ; Ні, відчистити прапор обробки
Isreset?:Jb Keyreset,Setreset        ;Натиснута кнопка скидання?
      CLR Resetprocessed             ; Ні, відчистити прапор обробки
      Jmp Wait                       ;Кнопки не натиснуті. Очікуємо 1/100
сек.

Setrun:Jb Runprocessed,Isstop?       ;Кнопка була відпущена?
      SETB Run                       ; Так, розв'язний рахунок.
      SETB Runprocessed              ;і встановимо прапор обробки кнопки
      Jmp Wait

Setstop:Jb Stopprocessed,Isreset?    ;Кнопка була відпущена?
      CLR Run                        ; Так, заборонимо рахунок.
      SETB Stopprocessed             ;і встановимо прапор обробки кнопки
      Jmp Wait

Setreset:Jb Resetprocessed, Wait     ;Кнопка була відпущена?
      CLR A                          ; Так, відчистимо акумулятор
      SETB Resetprocessed            ;і встановимо прапор обробки кнопки
      Jmp Clear                      ;Стрибок на відчищання лічильника
;Кінець програми
;-------------------------------------------------------------------------
-
END
```

Уводимо вихідний текст програми в редакторі текстових файлів «Блокнот» (або будь-якому іншому) і зберігаємо файл із розширенням *.asm

Для компіляції програми в командному рядку операційної системи вводимо:

Asm51.exe [шлях до файлу]Secmeter.asm

Результатом компіляції є файли, що мають назву вихідної програми й розширення *.HEX і *.LST. Перший з них – отриманий код програми у

форматі Intel Hex, а другий – лістинг компіляції, у якому компілятор вказує на помилки (якщо вони є) й іншу службову інформацію. Якщо ці файли не створені, то процес компіляції не міг початися через помилку. Найбільш частими причинами таких помилок бувають:

– Відсутні за зазначеним шляхом файли, оголошені директивою $include.

– У шляху до вхідного файла програми або її модулів є пробіли, символи кирилиці або інші елементи, що не підходять під формат файлів DOS8.3.

Якщо програма пишеться об'єктним кодом асемблера (використовуються показчики сегментів коду, директиви резервування змінних), то процес компіляції виконується у кілька етапів. На першому етапі за допомогою компілятора одержують об'єктний код кожного модуля програми з розширенням *.obj. Далі об'єктний код модулів за допомогою лінкера з'єднується в один файл, з якого одержують *.hex файл.

3.3 Оптимізація коду програми й стиль програмування

Критерії оптимізації програм з погляду економії ресурсів повинні вибиратися залежно від конкретних обставин. Завданням програміста є економія таких ресурсів, як час роботи програми й використовувані обсяги ОЗП й ПЗП. На відміну від великих проектів програмування для мікроконтролерів здійснюється, як правило, або одним програмістом, або невеликою групою програмістів. Гарний стиль програмування завжди корисний. Він важливий для економії часу, затрачуваного на розробку програми, її налагодження й підтримку, необхідну в процесі вдосконалення виробу за результатами його експлуатації або у випадку введення доробок в апаратуру.

Стиль програмування в значній мірі визначає успіхи програміста й відбиває його індивідуальність. Неохайність у програмуванні теж можна вважати стилем, але це поганий стиль. Для поліпшення стилю програмістові час від часу необхідно осмислювати пророблену роботу не тільки з погляду досягнутих результатів, але й з погляду витрачених на це засобів. Засвоєння гарного стилю програмування повинне йти паралельно з навчанням програмуванню на асемблері. Стиль характеризує творчу сторону роботи програміста, тому жорстоких правил тут немає. Однак

кожному програмістові дозволено застосовувати ті або інші поради по-своєму або не застосовувати їх.

4 ПРАКТИКУМ З АСЕМБЛЕРА MCS-51

4.1 Вимоги до звітів про практикум

У процесі виконання практичних завдань кожний студент повинен уважно вивчити завдання свого варіанта, виконати аналіз поставленого завдання й знайти способи рішення. У випадку виникнення непорозумінь у завданні, студент може проконсультуватися у викладача.

Кожний звіт повинен містити:

– Тему практичного завдання.

– Мету практичного завдання.

– Опис отриманого завдання.

– Блок-схему алгоритму вирішення завдання.

– Опис змінних, використовуваних у програмі, опис призначення регістрів, що зберігають дані. Опис регістрів, які використовуються тимчасово для яких-небудь цілей, допускається не вказувати. Приклад опису змінних:

R6 – Додаток №1;

R7 – Додаток №2;

R5 – Зберігає код одиниць числа для відображення на семисегментному індикаторі;

Keypress – прапор натискання клавіш.

– Вихідний текст програми мовою асемблер.

– Виводи з проробленої роботи.

При написанні підпрограм у програмі в поле коментарів перед самою підпрограмою необхідно вказувати вхідні й вихідні дані:

```
; Decode-Підпрограма перекладу числа в код семисегментного індикатора
;Вхід:  A – ДД число формату 0X
;Вихід: A – семисегментний код числа
;Змінює: DPTR
Decode:                          ;Тут починається підпрограма
```

Практичні завдання побудовані для виконання на навчально-відлагоджувальних стендах EV8031/AVR фірми Open Systems.

4.2 Практичне завдання 1

Тема. Система команд ОЕОМ MCS-51. Повноекраний наладник FD51.

Мета: знайомство з асемблером сімейства мікроконтролерів MCS-51. Одержання навичок у програмуванні й налагодженні програм для мікроконтролерів MCS-51.

Таблиця 3 – Варіанти завдань до самостійної роботи

Варіант	**Завдання**
1	*2*
1	Розробити програму підрахунку позитивних чисел у масиві
2	Розробити програму підрахунку негативних чисел у масиві
3	Розробити програму підрахунку кількості нульових елементів у масиві
4	Розробити програму підрахунку кількості елементів масиву, відмінних від нуля
5	Розробити програму підрахунку парних чисел у масиві
6	Розробити програму підрахунку непарних чисел у масиві
7	Розробити програму підрахунку нульових бітів у байтах масиву
8	Розробити програму підрахунку одиничних бітів у байтах масиву
9	Розробити програму пошуку кількості елементів у масиві, рівних першому елементу
10	Розробити програму сортування за зростанням масиву
11	Розробити програму сортування за убуванням масиву
12	Розробити програму заповнення масиву числами в арифметичній прогресії
13	Розробити програму заповнення масиву нулями
14	Розробити програму, що реалізує суму всіх елементів масиву
15	Розробити програму, що реалізує різницю всіх елементів

	масиву
16	Розробити програму пошуку найбільшого числа в масиві
17	Розробити програму пошуку найменшого числа в масиві

Продовження таблиці 3

1	*2*
18	Розробити програму обчислення середнього арифметичного масиву (кількість елементів прийняти рівною 8)
19	Розробити програму додавання елементів 2-х масивів. Результат затягти у 1-й масив
20	Розробити програму переміщення блоку пам'яті на 16 байт уперед
21	Розробити програму порівняння двох масивів. Виділити змінну для результату й дорівняти її до 1, якщо масиви однакові, інакше вона дорівнюватимеме 0
22	Розробити програму логічного зрушення всіх бітів у масиві вліво
23	Розробити програму логічного зрушення всіх бітів у масиві вправо
24	Розробити програму обчислення суми окремо позитивних і негативних елементів масиву
25	Розробити програму обчислення суми окремо парних і непарних елементів масиву
26	Розробити програму перетворення числа із двійкової форми на двійково-десяткову
27	Розробити програму заповнення елементів масиву за зростанням
28	Розробити програму заповнення елементів масиву за убуванням
29	Розробити програму знаходження суми найбільшого й найменшого елементів масиву
30	Розробити програму знаходження різниці найбільшого й найменшого елементів масиву

Примітка: якщо в завданні варіанта не зазначена кількість елементів

масиву, то рекомендується прийняти 10.

4.3 Практичне завдання 2

Тема. Вивчення стенда й команд ОЕОМ КР1816ВЕ31.

Мета: вивчення функціональних можливостей навчально-відлагоджувального стенда, внутрішньої структури й системи команд ОЕОМ КР1816ВЕ31.

Таблиця 4 – Варіанти завдань до самостійної роботи

Варіант	**Завдання**
1	*2*
1	Занести до регістру R4 ДД число 0X, в R6 – X0. Суму чисел відобразити на С_Інд. HG1, HG2 з попереміним миготінням тетрад із частотою 1 Гц.
2	Занести до регістру R4 число XXh. Виконати двійково-десяткове перетворення числа й вивести результат на С_Інд. (HG2, HG3, HG4 – сотні, десятки й одиниці відповідно)
3	Занести до регістру В ДД число XX, в R2 – 0X. Результат різниці чисел B-R2 з двійково-десятковою корекцією результату вивести на С_Інд. HG1, HG2 з миготінням 2 Гц
4	Занести до А ДД число XX, до регістру R5 – X0. Число з А відобразити на С_Інд. HG1, HG2, число з R5 – поперемінно відображати на С_Інд. HG3, HG4 із частотою 0,5 Гц
5	Занести до регістру R2 ДД число 0X, в R5 X0. Суму чисел поперемінно відображати на С_Інд. HG1, HG2 і HG3, HG4 із частотою 1 Гц
6	Занести до комірки внутрішньої пам'яті з адресою 70h ДД число 0X, до регістру R3 – X0. Суму чисел поперемінно відображати на С_Інд. HG2, HG3 і HG1 HG4 із частотою 2 Гц
7	Занести до регістру R0 число 0; додавати до регістру одиницю

	із частотою 4 Гц до 99, виконувати двійково-десяткову корекцію й виводити результат на С_Інд. HG1, HG2
8	Занести до регістру В ДД число Х0, до регістру R1 – ХХ. Число з В відображати на С_Інд. HG1 із частотою 1 Гц, число з R1 відображати на С_Інд. HG3, HG4 із частотою 0,5 Гц

Продовження таблиці 4

1	*2*
9	Прочитати значення регістру PSW, скинути старші біти тетрад прочитаного значення й відображати його на С_Інд. У наступному порядку: HG1-HG2→HG2-HG3→HG3-HG4 із часом зміни індикаторів 0,5 сек.
10	Занести до регістру R4 ДД число 0Х, до регістру – R1 Х0. Суму чисел відображати на С_Інд. HG1, HG2 із плавним вгасанням числа протягом 5 сек.
11	Занести до А ДД число Х0, до регістру В – 0Х. Молодшу тетраду суми постійно відображати на С_Інд. HG4, а старшу – поперемінно на індикаторах HG1, HG2 і HG3 з інтервалом в 0,5 сек.
12	Занести до регістру В ДД число 0Х, до регістру – R5 – Х0. Старшу тетраду суми постійно відображати на С_Інд. HG1, а молодшу – поперемінно на індикаторах HG4, HG3 і HG2 з інтервалом в 1 сек.
13	Занести до регістру R1 ДД число 0Х. Віднімаючи від числа одиницю, відображати результат на С_Інд. HG2 до нуля із частотою 1 Гц. Після цього згасити індикатор на 2 сек. і виконувати програму знову
14	Занести до регістру R3 ДД число ХХ, до регістру R5 – ХХ. Поперемінно відображати ці числа на С_Інд. HG4,HG3 і HG2,HG1 із частотою 2 Гц. Після 10 сек. згасити індикатори на 2сек., зациклити програму
15	Занести до регістру А ДД число 0Х, до регістру R2 – Х0. Число з А поперемінно відображати на С_Інд. HG2, HG1 із частотою 1 Гц, з R2 – на С_Інд. HG3, HG4 із частотою 2 Гц

16	Занести до регістру R4 ДД число X0, до R6 – 0X. Різницю чисел із двійково-десятковою корекцією відобразити на С_Інд. HG1, HG2 з попереміним миготінням тетрад із частотою 1 Гц
17	Занести до регістру R1 ДД число 0X, до регістру – R5 X0. Десятки суми послідовно відображати на всіх С_Інд. з затримкою часу 0,25 сек.

Продовження таблиці 4

1	*2*
18	Занести до регістру А ДД число 0X, до R5 X0. Суму чисел поперемінно відображати на С_Інд. HG1, HG3 і HG2, HG4 із частотою 2 Гц
19	Занести до комірки внутрішньої пам'яті з адресою 20h ДД число X0, до регістру R3 – 0X. Різницю чисел із двійково-десятковою корекцією поперемінно відображати на С_Інд. HG2, HG3 і HG1 HG4 із частотою 2 Гц
20	Занести до регістру R0 ДД число 10. Віднімати від регістру одиницю із частотою 2 Гц до 0, потім додавати до 10. Виводити результати на С_Інд. HG1, HG2
21	Занести до регістру В ДД число XX, до регістру R1 – 0X. Число з В відображати на С_Інд. HG1, HG2 із частотою 1 Гц, число з R1 відображати на С_Інд. HG3, HG4 із частотою 0,5 Гц
22	Прочитати значення регістру SP, відображати його значення на С_Інд. У наступному порядку: HG3-HG4→HG2-HG3→HG1-HG2 із часом зміни індикаторів 1 сек.
23	Занести до регістру R1 ДД число 0X, до регістру – R2 X0. Суму чисел відображати на С_Інд. HG1, HG4 із плавним запалюванням числа протягом 5 сек.
24	Занести до В ДД число 0X, до регістру R1 – X0. Молодшу тетраду суми постійно відображати на С_Інд. HG1, а старшу – поперемінно на індикаторах HG4, HG3 і HG2 з інтервалом в 1 сек.
25	Занести до регістру TH0 ДД число X0, до регістру R2 – 0X. Старшу тетраду суми постійно відображати на С_Інд. HG1, а молодшу – поперемінно на індикаторах HG2, HG3 з інтервалом в 1 сек.
26	Занести до регістру R1 число 0. Кожні 0,5 сек до числа додавати 1. Кожні 5 сек додавати 10. Рахунок вести у двійково-десятковому коді. Результат відображати на С_Інд. HG3, HG4

Продовження таблиці 4

1	*2*
27	Занести до регістру R1 число 0. Кожні 0,5 сек до числа додавати 1. Рахунок вести у двійково-десятковому коді. Результат відображати на С_Інд. HG3, HG4. По досягненню числа 20 зупинити рахунок і відображати результат з мерехтінням в 0,5 сек.
28	Занести до регістру R2 ДД число XX, до регістру R3 – XX. Одночасно відображати ці числа на С_Інд. HG3,HG4 і HG1,HG2 із частотою 1 Гц. Після 5 сек. згасити індикатори на 1сек., зациклити програму
29	Занести до регістру В ДД число X0, до регістру R1 – 0X. Суму чисел відобразити на С_Інд. HG1, HG2, А різницю із двійково-десятковою корекцією – на С_Інд. HG3, HG4
30	Занести до регістрів R1 і R4 числа 0Xh. Після множення виконати двійково-десяткове перетворення результату й вивести результат на С_Інд. (HG2, HG3, HG4 –сотні, десятки й одиниці відповідно)

Примітки: 1 ДД – двійково-десяткове число (числа у тетрадах 0…9).

2 XXh – довільне шістнадцятеричне число.

3 ДД 0X – двійково-десяткове число, старша тетрада якого дорівнює 0, а молодша має значення 0…9.

4 Індикатори HG1 і HG2 мають адреси в зовнішній пам'яті даних 0A000h (старша й молодша тетради відпов.).

5 Індикатори HG3 і HG4 мають адреси в зовнішній пам'яті даних 0B000h (старша й молодша тетради відпов.)

4.4 Практичне завдання 3

Тема. Програмування паралельного інтерфейсу КР580ВВ55.

Мета: вивчення схем динамічної індикації.

Таблиця 5 – Варіанти завдань до самостійної роботи

Варіант	**Завдання**
1	*2*
1	Занести до регістру R1 число XXh, віднімаючи від числа одиницю відображати на Д_Інд. HG5, HG6 отримане значення до нуля із частотою 1 сек. Із частотою 0,5 Гц. Перемикати Світлодіоди HL3, HL4
2	Занести до У ДД число X0, до регістру R1 – XXh, число з В. відображати на С_Інд. HG1 із частотою 1 Гц, число з R1 відображати на Д_Інд. HG5, HG6 із частотою 0,25 Гц
3	Вмик. Світлодіод HL1. Занести до регістру У ДД число 0X, до регістру R5 – X0, два розряди суми (десятки й одиниці) по черзі відображати на С_Інд. HG1, HG2 і на Д Інд. HG5, HG6 із частотою 1 Гц
4	Занести до per. R6 ДД число XX, до per.R5 ДД – XX, до регістру R0 ДД – XX, послідовно відображати ці числа по одному розряду на Д_Інд. HG5, HG6, С_Інд. HG1, HG2, HG3, HG4
5	Поперемінно вмик. світлодіоди HL8-HL7 із частотою 2 Гц. Занести до регістру R2 ДД число 0X, до регістру R1 – 0X, суму чисел відобразити на Д_Інд. HG5, HG6
6	Занести до регістру А ДД число 0X, до регістру R2 – X0, число з А відобразити на С_Інд. HG3, число з регістру R2 відображати на Д_Інд. HG5, HG6 із частотою в 0.25 Гц
7	Занести до А ДД число XX, до регістру R1 – XX, молодші два розряди суми чисел відобразити на Д_Інд. HG5, HG6, старшу тетраду на С_Інд. HG3
8	Занести до регістру R6 число XXh. Число з регістру R6 вивести в шістнадцятковій формі на Д_Інд. HG5, HG6, а його двійково-десятковий еквівалент на С_Інд. HG2-HG4
9	Занести до регістру У ДД число XX, до регістру R3 – XX, різницю чисел відобразити на Д_Інд. HG2, HG3 у ДД формі

Продовження таблиці 5

1	*2*
10	Вмик. світлодіод HL5. Занести до А ДД число XX, до регістру R5 – X0, число з А відобразити на С_Інд. HG1, HG2, число з R5 відобразити на Д_Інд. HG5,HG6
11	Занести до регістру R0 ДД число XX, поперемінно відображати мол. і ст. тетради на Д_Інд. HG5, HG6 із частотою 0,25 Гц
12	Занести до регістру R2 ДД число 0X, до регістру R5 – 0X, суму чисел відобразити на Д_Інд. HG2, HG3
13	Занести до регістру В ДД число, із частотою 2 Гц виводити це число на С_Інд. HG1, HG2 і одночасно на Д_Інд. HG5, HG6
14	Поперемінно вмик. світлодіоди HL4-HL6. Занести до комірки з адресою 0010h зовнішньої пам'яті ОЕОМ ДД число 0X, до регістру R3 – XXh, суму чисел відобразити на Д_Інд. HG5, HG6
15	Занести до регістру R1 ДД число 0X, до регістру R3 – XX, молодші два розряди суми відобразити на Д_Інд. HG5, HG6 зі змінною частотою зміни цих індикаторів, старшу на С_Інд. HG1

4.5 Практичне завдання 4

Тема. Система переривань. Опитування дискретних датчиків.

Мета: вивчення системи переривань ОЕОМ КР1816ВЕ31 і програм опитування дискретних датчиків.

Таблиця 6 – Варіанти завдань до самостійної роботи

Варіант	**Завдання**
1	*2*
1	Переривання INT0 збільшує показання лічильника на 1, стан якого виводиться на З_Інд. HG3, HG4, а переривання INT1 – зменшує
2	Реалізувати опитування клавіатури. Номер натиснутої клавіші відобр. запалюванням відпов. точки на індикаторі HG7

Продовження таблиці 6

1	*2*
3	Реалізувати опитування клавіатури. Номер натиснутої клавіші відображати на С_Інд.
4	Переривання INT0 запускає вогонь, що біжить, на світлодіодах HL1-HL8 із частотою 2 Гц, а INT1 – гасить усі світлодіоди
5	Програма виконує мерехтіння будь-якого числа на С_Інд. Переривання INT0 виконує зрушення вогню, що біжить, на світлодіодах HL1-HL8 на один крок уліво, а INT1 – вправо
6	Реалізувати опитування клавіатури. Номер клавіші вказувати позиційним кодом на світлодіодах HL1-HL8
7	Реалізувати програму введення двозначного числа із клавіатури використовуючи С_Інд. для відображення
8	Реалізувати опитування клавіатури. Номер клавіші вказувати на Д_Інд. HG5, HG6
9	Програма виконує мерехтіння будь-якого числа на С_Інд. Переривання INT0 запускає, вогонь, що біжить, на матриці HG7, а INT1 – гасить усі точки
10	Програма виконує біжучий вогонь на світлодіодах HL1-HL8. Переривання INT0 зупиняє вогонь. Повторне INT0 запускає біжучий вогонь
11	Переривання INT0 засвічує всі крапки на матриці HG7. Переривання INT1 – гасить усі точки
12	Переривання INT0 засвічує точки в шаховому порядку на матриці HG7. Переривання INT1 – гасить усі точки
13	Реалізувати програму введення тризначного числа із клавіатури, використовуючи С_Інд.
14	Переривання INT1 виконує запуск біжучої тіні на матриці HG7. Повторний виклик переривання гасить усі точки
15	Реалізувати опитування клавіатури. Номер клавіші вказувати двійковим кодом на світлодіодах HL1-HL8

4.6 Практичне завдання 5

Тема. Цифроаналогове перетворення. Робота таймерів.

Мета: вивчення методів цифроаналогового перетворення. Знайомство з таймерами ОЕОМ КР1816ВЕ31.

Таблиця 7 – Варіанти завдань до самостійної роботи

Варіант	**Завдання**
1	*2*
1	Сформувати пилкоподібну напругу із частотою повторення 50 Гц. Відображати на С_Інд число сгенерованих імпульсів
2	По натисканню S1 сформувати трикутні імпульси, передній фронт 20 мсек, задній 10 мсек.
3	По натисканню S2 сформувати трапецієподібні імпульси, передній фронт 13 мксек. задній 15 мсек.
4	Сформувати синусоїду із частотою повторення 120 Гц
5	Сформувати пилкоподібну напругу із частотою повторення 200 Гц і тривалістю переднього фронту 2 мсек.
6	По натисканню S3 сформувати синусоїду із частотою повторення 100 Гц
7	Сформувати прямокутні імпульси, із тривалістю 25мсек і шпаруватістю 4
8	По натисканню S4 сформувати трикутні імпульси, передній фронт 25 мсек., задній 5 мсек.
9	Сформувати синусоїду із частотою повторення 300 Гц. По натисканню S5 змінити частоту на 100 Гц
10	Сформувати два прямокутні імпульси, один максимальною амплітудою тривалістю й другий 2/3 амплітуди максимальної з періодом повторення 40 Гц.
11	Сформувати прямокутні імпульси, із тривалістю 25 мсек. по натисканню S6 сформувати трикутні імпульси

Продовження таблиці 7

1	*2*
12	Сформувати синусоїду із частотою повторення 70 Гц, по натисканню S7 прямокутні імпульси із тривалістю 25мсек і шпаруватістю 2
13	По натисканню S8 сформувати трикутні імпульси, передній фронт 15 мсек., задній 40 мсек.
14	По натисканню S9 сформувати трапецієподібні імпульси, передній фронт 40 мсек., задній 20 мсек.
15	Сформувати три прямокутні імпульси, один 1/3 макс. амплітуди, 2-й 2/3 амп. макс, 3-ий макс. амп. з періодом повторення 100 Гц

Примітка: для формування інтервалів часу використовувати внутрішні таймери мікроконтролера.

4.7 Практичне завдання 6

Тема. Аналого-цифрове перетворення.

Мета: навчитися вимірювати аналогову величину за допомогою цифрових пристроїв.

Таблиця 8 – Варіанти завдань до самостійної роботи

Варіант	**Завдання**
1	*2*
1	При натисканні на кнопку S10 запустити АЦП послідовного зрівноважування. Результат перетворення виводити на С_Інд у десятковій формі
2	При натисканні на кнопку S11 запустити АЦП половинних наближень. Результат перетворення виводити на С_Інд у десятковій формі
3	Запустити слідкуючий АЦП послідовного зрівноважування. Результат перетворення виводити на С_Інд у десятковій формі

Продовження таблиці 8

1	*2*
4	Запустити слідкуючий АЦП половинних наближень. Результат перетворення виводити на С_Інд у десятковій формі
5	По натисканню кнопки S1 запустити АЦП половинних наближень. Результат перетворення виводити на С_Інд у десятковій формі
6	По натисканню кнопки S2 запустити АЦП послідовного зрівноважування. Результат перетворення виводити на С_Інд у десятковій формі
7	По натисканню кнопки S3 запустити слідкуючий АЦП половинних наближень. Результат перетворення виводити на С_Інд у десятковій формі
8	По натисканню кнопки S4 запустити слідкуючий АЦП послідовного зрівноважування. Результат перетворення виводити на С_Інд у десятковій формі
9	По натисканню кнопки S5 запустити АЦП половинних наближень. Результат перетворення виводити на С_Інд у десятковій формі по натисканню кнопки S1
10	По натисканню кнопки S6 запустити АЦП послідовного зрівноважування. Результат перетворення виводити на С_Інд у десятковій формі по натисканню кнопки S2
11	По натисканню кнопки S7 запустити слідкуючий АЦП послідовного зрівноважування. Результат виводити на С_Інд у десятковій формі по натисканню S3
12	По натисканню кнопки S8 запустити слідкуючий АЦП половинних наближень. Результат виводити на С_Інд у десятковій формі по натисканню S4
13	При натисканні S9 запустити АЦП половинних наближень. Номер розряду, що врівноважується, виводити в якості точки на матрицю HG7. Після перетворення вивести результат на С_Інд у десятковій формі

Продовження таблиці 8

1	*2*
14	При натисканні S10 запустити АЦП послідовного зрівноважування. Номер кроку зрівноважування виводити на матрицю HG7 у двійковому коді. Після перетворення вивести результат на С_Інд у десятковій формі
15	При натисканні кнопки S11 запустити АЦП половинних наближень. По натисканню S2 вивести результат на Д_Інд у шістнадцятковій формі. Допускається використовувати С_Інд для старшого розряду (HG1)

5 ОПИС МІКРОКОНТРОЛЕРА PIC16F877

5.1 Загальна характеристика мікроконтролера

PIC16F877 – 8-розрядний мікроконтролер, що випускається фірмою Microchip Technology. Це спеціалізований мікропроцесор, призначений в основному для програмного керування автоматизованими системами, автомобільними й електричними двигунами, пристроями передачі інформації й вимірювальними приладами. На відміну від універсальних процесорів, він має розвинені засоби взаємодії із зовнішніми пристроями й більш просту систему команд.

PIC16F877 являє собою мікросхему з 40 виводами, з яких 32 призначені для передачі інформації від зовнішнього пристрою або до зовнішнього пристрою. Виконувана програма зберігається в перепрограмувальному ПЗП, куди вона заноситься спеціальним пристроєм – програматором. Необхідні дані, змінні, результати нескладних розрахунків і лічильники циклів зберігаються в ОЗП й губляться при вимиканні живлення. Щоб уникнути втрати даних при цьому, можна використовувати 256 комірок енергонезалежної пам'яті даних.

Мікроконтролер PIC16F877 має наступні технічні характеристики:

– Мікроконтролер виконаний за високошвидкісною RISC технологією. Висока продуктивність досягається за рахунок застосування конвеєрної архітектури й малої кількості команд (усього 35).

– Тактова частота МК становить 20 МГц, при цьому час тривалості машинного циклу досягає 200 нс.

– 8Кх14 слів FLASH пам'яті програм.

– 368х8 байт пам'яті даних (ОЗП).

– 256х8 байт EEPROM пам'яттю даних.

– Система переривань має 14 джерел.

Характеристика периферійних модулів:

– Два 8-розрядних таймера/лічильника.

– Один 16-розрядний таймер/лічильник з можливістю підключення зовнішнього кварцового резонатора.

– Два модулі захват/порівняння/ШІМ (CCP):

1) 16-розрядне захоплення (максимальна розв'язна здатність 12,5 нс);

2) 16-розрядне порівняння (максимальна розв'язна здатність 200 нс);

3) 10-розрядний ШІМ.

– 8-канальний 10-розрядний АЦП.

– Послідовний синхронний порт.

– Ведучий/ведений режим SPI.

– Ведучий/ведений режим I^2C.

– Послідовний асинхронний прийомопередавач USART с підтримкою детектування адреси.

– Ведений 8-розрядний паралельний порт PSP з підтримкою зовнішніх сигналів $\overline{RD}$, $\overline{WR}$, $\overline{CS}$.

5.2 Внутрішній пристрій ядра мікроконтролера

Внутрішній пристрій показаний на рисунку 30. МК можна умовно розділити на дві частини: обчислювальне ядро (сірий колір) і периферійні модулі (білий колір).

Обчислювальне ядро працює в такий спосіб. Програма роботи МК перебуває в FLASH пам'яті програм. Програма виконується послідовно доти, поки не зустрінеться команда переходу. Регістр команд (РК) містить поточну команду на час її дешифрації й виконання, а програмний лічильник (PC) призначений для зберігання адреси наступної команди. Коли поточна команда завершена, то:

– За адресою з PC проводиться вибірка команди з пам'яті програм у РК.

– При дешифрації цієї команди, проводиться інкремент PC на одиницю й PC адресує наступну команду.

– Коли виконання даної команди закінчується, зміст PC видається пам'яті програм і цикл повторюється.

Команди безумовного переходу дозволяють змінити природній порядок проходження команд шляхом заміщення вмісту PC (тобто адреси наступної одна за одною команди) адресою, обумовленою самою командою переходу.

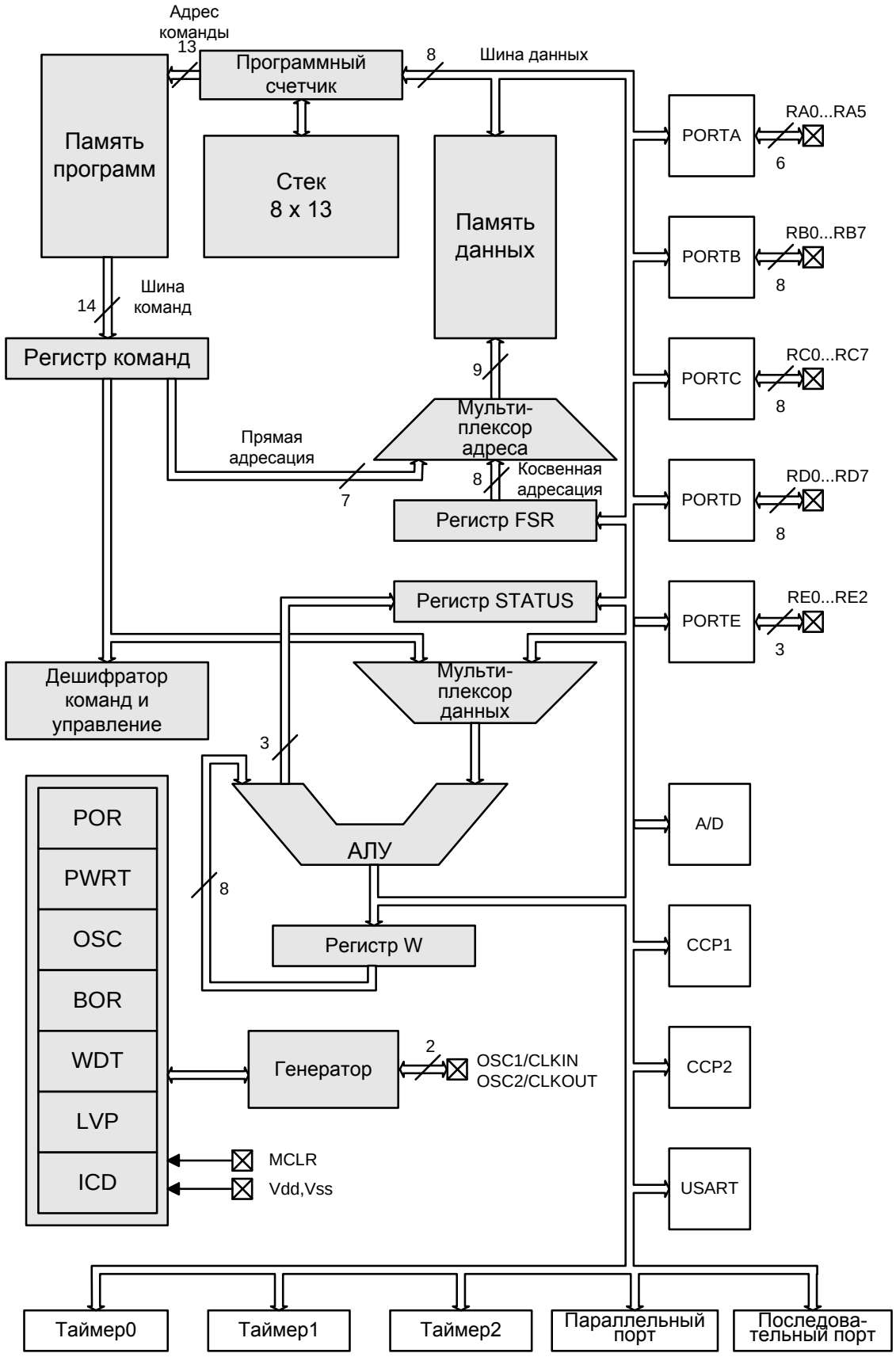


Рисунок 30 – Внутрішній пристрій мікроконтролера PIC16F877

Укр.!

Команди умовних переходів заміщають або не заміщають зміст PC залежно від ознак результатів попередніх команд. Ознаки результатів попередніх команд перебувають в регістрі STATUS. У цьому регістрі є біти, що показують такі умови, як одержання в попередніх операціях позитивного, негативного або нульового результату. Коли реалізований перехід, починається нова послідовність команд із адреси, до якої здійснений перехід.

Цикли реалізуються за допомогою команд умовних переходів.

Дії, пов'язані з викликом підпрограми, як і в інших МК, також заміняє зміст PC на адресу переходу, і також при цьому запам'ятовується поточний зміст PC у стеці. Команда повернення повинна відновити в PC адресу повернення, щоб після завершення підпрограми тривало послідовне виконання основної програми.

Добування команд із пам'яті команд і їх виконання ядром відбувається за 4 машинних такти, завдяки конвеєрній обробці, з яких складається машинний цикл. Діаграма вибірки команд і їх виконання подані на рисунку 31.

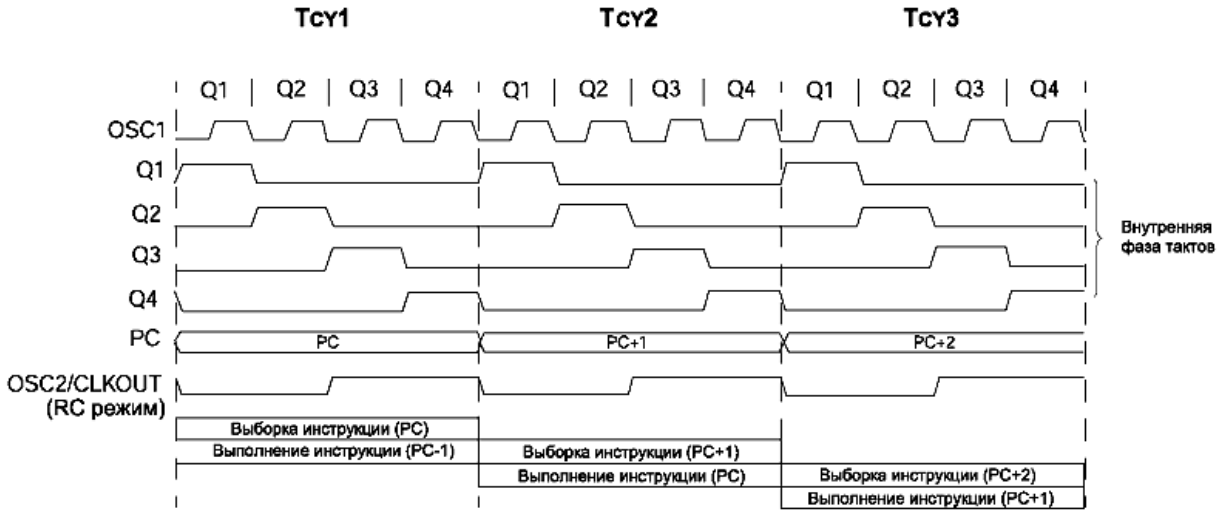


Рисунок 31 – Діаграма машинних циклів Укр.!

Мікроконтролери PIC містять 8-розрядний універсальний арифметичний модуль (АЛП) і 8-розрядний робочий регістр (W). АЛП виконує арифметичні й булеві операції між робочим регістром і будь-яким регістром пам'яті даних.

Вхідні дані АЛП залежно від коду операції можуть перебувати:

– У регістрі W.

– У регістрі команд.

– У пам'яті даних.

Результат виконання операції може зберігатися як у робочий регістр W, так і в будь-який регістр пам'яті даних (файловий регістр):

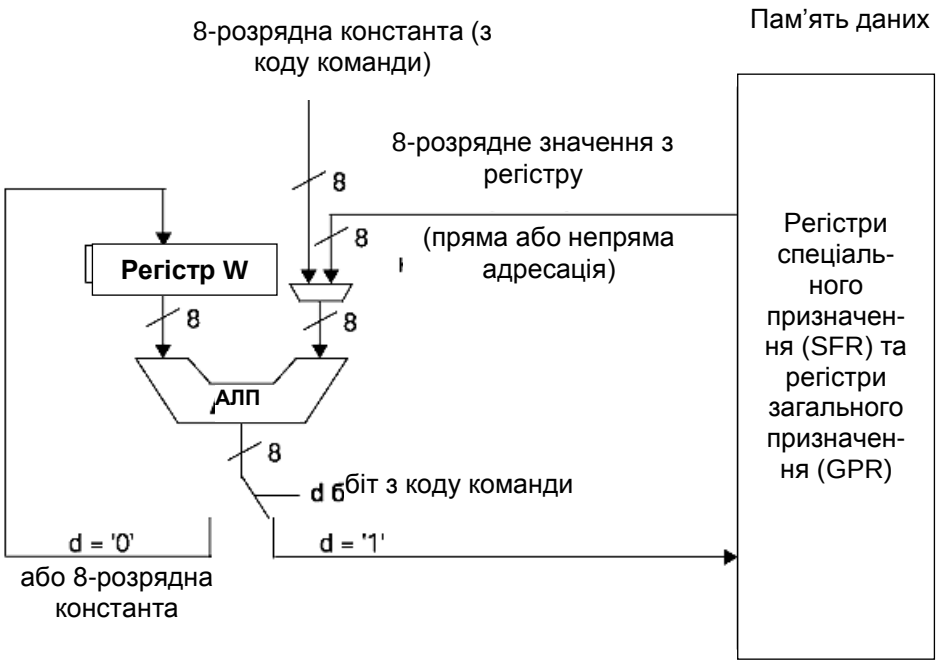


Рисунок 32 – Арифметико-логічний пристрій і робочий регістр

8-розрядне АЛП може виконувати додавання, вирахування, порозряде зрушення й логічне операції. Арифметичні операції виконуються за принципом доповнення до двох, якщо не зазначене інакше. У командах із двома операндами: перший операнд перебуває в робочому регістрі W, а другий операнд розташований у регістрі пам'яті даних або константа. У командах з одним операндом: операндом є регістр W або регістр пам'яті даних.

Регістр W – не адресований 8-розрядний робочий регістр, який використовується в операціях АЛП. Залежно від типу команди й результат команди АЛП може впливати на наступні прапори стану в регістрі STATUS: перенос (C), напівперенос (DC), прапор нульового результату (Z). Біти C і DC працюють як біти займу й десяткового займу при виконанні команд вирахування.

У регістрі STATUS утримуються прапори стану АЛП. Прапори причини скидання мікроконтролера й біти керування банками пам'яті даних.

Регістр STATUS (рис. 33) може бути адресований будь-якою командою, як і будь-який інший регістр пам'яті даних. Якщо звертання до регістру STATUS виконується командою, яка впливає на прапори Z, DC і C, то зміна цих трьох бітів командою заблоковане. Ці біти скидаються або встановлюються згідно з логікою ядра мікроконтролера. Команди зміни регістру STATUS також не впливають на біти -TO і -PD. Тому результат виконання команди з регістром STATUS може відрізнятися від очікуваного. Наприклад, команда CLRF STATUS скине три старші біти й установить біт Z.

При зміні бітів регістру STATUS рекомендується використовувати команди, що не впливають на прапори АЛП (SWAPF, MOVWF, BCF і BSF).

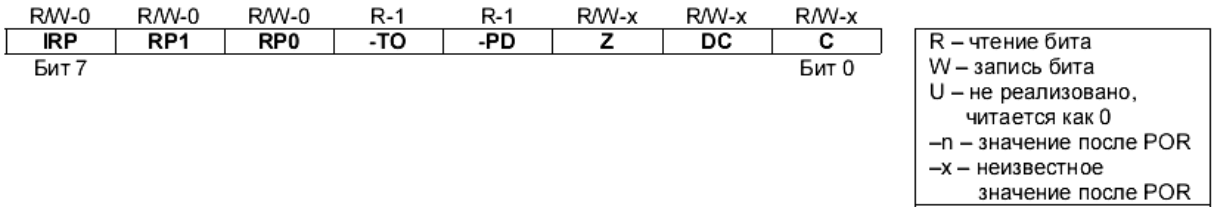


Рисунок 33 – Регістр STATUS

Призначення бітів регістру STATUS:

IRP – Біт вибору банку при непрямій адресації (0 – банк РЗП 0,1; 1 – банк РЗП 2,3).

RP0, RP1 – Біти вибору банку РЗП при прямій адресації.

-TO – Прапор переповнення сторожового таймера.

-PD – Прапор включення живлення (0 – після команди SLEEP).

Z – Прапор нульового результату операції.

DC – Прапор десяткового переносу/займу в молодшій тетроді.

C – Прапор переносу/займу при виконанні операції.

5.3 Слово конфігурації мікроконтролера

Біти конфігурації дозволяють настроїти деякі режими роботи мікроконтролера відповідно до вимог конкретного пристрою. При включенні живлення стан цих бітів визначає режим роботи мікроконтролера. Біти конфігурації розташовано за адресою 2007h у пам'яті програм. Програма користувача не може змінювати й читати стан бітів конфігурації (ця операція можлива тільки в режимі програмування мікроконтролера).

Біти конфігурації можуть бути запрограмовані (читаються як «0») або залишені без зміни (читаються як «1»), щоб вибрати режим роботи мікроконтролера.

Розташування бітів конфігурації мікроконтролера презентовано на рисунку 34.

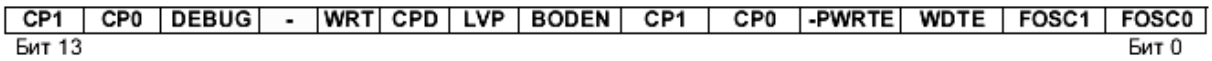


Рисунок 34 – Слово конфігурації

Біти даного слова виконують наступні функції:

CP1, CP0 – Біти захисту пам'яті програм:

11 – Захист пам'яті програм виключена.

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.

DEBUG – Біт включення внутрісхемного налагодження (ICD):

1 – Налагодження відключене.

0 – Налагодження включене (лінії RB6, RB7 використовуються відладником).

WRT – Біт дозволу запису у FLASH пам'ять програм:

1 – Запис у FLASH пам'ять програм дозволена через регістр керування EECON.

1 – Запис у FLASH пам'ять програм заборонена через регістр керування EECON.

CPD – Біт захисту від запису у EEPROM даних:

0 – Захист запису у EEPROM пам'ять даних включена.

LVP – Біт дозволу низьковольтного програмування:

1 – Низьковольтне програмування включене (вивід RB3/PGM використовується для програмування).

BODEN – Біт дозволу скидання по зниженню напруги живлення:

1 – Дозволене скидання BOR.

-PWRTE – Біт дозволу роботи таймера включення живлення:

0 – PWRTE включений;

WDTE – Біт дозволу роботи сторожового таймера:

1 – Робота сторожового таймера дозволена.

FOSC1, FOSC0 – Біти вибору режиму роботи тактового генератора:

11 – RC-Генератор.

10 – HS-Генератор (високочастотний резонатор 4<f<20 МГц).

01 – XT-Генератор (звичайний резонатор 0,2<f<4 МГц).

00 – LP-Генератор (низькочастотний резонатор f<200 кГц).

У макроасемблері MPASM надається можливість визначити біти конфігурації у вихідному тексті програми за допомогою директиви CONFIG. Використання директиви CONFIG гарантує запис бітів конфігурації при програмуванні мікроконтролера, що зменшує ризик запрограмувати неправильне слово конфігурації.

Приклад використання директиви CONFIG:

```
      LIST  p = p16F877             ; Указуємо тип процесора
      INCLUDE <P16F877.INC>         ; Підключаємо визначення регістрів
; Настроювання бітів конфігурації
      _CONFIG _XT_OSC & _PWRTE_ON & _BQDEN_OFF & _CP_OFF & _WDT_ON
org 0x00                            ; Початок пам'яті програм
RESET_ADDR                          ; Перша команда після скидання
            ...                     ; Наша програма
end
```

Список стандартних імен, використовуваних у директиві CONFIG для процесора PIC16F877, зазначено в таблиці 9.

Таблиця 9 – Стандартні імена директиви CONFIG

Призначення	**Символ**
1	*2*
Тактовий генератор	_RC_OSC
	_LP_OSC
	_XT_OSC

Продовження таблиці 9

1	*2*
	_HS_OSC
Сторожовий таймер WDT	_WDT_ON
	_WDT_OFF
Таймер включення живлення PWRT	_PWRTE_ON
	_PWRTE_OFF
Скидання зі зниження напруги живлення	_BODEN_ON
	_BODEN_OFF
Захист коду програми	_CP_ALL
	_CP_ON
	_CP_75
	_CP_50
	_CP_OFF
Захист EEPROM пам'яті даних	_DP_ON
	_DP_OFF

За адресою 2000h-2003h розташована пам'ять, призначена для зберігання контрольної суми пам'яті програм й іншої службової інформації (ID). Вони доступні для читання й запису тільки в режимі програмування мікроконтролера. Використовуються тільки молодші 4 біта в кожній комірці.

5.4 Організація пам'яті програм

Мікроконтролери середнього сімейства мають 13-розрядний лічильник команд, здатний адресувати 8 Кх14 слів пам'яті програм, і 14-розрядну шину даних пам'яті програм. Усі команди мікроконтролера складаються з 14-розрядного слова.

Уся пам'ять програм розподілена на 4 сторінки по 2 К слів кожна (0000h-07FFh. 0800h-0FFFh. 1000h-17FFh. 1800h-1FFh). На рисунку 36 показана карта пам'яті програм і 8-рівневий апаратний стек.

Для переходу між сторінками пам'яті програм необхідно змінити старші біти регістру лічильника команд PC, записом у регістр спеціального призначення PCLATH (старший байт лічильника команд). Змінивши значення регістру PCLATH і виконавши команду розгалуження, лічильник команд PC перетне межу сторінки пам'яті програм без додаткового втручання користувача.

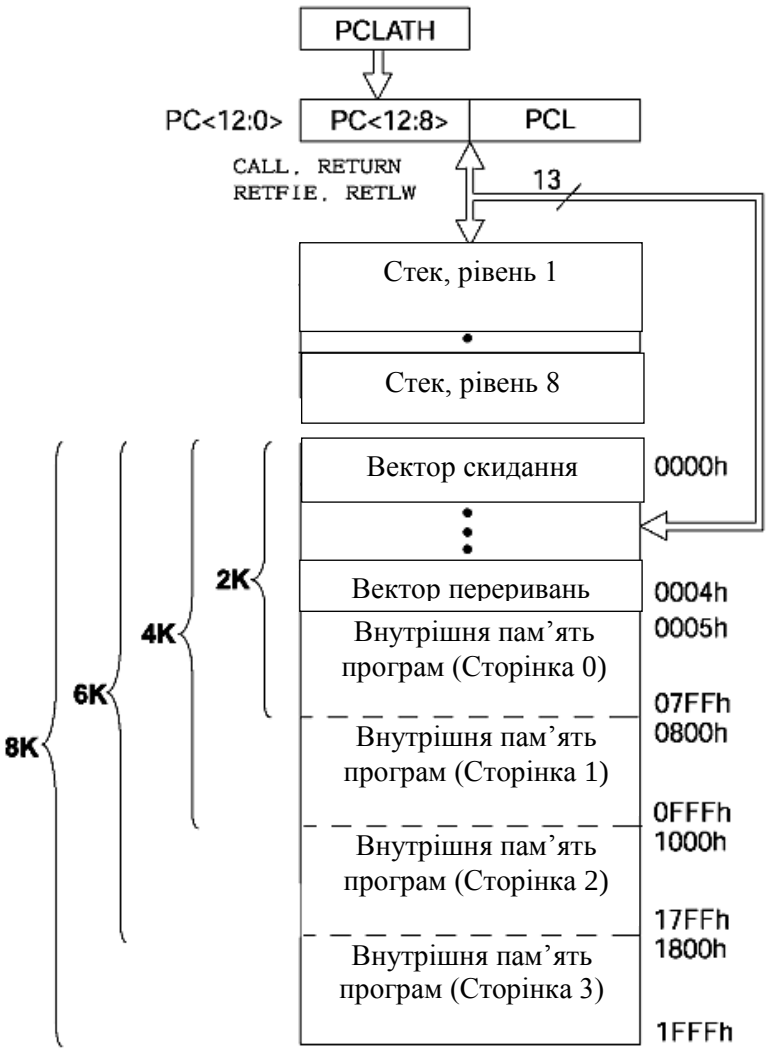


Рисунок 36 – Карта пам'яті програм і стек

У будь-якому мікроконтролері Picmicro скидання приведе до очищення лічильника команд (PC), установлюючи адресу 0h. Адреса 0000h

називається «адресу вектора скидання», тому що буде виконаний перехід по цій адресі при скиданні мікроконтролера. Разом з лічильником команд (PC) очищається регістр PCLATH, установлюючи робочу сторінку пам'яті програм 0.

Коли виникає дозволене переривання, у лічильник команд PC записується адреса 0004h, називаний «адреса вектора переривань», при цьому значення регістру PCLATH не змінюється.

Якщо в підпрограмі обробки переривань потрібно виконувати команди розгалуження, то необхідно попередньо записати в регістр PCLATH значення, що визначає потрібну сторінку пам'яті програм. Перш ніж регістр PCLATH буде змінений, його значення повинне бути збережене в іншому регістрі пам'яті даних, а потім відновлене перед виходом з підпрограми обробки переривань.

Лічильник команд PC

13-розрядний регістр лічильника команд PC указує адресу обираної команди для виконання. Молодший байт лічильника програм PCL доступний для читання й запису. Старший байт PCH, що містить біти лічильника команд PC<12:8>, не доступний для читання й запису. Усі операції з регістром PCH відбуваються через додатковий регістр PCLATH.

Перехід, що обчислюється

Перехід, що обчислюється, може бути виконаний командою збільшення до регістру PCL (наприклад, ADDWF PCL). При виконанні переходу, що обчислюється, слід опікуватися про те, щоб значення PCL не перетнуло межу блоку пам'яті (кожний блок 256 байт).

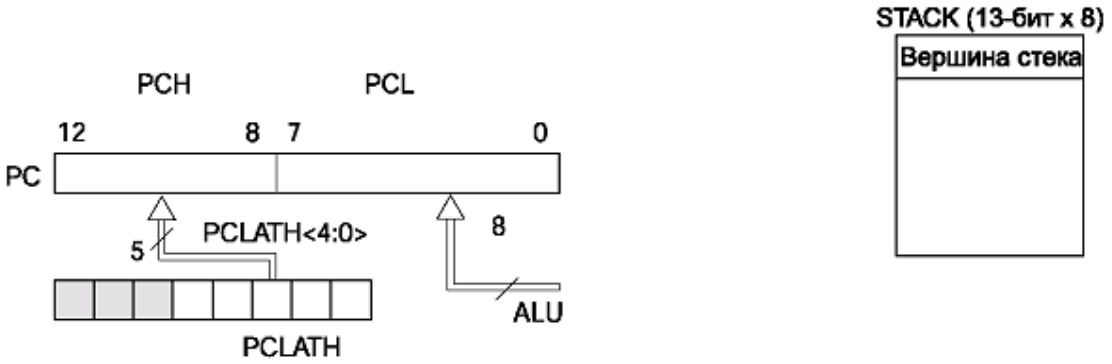
При записі значення в регістр PCL автоматично відбувається перезапис 5 молодших біт з регістру PCLATH<4:0> у регістр PCH.

Апаратний стік

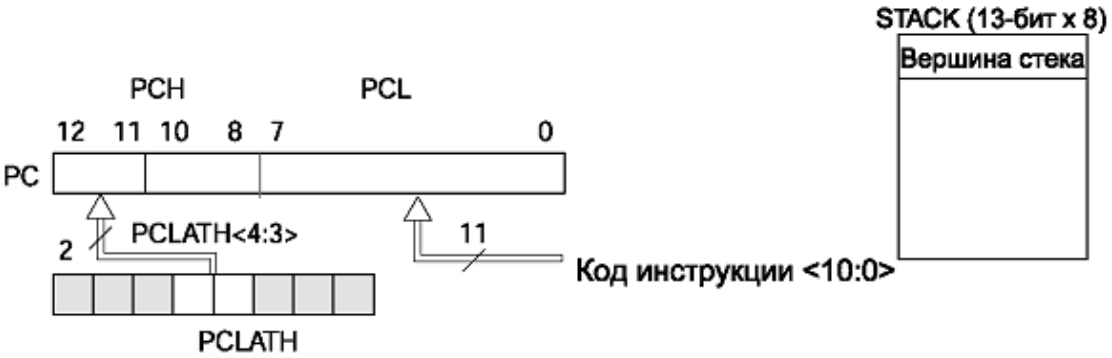
Стік підтримує до 8 рівнів вкладеності підпрограм користувача, включаючи обробку переривань. У стеці зберігається адреса повернення до основної програми.

У мікроконтролерах середнього сімейства Picmicro реалізований 8-рівневий 13-розрядний апаратний стек. Стек не має відображення на пам'ять програм і пам'ять даних, не можна записати або прочитати дані зі стека. Значення лічильника команд заноситься до вершини стека при

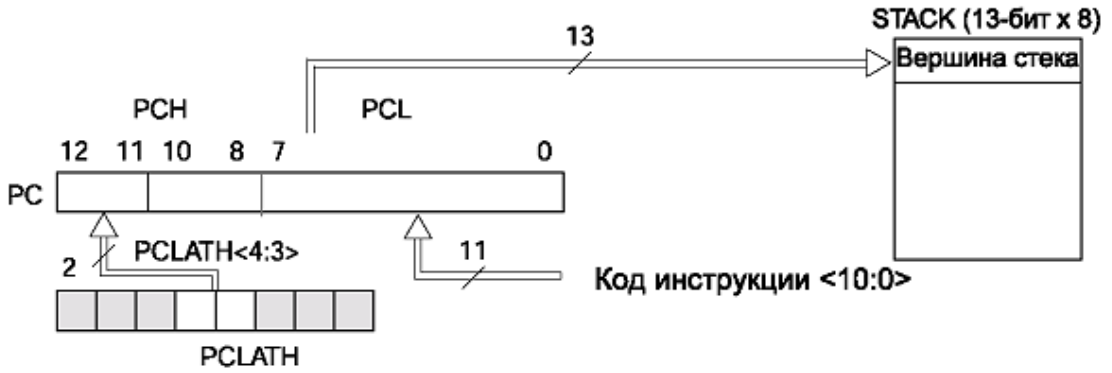
виконанні інструкцій переходу на підпрограму (CALL) або обробку переривань. Читання зі стека й запис у лічильник команд PC відбувається при виконанні інструкцій повернення з підпрограми або обробки переривань (RETURN, RETLW, RETFIE), при цьому значення регістру PCLATH не змінюється.



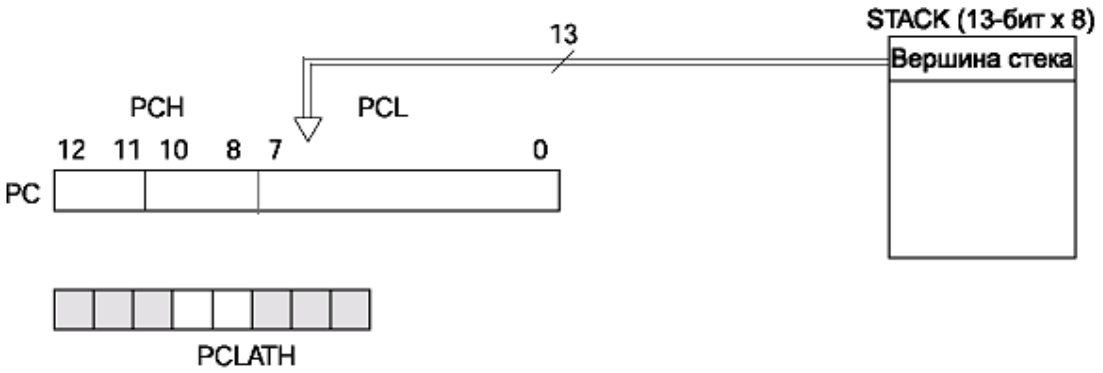
а – Безпосередній запис у регістр PCL (PLATH<4:0> →PCH)



б – Зміна значення PC при виконанні інструкції GOTO (PLATH<4:3> →PCH).



в – Зміна значення PC при виконанні інструкції CALL (PLATH<4:3> →PCH). Старе значення PC зберігається в стеку



г – Повернення з підпрограми. Значення PC вертається зі стека

Рисунок 37 – Можливі варіанти зміни стану PC укр.!

Після 8 записів у стек, дев'ятий запис запишеться на місце першого, а десятий запис замінить другий і так далі, що порушує послідовність збережених даних. Програміст повинен сам стежити за вкладеністю підпрограм і враховувати, що в будь-який момент часу, можливо, буде оброблятися переривання, якому необхідний 1 запис у стеку.

Команди переходів (CALL, GOTO) у мікроконтролерах середнього сімейства Picmicro мають 11-розрядне поле для вказівки адреси, що дозволяє безпосередньо адресувати 2К слів пам'яті програм. Мікроконтролер PIC16F877 має пам'ять програм 8Кслів. 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 регістру PCLATH<4:3> для адресації необхідної сторінки (рис. 37, б, в).

При виконанні інструкцій повернення з підпрограми 13-розрядне значення для лічильника програм PC береться з вершини стека, тому маніпуляція бітами регістру PCLATH<3:4> не потрібна (рис. 37, г).

Приклад перемикання з 0 в 1 сторінку пам'яті програм для виконання підпрограми в 1-й сторінці:

```
ORG 0x500
      BSF PCLATH,3           ; Вибір сторінки 1 (800h-fffh)
      CALL SUBI_PI           ; Перехід на сторінку 1 (800h-fffh)
            ...
;
ORG 0x900
SUBI_PI                      ; Сторінка 1 (800h-fffh)
            ...
      RETURN                 ; Повернення на сторінку 0 (000h-7Ffh)
```

5.5 Організація пам'яті даних

Пам'ять даних розподіляється на регістри двох типів:

– Регістри спеціального призначення (SFR), що управляють роботою мікро контролера.

– Регістри загального призначення (GPR), призначені для зберігання даних програми.

Пам'ять даних розподілена на банки, що містять регістри загального й спеціального призначення. Регістри загального призначення розміщаються в різних банках пам'яті даних для того, щоб була можливість організувати більш 96 байт ОЗП. Регістри спеціального призначення призначені для керування периферійними модулями й функціями мікроконтролера. Керування банками пам'яті виконується бітами в регістрі STATUS<7:5>. На рисунку 38 зображена карта пам'яті даних.

Адрес		Адрес		Адрес		Адрес	
Регистр косвенной адресации	0x000	Регистр косвенной адресации	0x080	Регистр косвенной адресации	0x100	Регистр косвенной адресации	0x180
TMR0	0x001	OPTION_REG	0x081	TMR0	0x101	OPTION_REG	0x181
PCL	0x002	PCL	0x082	PCL	0x102	PCL	0x182
STATUS	0x003	STATUS	0x083	STATUS	0x103	STATUS	0x183
FSR	0x004	FSR	0x084	FSR	0x104	FSR	0x184
PORTA	0x005	TRISA	0x085		0x105		0x185
PORTB	0x006	TRISB	0x086	PORTB	0x106	TRISB	0x186
PORTC	0x007	TRISC	0x087		0x107		0x187
PORTD	0x008	TRISD	0x088		0x108		0x188
PORTE	0x009	TRISE	0x089		0x109		0x189
PCLATH	0x00A	PCLATH	0x08A	PCLATH	0x10A	PCLATH	0x18A
INTCON	0x00B	INTCON	0x08B	INTCON	0x10B	INTCON	0x18B
PIR1	0x00C	PIE1	0x08C	EEDATA	0x10C	EECON1	0x18C
PIR2	0x00D	PIE2	0x08D	EEADR	0x10D	EECON2	0x18D
TMR1L	0x00E	PCON	0x08E	EEDATH	0x10E	Резерв	0x18E
TMR1H	0x00F		0x08F	EEADRH	0x10F	Резерв	0x18F
T1CON	0x010		0x090		0x110		0x190
TMR2	0x011	SSPCON2	0x091		0x111		0x191
T2CON	0x012	PR2	0x092		0x112		0x192
SSPBUF	0x013	SSPADD	0x093		0x113		0x193
SSPCON	0x014	SSPSTAT	0x094		0x114		0x194
CCPR1L	0x015		0x095		0x115		0x195
CCPR1H	0x016		0x096	Регистры общего назначения 16 байт	0x116	Регистры общего назначения 16 байт	0x196
CCP1CON	0x017		0x097		0x117		0x197
RCSTA	0x018	TXSTA	0x098		0x118		0x198
TXREG	0x019	SPBREG	0x099		0x119		0x199
RCREG	0x01A		0x09A		0x11A		0x19A
CCPR2L	0x01B		0x09B		0x11B		0x19B
CCPR2H	0x01C		0x09C		0x11C		0x19C
CCP2CON	0x01D		0x09D		0x11D		0x19D
ADRESH	0x01E	ADRESL	0x09E		0x11E		0x19E
ADCON0	0x01F	ADCON1	0x09F		0x11F		0x19F
Регистры общего назначения	0x020	Регистры общего назначения 80 байт	0x0A0	Регистры общего назначения 80 байт	0x120	Регистры общего назначения 80 байт	0x1A0
	0x07F	Доступ к 0x070-0x07h	0x0EF		0x16F		0x1EF
			0x0F0	Доступ к 0x070-0x07h	0x170	Доступ к 0x070-0x07h	0x1F0
	0x07F		0x0FF		0x17F		0x1FF
Банк 0		Банк 1		Банк 2		Банк 3	

Рисунок 38 – Карта пам'яті мікроконтролера PIC16F877 укр!

Щоб передати дані з одного регістру в інший, необхідно використовувати додатковий регістр W. Ця операція виконується двома командами за два машинні цикли мікроконтролера.

Звертання до всіх регістрів пам'яті даних може бути виконане прямою або непрямою адресацією. **При прямій адресації** для вказівки банку пам'яті даних необхідно використовувати біти PR1-PR0 регістру

STATUS. **У випадку непрямої адресації** адреса регістру зберігається в FSR (рис. 39), а в біті IRP регістру STATUS вказується, до якої пари банків пам'яті даних виконують звертання (0/1 або 2/3). Для виконання непрямої адресації необхідно звернутися до регістру INDF. Звертання до регістру INDF фактично викличе дію з регістром, адреса якого зазначена в FSR. Непряме читання регістру INDF (FSR=0) дасть результат значення комірки 00h. Непрямий запис у регістр INDF не викличе ніяких дій (викликає вплив на прапори АЛП в регістрі STATUS).

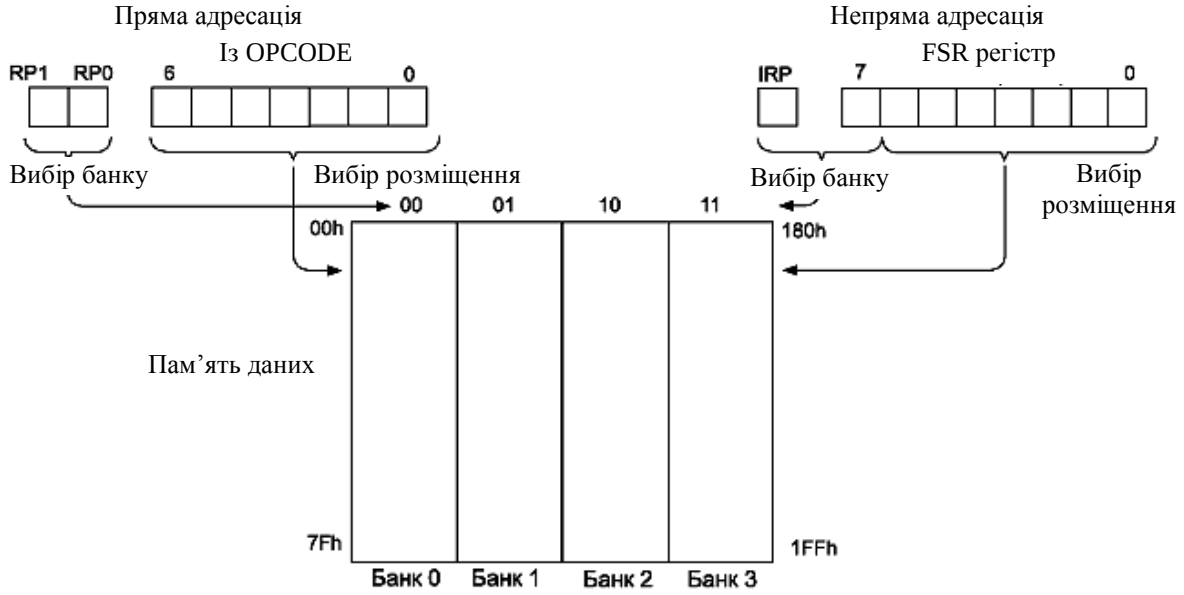


Рисунок 39 – Механізми прямої і непрямої адресації

Приклад використання непрямої адресації для відчищання блоку пам'яті з адресами 20h-2Fh:

```
      BCF STATUS, IRP          ; Установити банк 0,1
      MOVLW 0X20               ; Указати перший регістр в ОЗП
      MOVWF FSR
NEXT:
      CLRF INDF                ; Очистити регістр
      INCF FSR, F              ; Збільшити адресу
      BTFSS FSR, 4             ; Завершити?
      GOTO NEXT                ; Ні, продовжити очищення

CONTINUE:
                               ;Так
```

Приклад перемикання меж пам'яті даних для прямої адресації:

```
      CLRF STATUS              ; Очищення регістру STATUS (Банк 0)
```

```
;
      BSF STATUS,RP0          ; Банк 1
      ...
      BCF STATUS,RP0          ; Банк 0
      ...
      MOVLW 0x60              ; Установити RP0 і RP1 в STATUS регістрі
      XORWF STATUS,F          ; (Банк 3)
      ...
      BCF STATUS,RP0          ; Банк 2
      ...
      BCF STATUS,RP1          ; Банк 0
```

Регістри загального призначення (GPR)

Регістри загального призначення розміщаються в різних банках пам'яті даних. Ці регістри не ініціалізуються при скиданні по включенню живлення й мають невідоме значення, а при всіх інших скиданнях мікроконтролера не змінюють свого значення.

Звертання до регістрів загального призначення може бути виконане прямою або непрямою адресацією (через регістри FSR і INDF). У мікроконтролері існують регістри загального призначення, адресовані до однеї і тієї же комірки ОЗП, незалежно від поточного банку пам'яті даних. При адресації до старшої області пам'яті даних (останні 16 комірок) у будь-якому банку відбувається «дзеркальне» отображення тільки до регістрів з адресами 70h-7Fh.

Регістри спеціального призначення (SFR)

Регістри спеціального призначення використовуються для керування ядром і периферійними модулями мікроконтролера. Ці регістри реалізовані як статична ОЗП. Опис регістрів SFR, керуючих периферійними модулями, дивіться у відповідному розділі.

Регістри спеціального призначення розміщені в різних банках пам'яті даних, а деякі з регістрів відображаються у всіх банках.

Перемикання робочого банку пам'яті виконується настроюванням бітів RP1-RP0 регістру STATUS. При скиданні по включенню живлення й інших видах скидання мікроконтролера в деякі регістри спеціального призначення записується певне значення. Існують регістри SFR, які містять невідоме значення при скиданні по включенню живлення, а при інших видах скидання не змінюються.

Звертання до регістрів спеціального призначення може бути виконане прямою або непрямою адресацією.

5.6 Система переривань мікроконтролера

Мікроконтролер PIC16F877 має 14 джерел переривань. Регістр INTCON містить прапори окремих переривань, біти дозволу цих переривань і біт глобального дозволу переривань. Структурна схема логіки переривань показана на рисунку 40.

Якщо біт GIE (INTCON<7>) установлений в «1», дозволені всі немасковані переривання. Якщо GIE=0, то всі переривання заборонені. Кожне переривання окремо може бути дозволене або заборонене установкою/скиданням відповідного біта в регістрах INTCON (рис. 41), PIE1 (рис. 42) і PIE2 (рис. 43). При скиданні мікроконтролера біт GIE скидається в «0».

При поверненні з підпрограми обробки переривання, за командою RETFIE, біт GIE апаратно встановлюється в «1», дозволяючи всі немасковані переривання.

У регістрі INTCON перебувають прапори наступних переривань: зовнішнього сигналу INT, зміни рівня сигналу на входах RB7-RB4, переповнення TMR0.

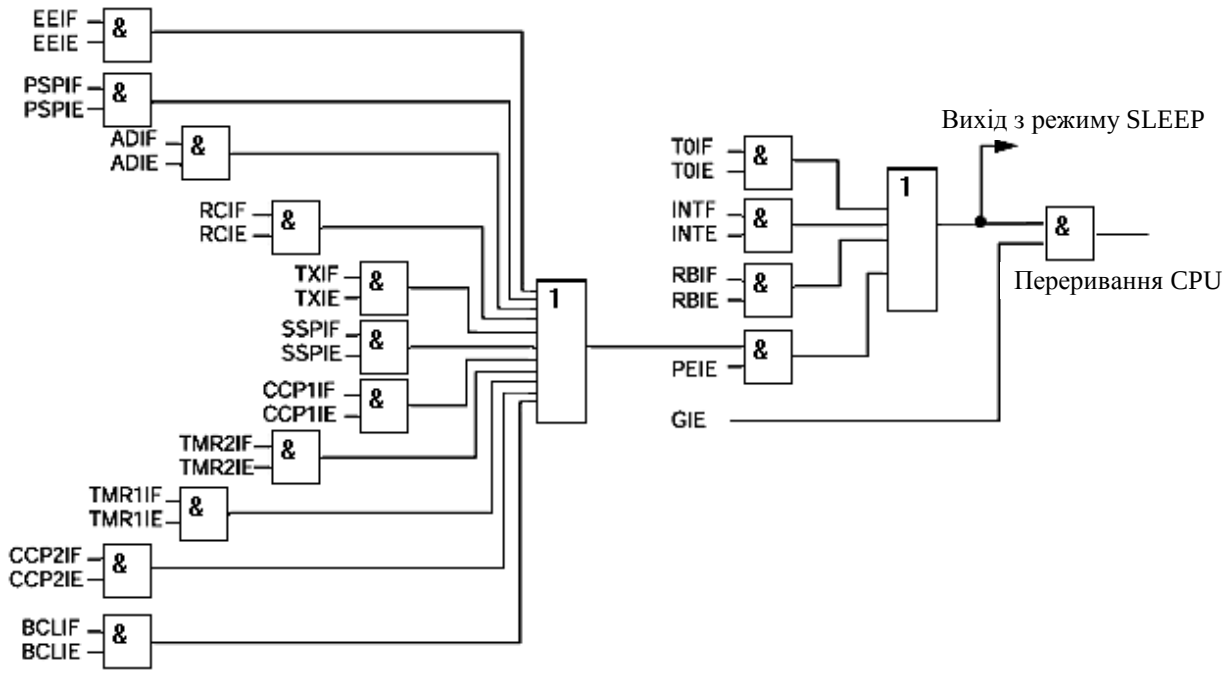


Рисунок 40 – Структурна схема системи переривань

У регістрах PIR1 (рис. 44), PIR2 (рис. 45) утримуються прапори переривань периферійних модулів мікроконтролера, а в регістрах PIE1, PIE2 відповідні біти дозволу переривань. У регістрі INTCON перебуває 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 записується у 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.

Прапори переривань установлюються незалежно від стану відповідних бітів маски й біта GIE.

Зовнішнє переривання із входу RB0/INT відбувається за переднім фронтом сигналу, якщо біт INTEDG (OPTION_REG<6>) установлений в «1»; за заднім фронтом сигналу, якщо біт INTEDG скинутий в «0». Коли активний фронт сигналу з'являється на вході RB0/INT, біт INTF (INTCON<1>) установлюється 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 у режим SLEEP. Стан біта GIE визначає, переходити на підпрограму обробки переривань після виходу з режиму SLEEP.

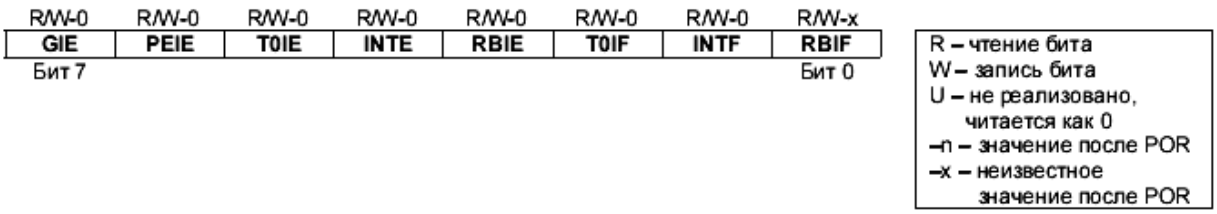


Рисунок 41 – Регістр керування перериваннями INTCON укр.!

Призначення бітів регістру INTCON:

GIE – Біт загального дозволу переривань.

PEIE – Біт дозволу переривань від периферійних модулів.

T0IE – Біт дозволу переривань за переповненням TMR0.

INTE – Біт дозволу переривань за зовнішним входом INT.

RBIE – Біт дозволу переривань за зміною рівня на входах RB4-RB7. Звичайно використовується для обслуговування клавіатурної матриці.

T0IF – Прапор переривання за переповненням таймера TMR0 (скидається програмно).

INTF – Прапор переривання за зовнішним входом INT (скидається програмно).

RBIF – Прапор переривання за зміною рівня сигналу на входах RB4-RB7 (скидається програмно).

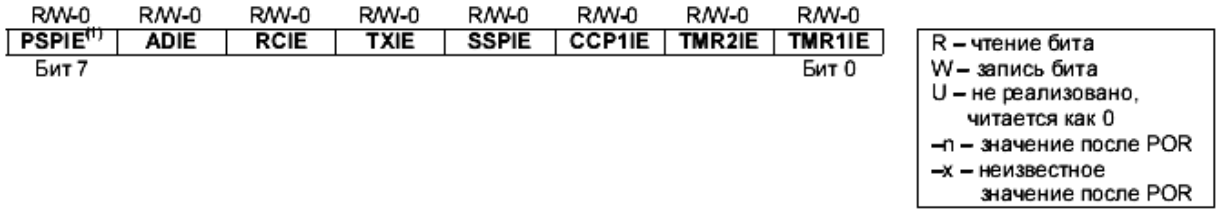


Рисунок 42 – Регістр керування перериваннями PIE1 укр!

Призначення бітів регістру PIE1:

PSPIE – Дозвіл переривань запису/читання відомого паралельного порту.

ADIE – Дозвіл переривань по закінченню перетворення АЦП.

RCIE – Дозвіл переривань від приймача УСАПП.

TXIE – Дозвіл переривань від передавача УСАПП.

SSPIE – Дозвіл переривань від модуля синхронного послідовного порту.

CCP1IE – Дозвіл переривань від модуля CCP1.

TMR2IE – Дозвіл переривань за переповненням TMR2.

TMR1IE – Дозвіл переривань за переповненням TMR1.

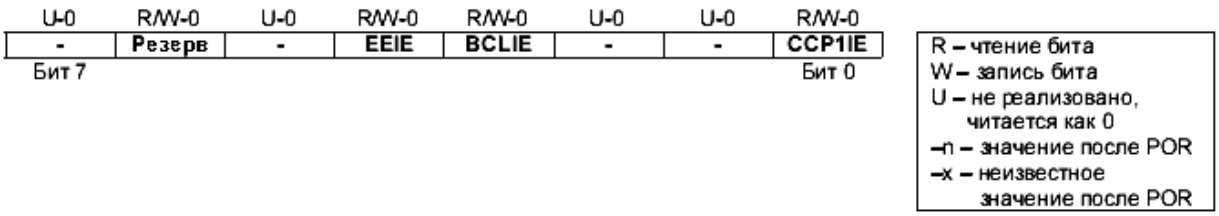


Рисунок 43 – Регістр керування перериваннями PIE2 укр!

Призначення бітів регістру PIE2:

EEIE – Дозвіл переривань по закінченню запису у EEPROM даних;

BCLIE – Дозвіл переривань під час виникнення колізій на шині;

CCP2IE – Дозвіл переривань від модуля CCP2.

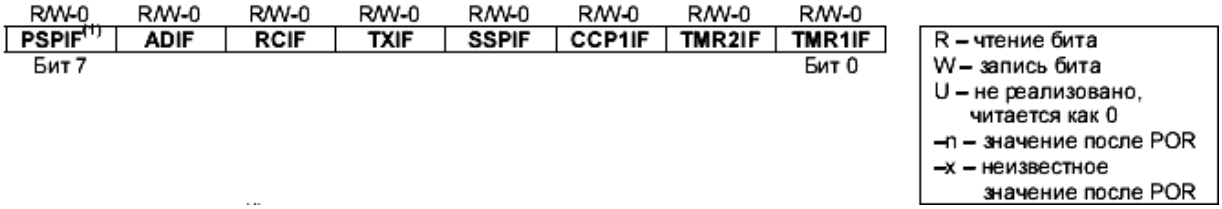


Рисунок 44 – Регістр прапорів периферійних модулів PIR1 укр!

Призначення бітів регістру PIR1:

PSPIF – Прапор переривання від відомого паралельного порту після операції читання/запису.

ADIF – Прапор переривання від АЦП після завершення перетворення.

RCIF – Прапор переривання від приймача УСАПП під час заповнення буфера приймання.

TXIF – Прапор переривання від передавача УСАПП під час спустошення буфера передачі.

SSPIF – Прапор переривання від модуля MSSP (скидається програмно).

CCP1IF – Прапор переривання від модуля CCP1:

а) У режимі захоплення виконане захоплення значення TMR1 (скидається програмно).

б) У режимі порівняння значення TMR1 досягло величини, записаної в регістри CCPR1H:CCPR1L (скидається програмно).

в) У режимі ШІМ не використовується.

TMR2IF – Прапор переривання під час переповнення таймера TMR2 (скидається програмно).

TMR1IF – Прапор переривання під час переповнення таймера TMR1 (скидається програмно).

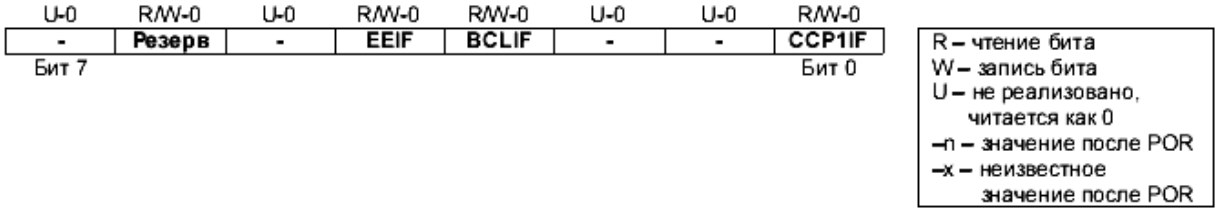


Рисунок 45 – Регістр прапорів периферійних модулів PIR2 укр!

Призначення бітів регістру PIR2:

EEIF – Прапор переривання по закінченню запису в EEPROM даних (скидається програмно).

BCLIF – Прапор переривання під час виявлення колізії в режимі ведучого на шині I^2C (скидається програмно).

CCP2IF – Прапор переривання від модуля CCP2:

а) У режимі захоплення виконане захоплення значення TMR1 (скидається програмно).

б) У режимі порівняння значення TMR1 досяглося величини, записаної в регістри CCPR2H:CCPR2L (скидається програмно).

в) У режимі ШІМ не використовується.

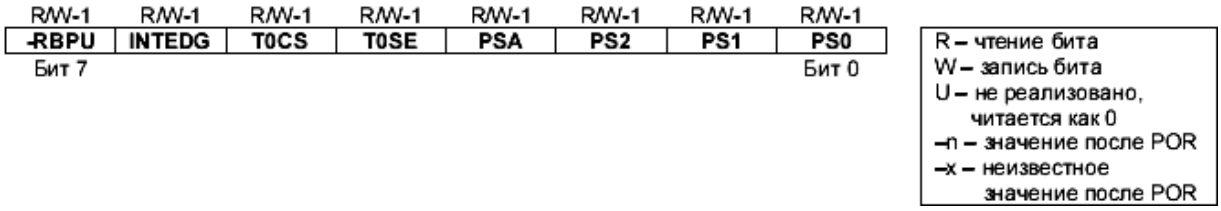


Рисунок 46 – Регістр настроювань OPTION_REG укр.!

Призначення бітів регістру OPTION_REG:

RBPU – Біт включення внутрішніх підтягуючих резисторів, на входах PORTB.

0 – підтягуючі резистори, включені.

INTEDG – Вибір активного сигналу на зовнішньому вході переривання INT:

0 – Переривання за заднім фронтом сигналу.

1 – Переривання за переднім фронтом сигналу.

T0CS – Вибір тактового сигналу для таймера TMR0:

0 – Внутрішній сигнал CLKOUT;

1 – Зовнішній сигнал з виводу RA4/T0CKI.

T0SE – Вибір фронту збільшення TMR0 при зовнішньому тактовому сигналі:

0 – Збільшення за переднім фронтом.

1 – Збільшення за заднім фронтом.

PSA – Вибір включення переддільника:

0 – Переддільник включений перед TMR0.

1 – Переддільник включений перед сторожовим таймером.

PSA2, PSA1, PSA0 – вибір коефіцієнта розподілу переддільника:

Значення:	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

Якщо переддільник підключений до WDT, то його коефіцієнт розподілу для модуля TMR0 дорівнює 1:1.

При переході на підпрограму обробки переривань у стеку апаратно зберігається тільки адреса повернення. Як правило, додатково необхідно зберігати ключові регістри (наприклад W, STATUS), що виконується програмно.

Операція збереження значення регістрів звичайно позначається PUSH, а відновлення значення регістрів позначається POP. Зверніть увагу, що PUSH, POP не є мнемонікою команд, а лише позначають дію, яка може бути виконана послідовністю команд. Слід відрізняти ці операції від команд мікропроцесора I8051, тому що мікроконтролери Picmicro мають апаратний стек, не доступний із програми. Для спрощення тексту програми можна ці сегменти коду програми подати у вигляді макросів MPASM.

Послідовність операцій при збереженні контексту програми:

1) Зберегти регістр W незалежно від поточного банку пам'яті.

2) Зберегти регістр STATUS у банку 0.

3) Виконати підпрограму обробки переривань.

4) Відновити регістр STATUS і поточний банк пам'яті даних.

5) Відновити регістр W.

Якщо необхідно зберегти й інші регістри, то збереження потрібно виконувати після збереження регістру STATUS (крок 2), а відновлення перед відновленням STATUS (крок 4).

Можна застосовувати для збереження контексту програми область пам'яті даних, яка має «дзеркальне» відображення у всіх банках регістрів – адреси 70h-7Fh.

Приклад написання макросу збереження регістрів програми:

```
PUSH_MACRO MACRO        ; Макрос збереження регістрів
MOVWF W_TEMP            ; Копіювати W у тимчасовий регістр незалежно
                        ; від поточного банку
SWAPF STATUS,W          ; Обміняти напівбайти в регістрі STATUS
                        ; і записати у W
MOVWF STATUS_TEMP       ; Зберегти STATUS у тимчасовому регістрі банку 0
ENDM                    ; Кінець макросу
;
POP_MACRO MACRO         ; Макрос відновлення регістрів
SWAPF STATUS_TEMP,W     ; Обміняти напівбайти оригінального значення STATUS
                        ; і записати у W (відновити поточний банк)
MOVWF STATUS            ; Відновити значення STATUS з регістру W
SWAPF W_TEMP,F          ; Обміняти напівбайти у регістрі W_TEMP і зберегти
                        ; результат у W_TEMP
SWAPF WJTEMP,W          ; Обміняти напівбайти в регістрі W_TEMP і
                             ; відновити оригінальне значення W без
                        ; впливу на STATUS
ENDM                    ; Кінець макросу
```

Приклад ініціалізації системи переривань:

```
PIE1_MASK1 EQU B'01101010'    ; Значення для регістру маски переривань
                              ;
      CLRF STATUS             ; Банк 0
      CLRF INTCON             ; Виключити переривання й скинути прапори
      CLRF PIR1               ; Скинути всі прапори
      BSF STATUS,RP0          ; Банк 1
      MOVLW PIE1_MASK1        ; Записати маску переривань у регістр PIE1
      MOVWF PIE1              ;
      BCF STATUS,RP0          ; Банк 0
      BSF INTCON,GIE          ; Включити переривання
```

У прикладі показана ініціалізація переривань, де PIE1_MASK – значення, записуване в регістр маски периферійних переривань.

Макрокоманди повинні бути визначені перш, ніж вони будуть використовуватися. Для простоти налагодження тексту програми макрокоманди рекомендується поміщати в окремі файли, що включаються у вихідний файл програми, до застосування макрокоманди. Рекомендується включати файли з макрокомандами на початку вихідного файла:

```
      LIST p=pl6F877          ; Список директив
      #INCLUDE <P16F877.INC>  ; Додатковий файл до мікроконтролера
      #INCLUDE <MY_STD.MAC>   ; Підключити файл стандартних макрокоманд
      #INCLUDE <APP.MAC>      ; підключити файл спеціальних макрокоманд
; Визначення бітів конфігурації для цього додатка
      _CONFIG _XT_OSC & _PWRTE_ON & _BODEN_OFF & _CP_OFF & _WDT_ON
      org 0x0000              ; Початок пам'яті програм
RESET_ADDR                    ; Перша виконувана інструкція після скидання
end
```

Як вказувалося вище, система переривань мікроконтролера PIC16F877 має 14 джерел переривань і всього лише один вектор входу до підпрограм переривань. Така організація системи переривань накладає деякі особливості при написанні програм. Визначення джерела переривання призначається програмі, яка при обслуговуванні переривань повинна визначати, який вузол мікроконтролера потребує обробки, шляхом перевірки встановлених прапорів переривань.

У нижчеподаному прикладі подана типова структура перевірки виниклого переривання. У цьому прикладі використовуються макрокоманди для збереження значення регістрів перед виконанням коду обробки переривань:

```
Org ISR_ADDR                  ;
      PUSH_MACRO              ; Макрокоманда збереження регістрів.
                              ; або інший код
      CLRF STATUS             ; Банк 0
      BTFSC PIR1,TMR1IF       ; Переривання от TMR1?
      GOTO т1_INT             ; Так
      BTFSC PIR1,ADIF         ; Ні, переривання від АЦП?
      GOTO AD_INT             ; Так, від АЦП
            ...               ; Ні, перевірка інших джерел
            ...               ; переривань
      BTFSC INTCON,RBIF       ; Ні, переривання під час зміни сигналу
                              ; на RB7:RB6?
```

```
      GOTO PORTB_INT           ; Так.
INT_ERROR_LP1                  ; Ні, процедура відновлення при помилці
      GOTO INT_ERROR_Lpl       ; Тут повинна розташовуватися процедура
                               ; обробки виникнення неочікуваного
                               ; переривання
T1_INT                         ; Обробка переривань від TMR1
            ...
      BCF PIR1,TMR1IF          ; Скидання прапора переривання від TMR1
      GOTO END_ISR             ; Завершення обробки переривань
AD_INT                         ; Обробка переривань від АЦП
            ...
      BCF PIR1,ADIF            ; Скидання прапора переривання від АЦП
      GOTO END_ISR             ; Завершення обробки переривань
PORTB_INT                      ; Обробка переривань під час зміни
            ...                ; сигналу на RB7:RB6
END_ISR
      POP_MACRO                ; Макрокоманда відновлення значення
                               ; регістрів або інший код
      RETFIE                   ; Повернення з обробки переривань,
                               ; дозвіл переривань
```

5.7 Порти введення/виводу

Універсальні порти введення/виводу можуть розглядатися як найпростіші периферійні модулі. Вони дозволяють мікроконтролерам PIC контролювати роботу й управляти іншими пристроями. Для більшості каналів портів введення/виводу регістри TRIS управляють напрямком даних на виводі. Біт TRIS<x> управляє напрямком даних на каналі PORT<x>. Якщо біт TRIS установлений в «1», то відповідний канал порту введення/виводу працює як вхід, а якщо біт TRIS скинутий в «0», то канал введення/виводу працює як вихід. Простий спосіб запам'ятати напрямок каналу введення/виводу й стан бітів регістрів TRIS: 1 – нагадує «In» (введення); «0» – нагадує «Out» (вихід).

Регістр PORT – засувка даних, виведених на порт введення/виводу. При читанні регістру PORT вертається стан виводів порту. Це означає, що необхідна деяка обережність при виконанні команд зі структурою «читання-модифікація-запис» для зміни логічного рівня на виходах порту.

Читання регістру PORT повертає стан на виводах порту, а запис виконується у вихідну засувку. Зверніть увагу на операції «читання-

модифікація-запис» (наприклад, BSF і BCF). Спочатку відбувається читання стану виводів порту, зміна отриманого значення, а потім виконується запис у вихідну засувку порту.

Розташування виводів мікроконтролера PIC16F877 можна побачити на рисунку 47. Мультипліцірування каналів введення/виводу з функціями периферійних модулів вносить свої особливості при програмуванні й настроюванні мікроконтролера.

Коли периферійний модуль підключений до виводу порту, функціональні можливості каналу порту введення/виводу можуть змінитися, відповідно до вимог периферійного модуля. Наприклад, модуль АЦП, в якому призначають відповідні канали як аналогові входи. Таким чином, виводи портів можуть бути мультиплікані з аналоговими входами й входом Vref. Для кожного виводу необхідно визначити режим його роботи (аналоговий вхід або цифровий канал введення/виводу) настроюванням керуючих бітів у регістрі ADCON1 (регістр керування АЦП). Коли вивід працює як аналоговий вхід, то читання стану цього виводу буде давати результат «0». Слід зазначити, що при скиданні мікроконтролера виводи, мультиплікані з АЦП, настроюються на аналогове введення.

Регістри TRIS управляють напрямком каналів введення/виводу, навіть коли він працює в режимі аналогового входу. Користувач повинен гарантувати, що відповідний біт TRIS установлений в «1», якщо вивід використовується як аналоговий вхід.

При включенні деяких периферійних модулів відміняється дія бітів TRIS. Тому слід уникати команд «читання-модифікація-запис» з регістрами TRIS (наприклад, BSF, BCF, XORWF і т.д.).

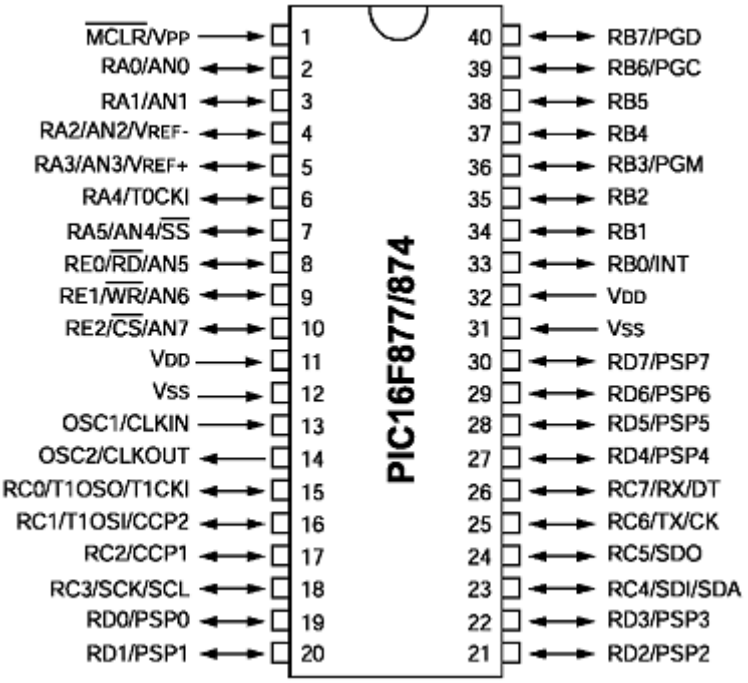


Рисунок 47 – Розташування виводів мікроконтролера PIC16F877

Регістри PORTA і TRISA

Лінія RA4 має тригер Шмідта на вході й відкритий стік на виході. Усі інші канали PORTA мають ТТЛ буфер на вході й повнофункціональні вихідні КМОП буфери. Усі виводи мають біти керування напрямку даних у регістрі TRISA, за допомогою яких можна настроїти виводи як входи або виходи.

Запис «1» у TRISA переводить відповідний вихідний буфер у Z-стан. Запис «0» у регістр TRISA визначає відповідний канал як вихід, уміст засувки PORTA передається на вивід мікроконтролера. Приклад настроювання каналів порту PORTA:

```
BCF STATUS,RP0    ; Вибрати банк 0
CLRF PORTA        ; Ініціалізація засувок PORTA
BSF STATUS,RP0    ; Вибрати банк 1
MOVLW 0xCF        ; Значення для ініціалізації
                  ; напрямку каналів PORTA
MOVWF TRISA       ; Настроїти RA<3:0> як входи.
                  ; настроїти RA<5:4> як виходи
                  ; Біти TRISA<7:6> завжди читаються як '0'.
```

Регістри PORTB і TRISB

PORTB – 8-розрядний двонаправлений порт введення/виводу. Біти регістру TRISB визначають напрямок каналів порту. Установка бита у «1» регістра TRISB переводить вихідний буфер у Z-стан. Запис «0» у регістр

TRISB настроює відповідний канал як вихід, зміст засувки PORTB передається на вивід мікроконтролера (якщо вихідна засувка підключена до виводу мікроконтролера).

Приклад настроювання порту PORTB:

```
BCF STATUS,RP0    ; ВИБРАТИ БАНК 0
CLRF PORTB        ; ІНІЦІАЛІЗАЦІЯ ЗАСУВОК PORTB
BSF STATUS,RP0    ; ВИБРАТИ БАНК 1
MOVLW 0xCF        ; ЗНАЧЕННЯ ДЛЯ ІНІЦІАЛІЗАЦІЇ
                  ; НАПРЯМКУ КАНАЛІВ PORTB
MOVWF TRISB       ; НАСТРОЇТИ RB<3:0> ЯК ВХОДИ.
                  ; RB<5:4> ЯК ВИХОДИ. RB<7:6> ЯК ВХОДИ
```

До кожного виводу PORTB підключений внутрішній підтягуючий резистор. Біт -RBPU (регістр OPTION_REG<7>) визначає, підключені (-RBPU=0) чи ні (-RBPU=1) підтягуючі резистори. Підтягуючі резистори автоматично відключаються, коли канали порту настроюються на вихід і після скидання під час включення живлення POR.

Чотири канали PORTB RB7-RB4 (рис. 47) настроєні на вхід, можуть генерувати переривання під час зміни логічного рівня сигналу на вході. Якщо один з каналів RB7-RB4 настроєний на вихід, то він не може бути джерелом переривань. Сигнал на виводах RB7-RB4 зрівнюється зі значенням, збереженим при останньому читанні PORTB. У випадку розбіжності одного зі значень установлюється прапор RBIF (INTCON<0>) і, якщо дозволене, генерується переривання.

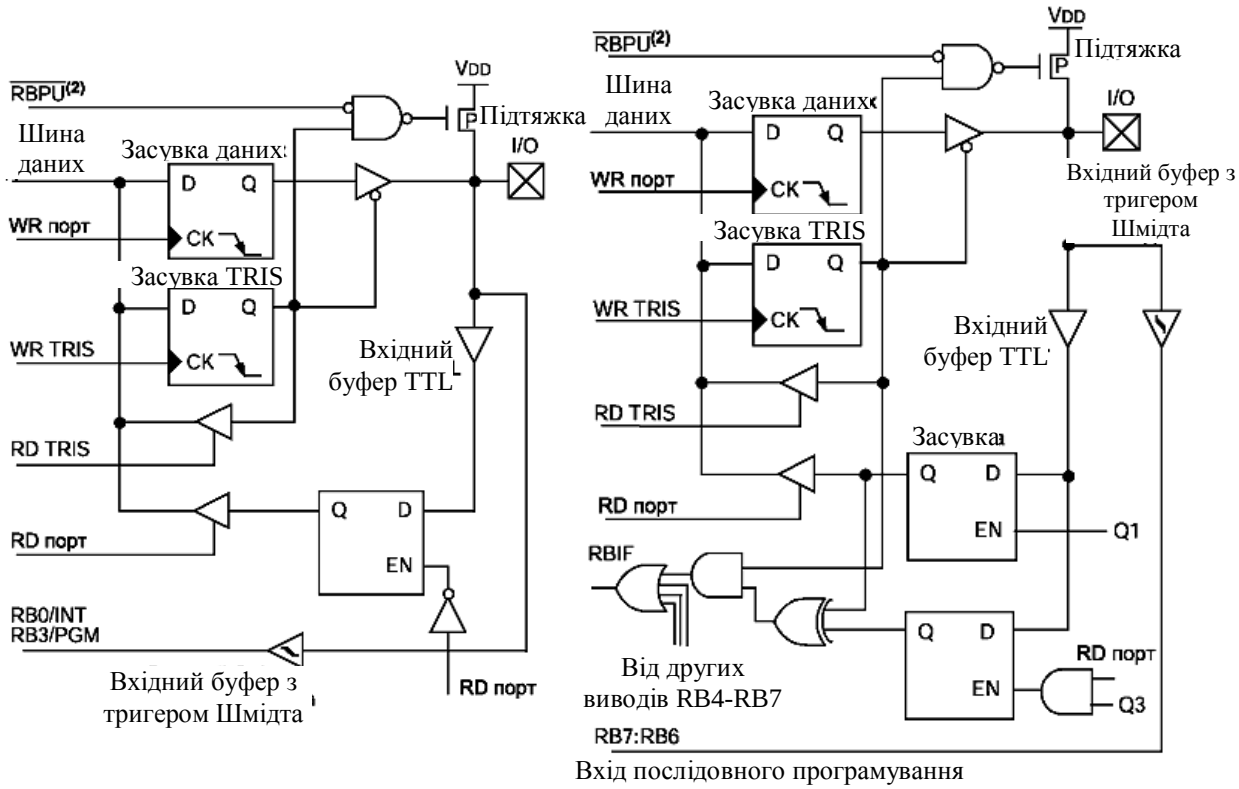


Рисунок 47 – Структурна схема виводів RB0-RB3 (а) і RB4-RB7 (б)

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 із сигналом на вході PORTB завжди встановлює біт RBIF в «1». Читання з PORTB перерве умова невідповідності й дозволить скинути прапор RBIF в «0».

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

Переривання під час зміни сигналу на входах рекомендується використовувати для визначення натискання клавіш, коли PORTB повністю задіяний для реалізації клавіатури. Не рекомендується опитувати PORTB при використанні переривань під час зміни вхідного сигналу.

Вивід RB0/INT може служити для введення зовнішнього сигналу

переривань, що настроюється бітом INTEDG регістру OPTION_REG<6>.

Регістри PORTC і TRISC

PORTC – 8-розрядний двонаправлений порт введення/виводу. Біти регістру TRISC визначають напрямок каналів порту. Установка біта в «1» регістру TRISC переводить вихідний буфер в Z-стан. Запис «0» у регістр TRISC настроює відповідний канал як вихід, уміст засувки PORTC передається на вивід мікроконтролера (якщо вихідна засувка підключена до виводу мікроконтролера).

Виводи PORTC мультиплікові з декількома периферійними модулями. На каналах PORTC є присутнім вхідний буфер із тригером Шмідта.

При використанні периферійних модулів необхідно відповідним чином ініціалізувати біти регістру TRISC для кожного виводу PORTC. Деякі периферійні модулі скасовують дію бітів TRISC, примусово призначаючи вивід як вхід або вихід. У зв'язку із чим не рекомендується використовувати команди «читання-модифікація-запис» з регістром TRISC.

Настроювання каналів PORTC проводиться аналогічними способами, зазначеними вище.

Регістри PORTD і TRISD.

PORTD – 8-розрядний двонаправлений порт введення/виводу. Біти регістру TRISD визначають напрямок каналів порту.

Канали PORTD можуть працювати як 8-розрядний мікропроцесорний порт (ведений паралельний порт), якщо біт PSPMODE (TRISE<4>) установлений в «1». У режимі веденого паралельного порту до входів підключені буфери з рівнями TTL.

Регістри PORTE і TRISE

PORTE має три виводи (RE0/-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>) установлений 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 мультиплексовані з аналоговими входами. Коли канали PORTE настроєні як аналогові входи, біти регістру TRISE не впливають на напрямок даних PORTE, читання буде давати результат «0».

Після скидання під час включення живлення виводи настроюються як аналогові входи, а читання дає результат «0».

Особливості виконання команд «читання-модифікація-запис»

Усі операції запису в порт виконуються за принципом «читання-модифікація-запис». Наприклад, команди BCF і BSF зчитують значення в регістр ЦПП, виконують бітову операцію й записують результат назад у регістр. Потрібно деяка обережність при застосуванні подібних команд до регістрів портів введення/виводу. Наприклад, команда BSF PORTB,5 зчитує всі вісім бітів PORTB у ЦПП, змінює стан біта 5 і записує результат у вихідні засувки PORTB. Якщо інший канал PORTB (наприклад, RB0) настроєний на вхід, то сигнал на виводі буде зчитаний у ЦПП й записаний у засувку даних, поверх попереднього значення. Поки RB0 настроєний як вхід, ніяких проблем не виникає. Але, якщо RB0 буде пізніше настроєний як вихід, значення в засувці даних може відрізнятися від необхідного значення.

У нижчезазначеному прикладі показаний ефект послідовного виконання команд «читання-модифікація-запис» з регістром порту введення/виводу. Початкові установки порту: PORTB<7:4> входи; PORTB<3:0> виходи. Виводи RB7-RB6 мають зовнішні підтягуючі резистори, і не підключені до інших ланцюгів у схемі:

```
                    Засувка PORTB     Виводи PORTB
BCF STATUS,RP0
BCF PORTB, 7        ;01PP PPPP        11pp PPPP
BCF PORTB,6         ;10PP PPPP        11PP PPPP
BSF STATUS,RP0      ;
BCF TRISB,7         ;10PP PPPP        11PP PPPP
BCF TRISB,6         ;10PP PPPP        10PP PPPP
```

Зверніть увагу. Можливо, користувач очікував, що після виконання програми на виходах PORTB буде значення 00PP PPPP. Однак друга команда BCF установила в «1» RB7.

Запис у порт введення/виводу фактично відбувається наприкінці машинного циклу, а читання даних виконується на початку циклу. Тому потрібно деяка обережність при записі у порт введення/виводу, якщо перед записом виконується читання стану цього порту. Послідовність команд повинна бути такою, щоб установилася напруга на виводі порту перш, ніж буде виконана команда запису в порт, супроводжувана читанням стану виводів (інакше замість нового значення може бути зчитане попереднє). Якщо можлива описана ситуація, розподіліть команди запису інструкціями NOP або будь-якими іншими командами, які не звертаються до порту введення/виводу.

При ініціалізації портів введення/виводу рекомендується спочатку записати стартове значення у вихідну засувку порту (регістр PORT), а потім настроїти напрямок каналів порту (регістр TRIS). Ця послідовність усуває можливість неправильного рівня на виході порту, тому що при включенні живлення у вихідних засувках порту втримується випадкове значення.

5.8 Модуль таймера TMR0

TMR0 – таймер/лічильник, має наступні особливості:

– 8-розрядний таймер/лічильник.

– Можливість читання й запису поточного значення лічильника.

– 8-розрядний програмувальний преддільник.

– Внутрішнє або зовнішнє джерело тактового сигналу.

– Вибір активного фронту зовнішнього тактового сигналу.

– Переривання при переповненні (перехід значення від FFh до 00h).

Блок схема модуля TMR0 і загального з WDT переддільника, показана на рисунку 48.

Коли біт T0CS скинутий в «0» (OPTION_REG<5>, рис. 46), TMR0 працює від внутрішнього тактового сигналу. Якщо біт T0CS установлений в «1» (OPTION_REG<5>), TMR0 працює від зовнішнього джерела тактового сигналу із входу RA4/T0CKI. Активний фронт зовнішнього тактового сигналу вибирається бітом T0SE у регістрі OPTION_REG<4>

(T0SE=0 – активним є передній фронт сигналу). Переддільник може бути включений перед WDT або TMR0, залежно від стану біта PSA (OPTION_REG<3>). Не можна прочитати або записати нове значення у переддільник.

Будь-який запис у регістр TMR0 викличе заборону збільшення таймера TMR0 протягом двох наступних машинних циклів ($2T_{CY}$). Якщо переддільник включений перед TMR0, то запис у регістр TMR0 викличе негайну зміну TMR0 і скидання переддільника. Збільшення TMR0 і переддільника заборонене протягом 2-х машинних циклів ($2T_{CY}$), після запису в TMR0. Наприклад, якщо коефіцієнт переддільника дорівнює 2, то після операції запису в регістр TMR0 збільшення таймера не буде відбуватися протягом 4 циклів для TMR0. Далі таймер працює в нормальному режимі.

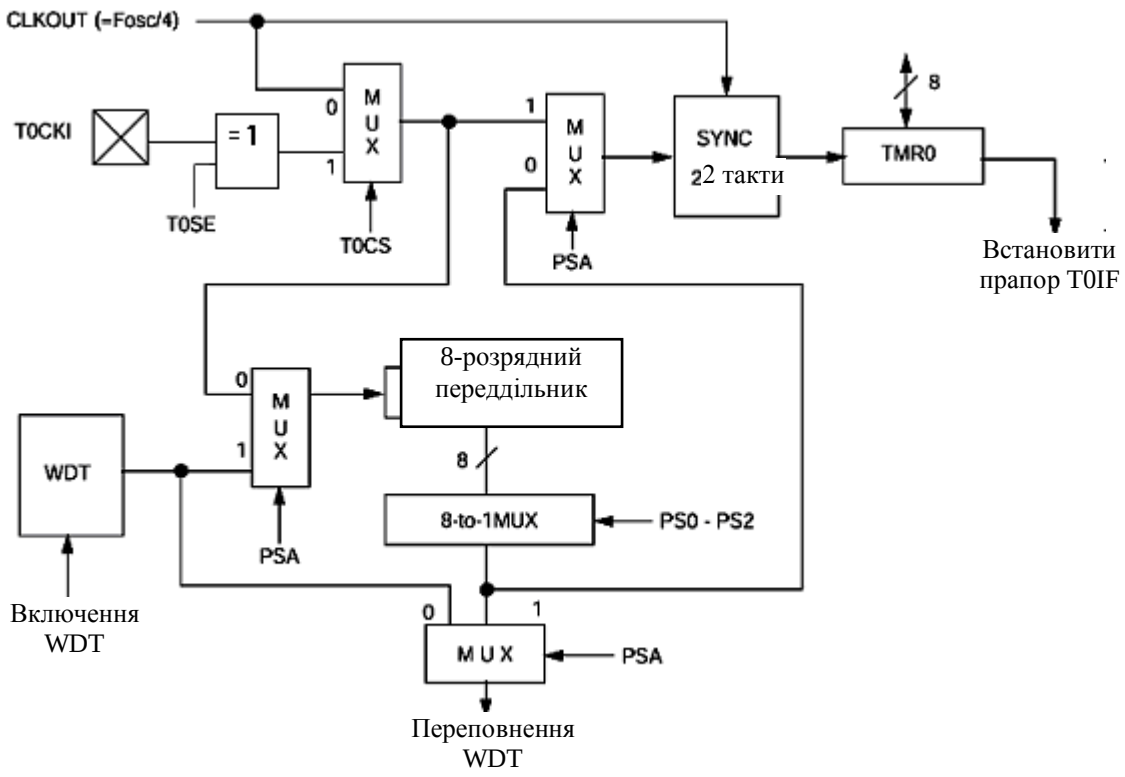


Рисунок 48 – Блок-схема модуля таймера TMR0

Переривання від TMR0 виникають при переповненні лічильника, тобто при переході його значення від FFh до 00h. При виникненні переривання встановлюється у «1» біт T0IF (INTCON<2>). Саме переривання може бути дозволене або заборонене установкою/скиданням

біта T0IE в регістрі INTCON<5>. Прапор переривання від TMR0 T0IF (INTCON<2>) повинен бути скинутий у підпрограмі обробки переривань. В SLEEP режимі мікроконтролера модуль TMR0 виключений і не може генерувати переривання.

Перемикання переддільника виконується програмним способом, тобто перемикання можна зробити під час виконання програми.

Для запобігання випадкового скидання мікроконтролера слід виконувати перемикання преддільника від TMR0 до WDT, як показано в прикладі нижче, навіть якщо WDT виключений.

У даному прикладі перша частина зміни регістру OPTION_REG не повинна виконуватися, якщо бажаний коефіцієнт переддільника відмінний від 1:1. Якщо потрібне настроювання коефіцієнта переддільника 1:1, то необхідно встановити проміжне значення коефіцієнта (відмінне від 1:1), а потім установити коефіцієнт переддільника 1:1 в останній частині зміни OPTION_REG.

Перемикання переддільника від TMR0 до WDT:

```
BSF STATUS,RP0          ;Банк 1
MOVLW Ь'XX0X0XXX'       ; ВИБРАТИ ДЖЕРЕЛО ТАКТОВОГО СИГНАЛУ И
MOVWF OPTION_REG        ; КОЕФІЦІЄНТ ПЕРЕДДІЛЬНИКА. ВІДМІННИЙ ВІД 1:1
BCF STATUS,RP0          ; БАНК 0
CLRF TMR0               ; СКИНУТИ TMR0 І ПЕРЕДДІЛЬНИК
BSF STATUS,RP0          ; БАНК 1
MOVLW B'XXXX1XXX'       ; ВКЛЮЧИТИ ПЕРЕДДІЛЬНИК ПЕРЕД WDT.
MOVWF OPTION_REG        ; АЛЕ НЕ ВИБИРАТИ КОЕФІЦІЄНТ РОЗПОДІЛУ
CLRWDT                  ; СКИНУТИ WDT І ПЕРЕДДІЛЬНИК
MOVLW B'XXXX1XXX'       ; ВИБРАТИ НОВЕ ЗНАЧЕННЯ КОЕФІЦІЄНТА
MOVWF OPTION_REG        ; ПЕРЕДДІЛЬНИКА
BCF STATUS,RP0          ; БАНК 0
```

Якщо бажане значення коефіцієнта розподілу відмінне від 1:1, то рядки 2 і 3 у текст програми не повинні включатися. Якщо потрібне настроювання коефіцієнта переддільника 1:1, то необхідно встановити проміжне значення коефіцієнта (відмінне від 1:1) у рядках 2 і 3, а потім установити коефіцієнт переддільника 1:1 у рядках 10 і 11.

Приклад перемикання переддільника від WDT до TMR0:

```
CLRWDT            ; СКИНУТИ WDT І ПЕРЕДДІЛЬНИК
BSF STATUS,RP0    ;БАНК1
MOVLW B'XXXX0XXX' ; ВКЛЮЧИТИ ПЕРЕДДІЛЬНИК ПЕРЕД TMR0 ТА
MOVWF OPTION_REG  ; ВИБРАТИ НОВЕ ЗНАЧЕННЯ КОЕФІЦІЄНТА РОЗПОДІЛУ
BCF STATUS,RP0    ; БАНК 0
```

Приклад ініціалізації TMR0 (внутрішнє джерело тактового сигналу):

```
      CLRF TMR0         ; СКИДАННЯ TMR0
      CLRF INTCON       ; ВИКЛЮЧИТИ ПЕРЕРИВАННЯ Й СКИНУТИ T0IF
      BSF STATUS,RP0    ; БАНК 1
      MOVLW 0XC3        ; ВИКЛЮЧИТИ ПІДТЯГУЮЧІ РЕЗИСТОРИ НА PORTB.
      MOVWF OPTION_REG  ; ПЕРЕРИВАННЯ ЗА ПЕРЕДНІМ ФРОНТОМ СИГНАЛУ НА
                        ; RB0
; TMR0 ІНКРЕМЕНТУЄТЬСЯ ВІД ВНУТРІШНЬОГО ТАКТОВОГО СИГНАЛУ
; ПЕРЕДДІЛЬНИК 1:16.
      BCF STATUS,RP0    ; БАНК 0
;**   BSF INTCON,T0IE   ; ДОЗВОЛИТИ ПЕРЕРИВАННЯ ВІД TMR0
;**   BSF INTCON,GIE    ; ДОЗВОЛИТИ ВСІ ПЕРЕРИВАННЯ

; ЯКЩО ПЕРЕРИВАННЯ ВІД TMR0 ВИКЛЮЧЕНІ, ТО ВИКОНУЙТЕ ПЕРЕВІРКУ БІТА
; ПЕРЕПОВНЕННЯ.
T0_OVFL_WAIT
      BTFSS INTCON,T0IF
      GOTO T0_OVFL_WAIT
; ВІДБУЛОСЯ ПЕРЕПОВНЕННЯ TMR0
```

Приклад ініціалізації TMR0 (зовнішнє джерело тактового сигналу):

```
      CLRF TMR0         ; СКИДАННЯ TMR0
      CLRF INTCON       ; ВИКЛЮЧИТИ ПЕРЕРИВАННЯ Й СКИНУТИ T0IF
      BSF STATUS,RP0    ; БАНК 1
      MOVLW 0X37        ; ВКЛЮЧИТИ ПІДТЯГУЮЧІ РЕЗИСТОРИ НА PORTB.
      MOVWF OPTION_REG  ; ПЕРЕРИВАННЯ ЗА ЗАДНІМ ФРОНТОМ СИГНАЛУ НА
                        ; RB0
; TMR0 ІНКРЕМЕНТУЄТЬСЯ ВІД ЗОВНІШНЬОГО ТАКТОВОГО СИГНАЛУ;
; ПЕРЕДДІЛЬНИК 1:256.
      BCF STATUS,RP0    ; БАНК 0
;**   BSF INTCON,T0IE   ; ДОЗВОЛИТИ ПЕРЕРИВАННЯ ВІД TMR0
;**   BSF INTCON, GIE   ; ДОЗВОЛИТИ ВСІ ПЕРЕРИВАННЯ

; ЯКЩО ПЕРЕРИВАННЯ ВІД TMR0 ВИКЛЮЧЕНІ, ТО ВИКОНУЙТЕ ПЕРЕВІРКУ БІТА
; ПЕРЕПОВНЕННЯ.
T0_OVFL_WAIT
      BTFSS INTCON, T0IF
      GOTO T0_OVFL_WAIT
; ВІДБУЛОСЯ ПЕРЕПОВНЕННЯ TMR0
```

5.9 Модуль таймера TMR1

Таймер TMR1 – 16-розрядний таймер/лічильник, що складається із двох 8-розрядних регістрів (TMR1H і TMR1L), доступних для читання й запису. Рахунок виконується в спарених регістрах (TMR1H-TMR1L),

інкрементуючи їх значення від 0000h до FFFFh, далі виконується з 0000h. При переповненні лічильника встановлюється у «1» прапор переривання TMR1IF у регістрі PIR1<0>. Саме переривання можна дозволити або заборонити установкою/скиданням біта TMR1IE у регістрі PIE1<0>. TMR1 може працювати у двох режимах:

– Режим таймера.

– Режим лічильника.

Включення модуля TMR1 здійснюється установкою біта TMR1ON в «1» у регістрі T1CON:

бітом TMR1CS (T1CON<1>) вибирається джерело тактових імпульсів (рис. 50). У режимі таймера TMR1 інкрементується на кожному машинному циклі. Якщо TMR1 працює із зовнішнім джерелом тактового сигналу, то збільшення відбувається за кожним переднім фронтом сигналу.

Таймер TMR1 має внутрішній вхід скидання від CPP модуля.

Коли включений генератор тактових імпульсів (T1OSCEN=1), виводи RC1/T1OSI/CCP2 і RC0/T1OSO/T1CKI настроєні як входи. Значення бітів TRISC<1:0> ігнорується, а читання даних із цих виводів дає результат «0».

Керуючі біти TMR1 перебувають у регістрі T1CON:

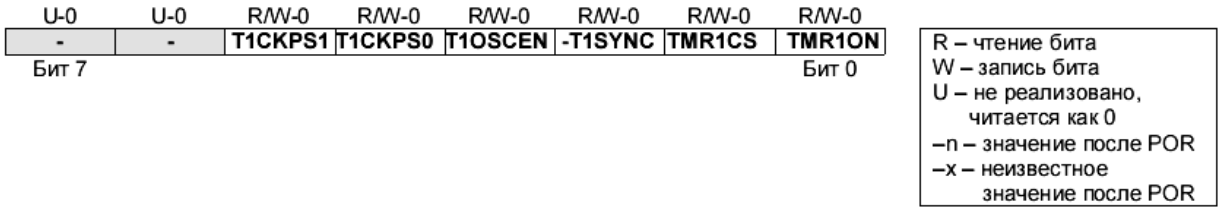


Рисунок 49 – Регістр керування таймером T1CON *укр.!*

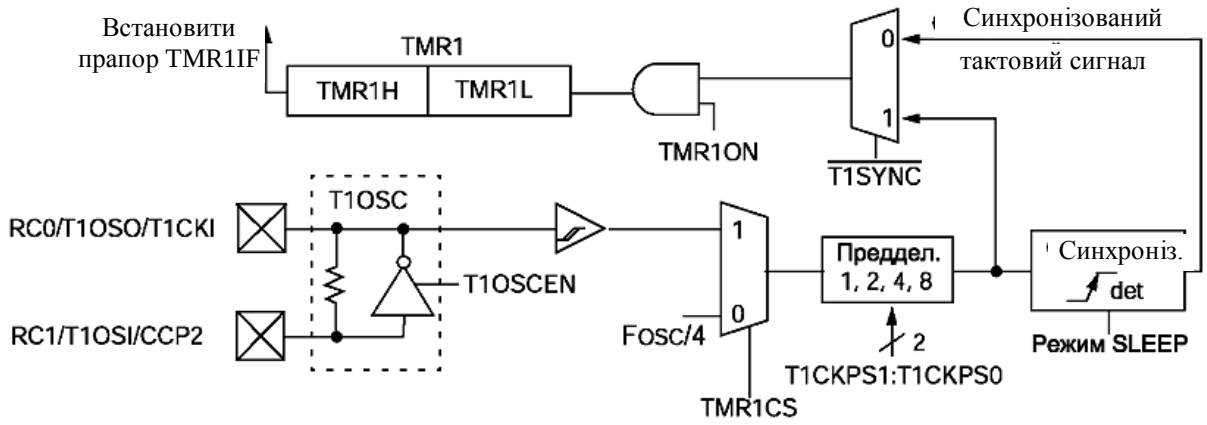


Рисунок 50 – Модуль таймера TMR1

Призначення бітів регістру T1CON:

TMR1ON – Біт включення модуля таймера.

TMR1CS – Вибір джерела тактового сигналу для збільшення таймера (внутрішній/зовнішній).

-T1SYNC – Синхронізація зовнішнього тактового сигналу.

T1OSCEN – Включення тактового генератора TMR1.

T1CKPS1, T1CKPS0 – Вибір коефіцієнта розподілу переддільника TMR1:

Значення:	Коефіцієнт:
00	1:1
01	1:2
10	1:4
11	1:8

Читання TMR1H або TMR1L. під час рахунку в асинхронному режимі, гарантує одержання поточного значення лічильника (реалізоване апаратно). Однак користувач повинен мати на увазі, що читання 16-розрядного значення виконується по одному байту. Це накладає деякі обмеження, тому що таймер може переповнитися між читаннями байт.

Запис у TMR1 рекомендується виконувати після зупинки таймера. Запис у регістри TMR1 під час збільшення таймера може призвести до непередбаченого значення регістру.

Читання 16-розрядного значення потребує деякої обережності, тому що потрібно два цикли читання для одержання всіх 16 розрядів.

У нижченаведенному прикладі подана рекомендована послідовність операцій читання 16-розрядного значення TMR1 в асинхронному режимі з вирішенням проблем переповнення. У даному прикладі таймер не зупиняється:

```
; ВИКЛЮЧИТИ ВСІ ПЕРЕРИВАННЯ
      MOVF TMR1H,W      ; ЧИТАННЯ СТАРШОГО БАЙТА
      MOVWF TMPH        ;
      MOVF TMR1L,W      ; ЧИТАННЯ МОЛОДШОГО БАЙТА
      MOVWF TMPL        ;
      MOVF TMR1H,W      ; ЧИТАННЯ СТАРШОГО БАЙТА
      SUBWF TMPH,W      ; ПОРІВНЯННЯ З ПОПЕРЕДНІМ ЧИТАННЯМ
```

```
      BTFSC STATUS,Z    ;
      GOTO CONTINUE     ; 16-РОЗРЯДНЕ ЗНАЧЕННЯ ПРОЧИТАНЕ ПРАВИЛЬНО

; МОЖЛИВО, МІЖ ЧИТАННЯМИ БАЙТІВ ВІДБУЛОСЯ
; ПЕРЕПОВНЕННЯ ТАЙМЕРА
; ПРОЧИТАТИ ЗНАЧЕННЯ ЗАНОВО
      MOVF TMR1H,W      ; ЧИТАННЯ СТАРШОГО БАЙТА
      MOVWF TMPH        ;
      MOVF TMR1L,W      ; ЧИТАННЯ МОЛОДШОГО БАЙТА
      MOVWF TMPL        ;
CONTINUE:
; ВКЛЮЧИТИ ПЕРЕРИВАННЯ (ЯКЩО НЕОБХІДНО)
```

Для запису 16-розрядного значення в регістри TMR1, спочатку потрібно очистити регістр TMR1L, щоб у запасі була велика кількість тактів TMR1 перш, ніж відбудеться перенос із молодшого регістру TMR1L в TMR1H. Виконати запис в TMR1H. а потім записати значення в TMR1L. Ця послідовність дій показана в прикладі:

```
; ВИКЛЮЧИТИ ВСІ ПЕРЕРИВАННЯ
      CLRF TMR1L        ; ОЧИСТИТИ МОЛОДШИЙ БАЙТ
                        ; ДЛЯ ЗАПОБІГАННЯ ПЕРЕНОСУ В TMRH
      MOVLW HI_BYTE     ; ЗНАЧЕННЯ ДЛЯ TMR1H
      MOVWF TMR1H       ; ЗАПИСАТИ СТАРШИЙ БАЙТ
      MOVLW L0_BYTE     ; ЗНАЧЕННЯ ДЛЯ TMR1L
      MOVWF TMRIH       ; ЗАПИСАТИ МОЛОДШИЙ БАЙТ
; ВКЛЮЧИТИ ПЕРЕРИВАННЯ (ЯКЩО НЕОБХІДНО)
CONTINUE:
```

Приклад ініціалізації TMR1 від внутрішнього тактового сигналу:

```
      CLRF T1CON        ; ВИКЛЮЧИТИ TMR1, ВНУТРІШНІЙ ТАКТОВИЙ СИГНАЛ.
; ГЕНЕРАТОР TMR1 ВИКЛЮЧЕНИЙ, ПЕРЕДДІЛЬНИК =1:1
      CLRF TMR1H        ; ОЧИСТИТИ СТАРШИЙ БАЙТ РЕГІСТРУ TMR1
      CLRF TMR1L        ; ОЧИСТИТИ МОЛОДШИЙ БАЙТ РЕГІСТРУ TMR1
      CLRF INTCON       ; ВИКЛЮЧИТИ ПЕРЕРИВАННЯ
      BSF STATUS,RP0    ;БАНК 1
      CLRF PIE1         ; ВИКЛЮЧИТИ ПЕРИФЕРІЙНІ ПЕРЕРИВАННЯ
      BCF STATUS,RP0    ; БАНК 0
      CLRF PIR1         ; ОЧИСТИТИ ПРАПОРИ ПЕРИФЕРІЙНИХ ПЕРЕРИВАНЬ
      MOVLW 0X30        ; ВНУТРІШНІЙ ТАКТОВИЙ СИГНАЛ ІЗ ПЕРЕДДІЛЬНИКОМ 1:8
      MOVWF T1CON       ; TMR1 І ГЕНЕРАТОР TMR1 ВИКЛЮЧЕНІ
      BSF TICON,TMR10N  ; ВКЛЮЧИТИ TMR1

; ПЕРЕРИВАННЯ ВІД TMR1 ВИКЛЮЧЕНІ, ПЕРЕВІРЯЙТЕ БІТ ПЕРЕПОВНЕННЯ

T1_OVFL_WAIT
      BTFSS PIR1,TMR1IF
      GOTO T1_OVFL_WAIT
```

```
; ПЕРЕПОВНЕННЯ TMR1

      BCF PIR1,TMR1IF
```

Приклад ініціалізації TMR1 від зовнішнього тактового сигналу:

```
      CLRF T1CON        ; ВИКЛЮЧИТИ TMR1, ВНУТРІШНІЙ ТАКТОВИЙ СИГНАЛ.
; ГЕНЕРАТОР TMR1 ВИКЛЮЧЕНИЙ, ПЕРЕДДІЛЬНИК =1:1
      CLRF TMR1H        ; ОЧИСТИТИ СТАРШИЙ БАЙТ РЕГІСТРУ TMR1
      CLRF TMR1L        ; ОЧИСТИТИ МОЛОДШИЙ БАЙТ РЕГІСТРУ TMR1
      CLRF INTCON       ; ВИКЛЮЧИТИ ПЕРЕРИВАННЯ
      BSF STATUS,RP0    ;БАНК 1
      CLRF PIE1         ; ВИКЛЮЧИТИ ПЕРИФЕРІЙНІ ПЕРЕРИВАННЯ
      BCF STATUS,RP0    ; БАНК 0
      CLRF PIR1         ; ОЧИСТИТИ ПРАПОРИ ПЕРИФЕРІЙНИХ ПЕРЕРИВАНЬ
      MOVLW 0X3E        ; ЗОВНІШНІЙ НЕ СИНХРОНІЗОВАНИЙ СИГНАЛ З
                        ; ПЕРЕДДІЛЬНИКОМ 1:8 І ЗОВНІШНІМ ГЕНЕРАТОРОМ
      MOVWF T1CON       ; TMR1 ВИКЛЮЧЕНИЙ
      BSF TICON,TMR10N  ; ВКЛЮЧИТИ TMR1

; ПЕРЕРИВАННЯ ВІД TMR1 ВИКЛЮЧЕНІ, ПЕРЕВІРЯЙТЕ БІТ ПЕРЕПОВНЕННЯ

T1_OVFL_WAIT
      BTFSS PIR1,TMR1IF
      GOTO T1_OVFL_WAIT

; ПЕРЕПОВНЕННЯ TMR1

      BCF PIR1,TMR1IF
```

5.10 Модуль таймера TMR2

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

Вхідний тактовий сигнал (Fosc/4) надходить через переддільник із програмованим коефіцієнтом розподілу (1:1, 1:4 або 1:16), обумовлений бітами T2CKPS1, T2CKPS0 (T2CON<1:0>).

Таймер TMR2 працює, інкрементуя своє значення від 00h до значення в регістрі PR2, потім скидається в 00h на наступному машинному

циклі. Регістр PR2 доступний для запису й читання. Після скидання значення регістру PR2 дорівнює FFh (рис. 51).

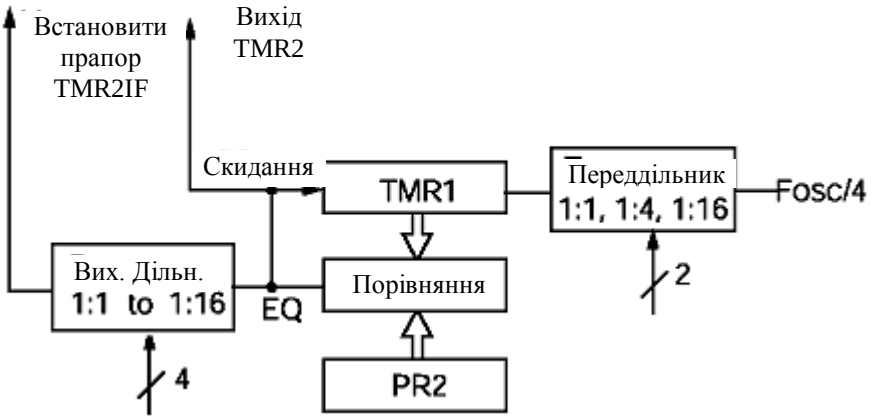


Рисунок 51 – Модуль таймера TMR2

Сигнал переповнення TMR2 проходить через вихідний 4-розрядний дільник із програмованим коефіцієнтом розподілу (від 1:1 до 1:16 включно) для установки прапора TMR2IF у регістрі PIR1<1>.

Для зменшення енергоспоживання таймер TMR2 може бути виключений скиданням біта TMR2ON (T2CON<2>) в «0»:

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
-	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
Бит 7							Бит 0

R – чтение бита
W – запись бита
U – не реализовано, читается как 0
–n – значение после POR
–x – неизвестное значение после POR

Рисунок 52 – Регістр керування таймером T2CON укр.!

Призначення бітів регістру:

T2CKPS1, T2CKPS0 – Вибір коефіцієнта переддільника TMR2:

Значення:	Коефіцієнт:
00	1:1
01	1:4
1x	1:16

TMR2ON – Біт включення таймера.

TOUTPS3-TOUTPS0 – Біти вибору коефіцієнта розподілу вихідного дільника TMR2:

Значення:	Коефіцієнт:

0000	1:1
0001	1:2
…	…
1111	1:16

Приклад ініціалізації таймера TMR2:

```
      CLRF T2C0N        ; Виключити TMR2. переддільник = 1:1,
                        ; вихідний дільник =1:1
      CLRF TMR2         ; Очистити регістр TMR2
      CLRF INTCON       ; Виключити переривання
      BSF STATUS,RP0    ; Банк 1
      CLRF PIE1         ; Виключити периферійні переривання
      BCF STATUS,RP0    ; Банк 0
      CLRF PIR1         ; Очистити прапори периферійних переривань
      MOVLW 0X72        ; переддільник = 1:15, вихідний дільник = 1:16,
      MOVWF T2CON       ; TMR2 виключений
      BSF T2CON,TMR20N  ; ВКЛЮЧИТИ TMR2

; Перериваня від TMR2 виключені, перевіряйте біт переповнення

T2_OVFL_WAIT
      BTFSS PIRI, TMR2IF; Відбулося переповнення TMR2?
      GOTO T2_OVFL_WAIT ; Ні, залишатися в циклі

; Переповнення TMR2

      BCF PIR1,TMR2IF
```

6 ТРАНСЛЯТОР АСЕМБЛЕРА МІКРОКОНТРОЛЕРІВ Picmicro MPASM

6.1 Правила написання програм

Принципи написання програм мовою асемблера для мікроконтролерів сімейства Picmicro залишаються такі ж, як і описані у главах 2 і 3 для мікроконтролерів сімейства MCS-51. Існують тільки деякі особливості:

– Необхідно враховувати апаратні тонкощі при програмуванні мікроконтролерів Microchip, описані в розділі 5 даного видання.

– Мікроконтролер має тільки 35 команд.

– Мікроконтролер має тільки 8 рівнів стека.

– Мікроконтролер має пряму, безпосередню й досить складний механізм непрямої адресації.

– Синтаксис директив транслятора трохи відрізняється від синтаксису транслятора ASM51.

– Величини числових констант записуються в такий спосіб:

H’9F’ – запис шістнадцятеричного числа.

0x9F – запис шістнадцятеричного числа.

D’10’ – десяткове число.

O’377’ – восьмеричне число.

B’11001101’ – двійкове число.

A’T’ – символ ASCII.

’R’ – символ ASCII.

6.2 Директиви асемблера MPASM

Більшу частину директив асемблера ASM51 розуміє й транслятор MPASM. Список основних директив транслятора зазначено в таблиці 10. Для більш детального опису й одержання інформації з директив зверніться до посібника користувача транслятора MPASM.

Таблиця 10 – Список основних директив

Директива	**Опис**
BANKISEL	Вибір банку РЗП для непрямої адресації
BANKSEL	Вибір банку РЗП для прямої адресації
__CONFIG	Установка бітів конфігурації
CONSTANT	Визначити символьну константу
DA	Збереження рядка в пам'яті програм
DATA	Збереження значень або тексту в пам'яті програм

DB	Побайтове збереження даних у пам'яті програм
#DEFINE	Визначає заміну тексту
DT	Визначає таблицю даних
END	Закінчення програми
ENDM	Закінчення макросу
EQU	Визначення константи асемблера

Продовження таблиці 10

EXPAND	Включення тексту макросу у файл лістингу програми
__IDLOCS	Установка значення ID
#INCLUDE	Підключення додаткового вихідного файла
LIST	Список параметрів
LOCAL	Оголосити локальну змінну макросу
MACRO	Визначити макрос
NOEXPAND	Не розвертати текст макросу
NOLIST	Виключити вивід у файл лістингу
ORG	Установити адресу програми
PROCESSOR	Вибір типу мікроконтролера
RADIX	Система числення за замовчуванням
SET	Визначення константи
#UNDEFINE	Скасувати заміну тексту

MPASM може використовуватися у двох випадках:

– Для генерації абсолютного коду, який може бути завантажений безпосередньо в мікро контролер.

– Для генерації об'єктних файлів, які зв'язуються з іншими компильованими модулями.

Абсолютний код – режим роботи програми MPASM за замовчуванням.

При компіляції вихідного файла в цьому режимі усі значення повинні бути явно зазначені у вихідному файлі або у файлах, що включаються. Якщо компіляція виконана без помилок, то буде створений HEX файл коду програми, який можна використовувати для безпосереднього програмування мікроконтролера.

Компілятор MPASM, як і ASM-51, так само має можливість генерувати об'єктні модулі, які можуть бути зв'язані один з одним з використанням лінкера MPLINK. Лінкер необхідний для остаточного формування виконуваного (абсолютного) коду. Даний метод дозволяє багаторазово використовувати налагоджені модулі програми. Об'єктні файли можуть бути згруповані в бібліотечні файли за допомогою програми MPLIB. Бібліотеки можуть вказуватися в якості параметра під час лінковки й, таким чином, у виконуваний код буде включені тільки необхідні процедури.

Для набору тексту вихідної програми може бути використаний будь-який текстовий редактор, як і в трансляторі ASM-51. Таким редактором, наприклад, як уже було зазначено раніше, є редактор «Блокнот» ОС Windows.

Типи файлів, пов'язані з асемблером MPASM:

*.asm – вихідний текст програми мовою асемблер;

*.lst – лістинг програми після трансляції;

*.err – список помилок, які виникли після компіляції;

*.hex – вихідний файл коду програми;

*.hxl – файл коду програми окремо молодших байтів коду;

*.hxh – файл коду програми окремо старших байтів коду;

*.cod – файл для відладника;

*.o – вихідний об'єктний файл модуля (програми).

Формат файла лістингу, генерованного MPASM, наступний: ім'я файла й версія, дата й час компіляції, номер сторінки виводяться на початку кожної сторінки.

Перша колонка цифр указує базову адресу коду в пам'яті. Друга колонка показує 32-розрядне значення всіх символьних змінних, створених директивами SET, EQU, VARIABLE, CONSTANT або CBLOCK. Третя колонка призначена для машинного коду, виконуваного мікроконтролером. Четверта колонка містить номер рядка відповідного вихідного файла.

Залишок рядка зарезервований для вихідного тексту, який породив машинний код.

Помилки, попередження й повідомлення вставляються між рядків вихідного коду й ставляться до наступного за текстом рядка вихідного

коду.

Таблиця символів (SYMBOL TABLE) показує всі символьні змінні, знайдені в програмі.

Карта використання пам'яті (MEMORY USAGE MAP) дає виставу про використання пам'яті в графічному виді. Символ «X» показує використану ділянку, а «-» – зазначає, що ділянка пам'яті не використовується даним об'єктом. При генерації об'єктного файла карта пам'яті не виводиться.

7 ПРАКТИКУМ З АСЕМБЛЕРА Picmicro

7.1 Вимоги до звітів

До звітів висуваються такі ж вимоги, які зазначено в підрозділі 4.1 даного видання.

Для написання програм можна використовувати мову асемблер MPASM Microchip Technology або інтегроване середовище розробки й налагодження програм Mplab IDE.

Програми рішення завдань даного розділу необхідно писати у відповідності до принципової електричної схеми демонстраційно-відлагоджувального стенда PICDEM 2 Plus.

7.2 Практичне завдання 1

Тема. Знайомство з асемблером Picmicro. Програмування цифрових портів введення/виводу.

Мета: вивчити систему команд мікроконтролера PIC16F877 середнього сімейства Picmicro. Одержати навички в програмуванні портів введення/виводу.

Таблиця 11 – Варіанти завдань до самостійної роботи

Варіант	**Завдання**
1	*2*
1	Виконати вогонь, що біжить, одного світлодіода із частотою 1Гц
2	Виконати вогонь, що біжить, двох світлодіодів із частотою 0,5Гц
3	Виконати тінь, що біжить, одного світлодіода із частотою 1Гц
4	Виконати вогонь, що біжить, 2 світлодіодів у шаховому порядку із частотою 0,5Гц

Продовження таблиці 11

1	*2*
5	Виконати включення, що накопичуються, світлодіодів із частотою перемикання 1,5Гц
6	Виконати попереміне включення світлодіодів D2, D3 і D4, D5 із частотою 1Гц
7	Виконати включення світлодіодів у послідовності D3, D5, D4, D2 із частотою 1,5Гц
8	Виконати включення світлодіодів у послідовності D4, D2, D5, D3 із частотою 0,5Гц
9	Виконати вогонь, що біжить, одного світлодіода з гасінням світлодіодів між перемиканнями частотою 0,5Гц
10	Виконати вогонь, що біжить, двох світлодіодів з гасінням світлодіодів між перемиканнями із частотою 1Гц
11	Виконати тінь, що біжить, одного світлодіода з його включенням між перемиканнями із частотою 1,5Гц
12	Виконати вогонь, що біжить, 2 світлодіодів з їхнім вимиканням між перемиканнями в шаховому порядку із частотою 0,5Гц
13	Виконати включення, що накопичуються, світлодіодів з їхнім вимиканням між перемиканнями із частотою перемикання 1Гц
14	Виконати попереміне включення світлодіодів D2, D3 і D4, D5 з їхнім вимиканням між перемиканнями із частотою 1,5Гц
15	Виконати включення світлодіодів у послідовності D3, D5, D4, D2 з їхнім вимиканням між перемиканнями із частотою 0,5Гц
16	Виконати включення світлодіодів у послідовності D4, D2, D5, D3 з їхнім вимиканням між перемиканнями із частотою 0,5Гц
17	Виконати вогонь, що біжить, одного світлодіода з реверсом на кінцевому значенні із частотою 1,5Гц

Продовження таблиці 11

1	*2*
18	Виконати вогонь, що біжить, двох світлодіодів з реверсом на кінцевім значенні із частотою 0,5Гц
19	Виконати тінь, що біжить, одного світлодіода з реверсом на кінцевім значенні із частотою 1Гц
20	Виконати запалювання світлодіодів за двійковим законом із частотою зміни комбінації 1Гц
21	Виконати вогонь, що біжить, одного світлодіода із дворазовим запалюванням у позиції із частотою 0,5Гц
22	Виконати вогонь, що біжить, двох світлодіодів із дворазовим запалюванням у позиції із частотою 0,5Гц
23	Виконати тінь, що біжить, одного світлодіода із дворазовим загасанням у позиції із частотою 1,5Гц
24	Виконати тінь, що біжить, двох світлодіодів із дворазовим загасанням у позиції із частотою 1,5Гц
25	Виконати поперемінне включення світлодіодів D2, D3 і D4, D5 із дворазовим запалюванням у позиції із частотою 1Гц
26	Виконати включення світлодіодів у послідовності D3, D5, D4, D2 із дворазовим запалюванням у позиції із частотою 1,5Гц
27	Виконати включення світлодіодів у послідовності D4, D2, D5, D3 із дворазовим запалюванням у позиції із частотою 0,5Гц
28	Виконати послідовне нагромадження, а потім убування світлодіодів в іншу сторону із частотою перемикань 0,5Гц
29	Виконати плавне запалювання світлодіодів D2 і D3 протягом 3сек., а потім різке гасіння
30	Виконати плавне гасіння світлодіодів D3 і D4 протягом 4сек., а потім їх різке запалювання

7.3 Практичне завдання 2

Тема. Таймери та система переривань мікроконтролера PIC16F877.

Мета: вивчити можливості й практично освоїти написання програм з використанням таймерів і системи переривань.

Таблиця 12 – Варіанти завдань до самостійної роботи

Варіант	**Завдання**
1	*2*
1	Виконати вогонь, що біжить, на світло діодах, використовуючи для затримки таймер і систему переривань. Частота зрушення становить 1 Гц. Кнопкою S3 змінювати напрямок, а S2 – зупиняти/запускати вогонь, що біжить
2	По натисканню кнопки S3 видати звуковий сигнал за допомогою п’єзовипромінювача P1 частотою 1000Гц. При повторному натисканні на S3 виключити сигнал. Натисканням кнопки S2 виконати зміну частоти сигналу до 500Гц. Повторне натискання на S2 виконує повернення частоти до 1000Гц
3	Виконати формування звукового сигналу п’єзовипромінювачем P1 частотою 1500Гц. При натисканні на S3 збільшувати частоту сигналів з дискретністю 250Гц. При натисканні на S2 – зменшувати з дискретністю 250Гц
4	Видати звуковий сигнал через п’єзовипромінювач P1 із частотою 1000Гц тривалістю 0,5с, потім сигнал частотою 500Гц тривалістю 0,5с. При натисканні на S3 зменшувати тривалість на 0,1с. При натисканні на S2 – збільшувати на 0,1с
5	Виконати вогонь, що біжить, на світлодіодах за допомогою натискання на кнопки. S3 зрушує вогонь уліво, S2 – вправо. При натисканні на кнопки S3, S2 виконати їх «озвучку» на п’єзовипромінювачі частотою 1000Гц тривалістю 0,2с для S3 і 500Гц тривалістю 0,2с для S2

Продовження таблиці 12

1	*2*
6	Виконати плавне запалювання світлодіодів за «трикутним» законом, використовуючи один з таймерів для завдання частоти, а інший – тривалості (ШІМ). Період імпульсів – 2с
7	Виконати плавне запалювання світлодіодів за «пилкоподібним» законом, використовуючи один з таймерів для завдання частоти, а інший – тривалості (ШІМ). Період імпульсів – 2с
8	Виконати вогонь, що біжить, на світлодіодах за допомогою натискання на кнопки. S3 зрушує вогонь уліво, S2 – вправо. При запалюванні парних світлодіодів видавати короткий звуковий сигнал на п’єзовипромінювачі P1 частотою 1500Гц, непарних – 750Гц
9	Виконати програвання музичного уривка з довільними тонами. Для завдання частоти використовувати один таймер, тривалості – інший. Для збільшення тривалості тону дозволяється використовувати додатковий розподіл у регістрах. Число тонів – не менш 5
10	По натисканню на кнопку S3 виконати вогонь, що біжить. При зміні запаленого світлодіода виконати звуковий сигнал частотою 1000Гц, тривалістю 0,2с. При досягненні останнього світлодіода видати звуковий сигнал частотою 750Гц, тривалістю 0,5с і змінити напрямок вогню, що біжить. При повторному натисканні на S3 згасити світлодіоди й видати двократний сигнал частотою 1500Гц і тривалістю 0,2с. Пауза між сигналами – 0,2с
11	Видати звуковий сигнал частотою 1000Гц тривалістю 0,5с, після цього зрушити запалений світлодіод. При натисканні на S3 виконати паузу 5с. Натисканням кнопки S2 можна перервати паузу. Якщо не натиснута жодна із кнопок – повторити процес

Продовження таблиці 12

12	Виконати вогонь, що біжить, на світлодіодах з інтервалом зрушення 5с. Допускається додатковий розподіл інтервалу часу в регістрі. При натисканні на S3 повинен горіти постійно старший світлодіод, а інші продовжують вогонь, що біжить, із інтервалом в 2с. При повторному натисканні згасити старший світлодіод і вернутися до початкових умов
13	Видавати звуковий сигнал частотою 1000Гц через п'єзовипромінювач P1. При натисканні на S2 змінити частоту до 2000Гц. При натисканні на S3 – до 500Гц. При натисканні на обидві кнопки – повернутися до частоти 1000Гц. Світлодіоди HL0, HL1, HL2 повинні вказувати поточну частоту звукового сигналу 500/1000/2000 відповідно
14	По натисканню S3 виконати тінь, що біжить, на світлодіодах. При запалюванні світлодіода подати звуковий сигнал частотою 500Гц і тривалістю 0,25с. Повторне натискання S3 зупиняє рух. Натискання S2 збільшує частоту сигналу до 1000Гц
15	По натисканню S3 видати звукові сигнали 500Гц, 750Гц і 1000Гц тривалістю 0,2с з паузами 0,5с. При натисканні на S2 збільшити тривалості сигналів до 1с. Повторне натискання S3 виключає подачу сигналів

7.4 Практичне завдання 3

Тема. Методи керування кроковими двигунами.

Мета: одержати навички в керуванні кроковими двигунами й освоїти методи табличного опису функції.

Принципова схема підключення крокового двигуна до стенда PICDEM 2 Plus:

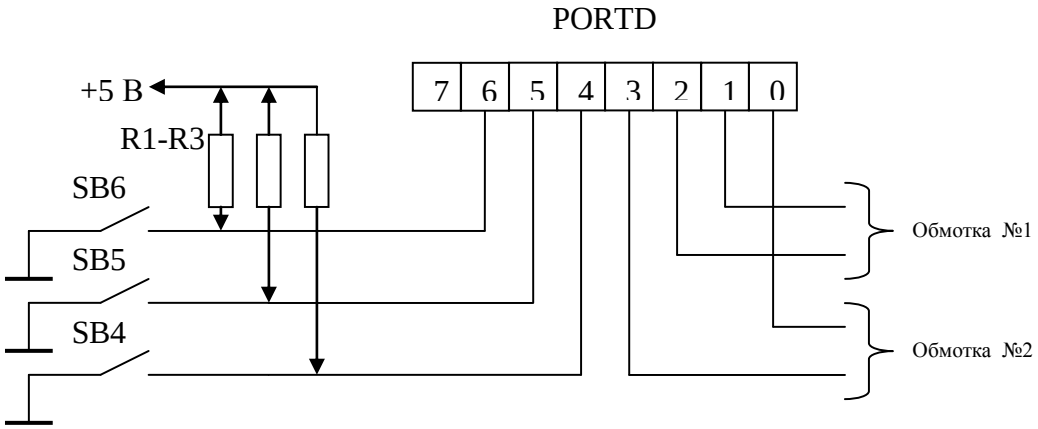


Рисунок 53 – Схема підключення модуля із кроковим двигуном

При керуванні кроковим двигуном (КД), на нього необхідно подавати послідовність керуючих імпульсів, які залежно від типу двигуна, його схеми включення й режиму роботи змінюються. Для руху двигуна в одну сторону послідовність пряма. Для руху у зворотну сторону – зворотня послідовність. Розрізняють кілька режимів роботи:

– Повнокроковий режим.

– Напівкроковий режим.

– Мікрокроковий режим.

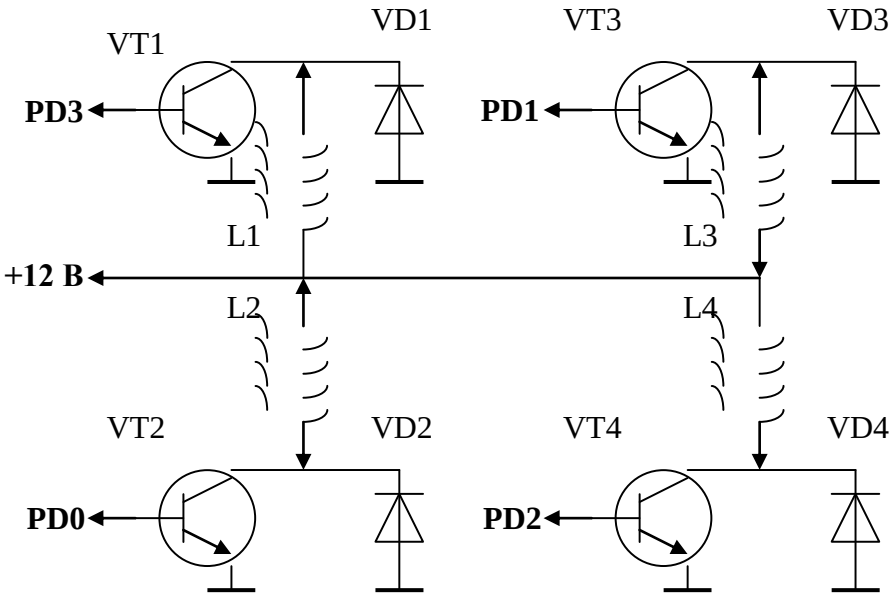


Рисунок 54 – Принципова електрична схема комутації обмоток КД

Для здійснення мікрокрокового режиму, крім керуючої послідовність комутації обмоток, виконують ще й ШІМ струму, що проходить через обмотку.

Послідовності імпульсів для обертання крокового двигуна із двома обмотками (рис. 54), що мають загальне з'єднання, у деяких режимах наведені нижче.

Послідовність комутації обмоток у **повнокроковому режимі при однофазній комутації струму**:

Номер кроку	PORTD
1	XXXX0001
2	XXXX0010
3	XXXX0100
4	XXXX1000
1	XXXX0001
Вимкн.	XXXX0000

Послідовність комутації обмоток у **повнокроковому режимі при двофазній комутації струму**:

Номер кроку	PORTD
1	XXXX0011
2	XXXX0110
3	XXXX1100
4	XXXX1001
1	XXXX0011
Вимкн.	XXXX0000

Послідовність комутації обмоток у **напівкроковому режимі при двофазній комутації струму**:

Номер кроку	PORTD
1	XXXX0001
2	XXXX0011
3	XXXX0010
4	XXXX0110
5	XXXX0100
6	XXXX1100
7	XXXX1000

8	XXXX1001
1	XXXX0001
Вимкн.	XXXX0000

При подачі керуючої послідовності на КД необхідно не забувати, що двигун має більшу електричну й механічну інерційність у порівнянні зі швидкістю роботи мікроконтролера. Це означає, що при зміні коду кроку необхідно виконати затримку часу не менш як 0,005 сек.

Таблиця 13 – Варіанти завдань до самостійної роботи

Варіант	**Завдання**
1	*2*
1	Виконати обертання КД у повнокроковому режимі при однофазній комутації струму в обмотках. При натисканні на кнопку S3 збільшувати швидкість обертання, при натисканні S2 – зменшувати. Максимальне значення швидкості обертання вказувати світлодіодом D4, мінімальне – D2, середнє – D3
2	Виконати обертання КД у повнокроковому режимі при двофазній комутації струму в обмотках. Кнопка S3 виконує старт/стоп функцію. Кнопка S2 – реверс. Світлодіоди інформують про режими: D2 – обертання в прямому напрямку, D3 – стоп, D4 – обертання у зворотному напрямку
3	Виконати обертання КД у напівкроковому режимі при двофазній комутації струму в обмотках. Після закінчення 1 с двигун змінює напрямок обертання після зупинки в 0,5 с. Кнопка S3 виконує функцію старт/стоп. Кнопка S2 змінює час роботи двигуна 1/0,5 с. Час зупинки залишається незмінним

Продовження таблиці 13

1	*2*
4	Виконати обертання КД у повнокроковому режимі при однофазній комутації струму в обмотках. Під час роботи проводиться підрахунок кількості кроків (максимум 2047). При натисканні на S3 виконується повернення ротора на кількість кроків, підрахованих до натискання даної кнопки. S2 виконує запуск двигуна з початкового положення. Світлодіоди: D4 – досягнутий максимальний крок; D3 – двигун перебуває в початковому положенні
5	Виконати обертання КД у напівкроковому режимі при двофазній комутації струму в обмотках. Кнопка S3 – крок у прямому напрямку; S2 – крок у зворотному напрямку. Номер кроку двигуна виводити на світлодіоди D3-D5 у двійковому коді
6	Виконати обертання КД у напівкроковому режимі при двофазній комутації струму в обмотках. При натисканні на кнопку S3 двигун плавно запускається протягом 3 сек. При натисканні на S2 виконує реверс із плавним гальмуванням і розгоном. Повторне натискання S3 виконує плавну зупинку
7	Виконати обертання КД у повнокроковому режимі при однофазній комутації струму в обмотках. По натисканню кнопки S3 виконати технологічний цикл: розгін протягом 2 сек., робота 5 сек., реверс 4 сек., робота у зворотному напрямку 5 сек., гальмування 2 сек. Кнопка S2 – аварійна зупинка. Світлодіоди інформують стан: D5 – прискорення; D4 – робота; D3 – гальмування
8	Виконати обертання КД у повнокроковому режимі при двофазній комутації струму в обмотках. Функція зміни швидкості – синусоїдальна. Кнопка S2 – старт/стоп

Продовження таблиці 13

1	*2*
9	Виконати обертання КД у повнокроковому режимі при однофазній комутації струму в обмотках. Двигун виконує обертання. При натисканні на кнопку S3 він робить 40 кроків у зворотному напрямку зі швидкістю в 4 рази менше за основне обертання, а потім продовжує обертання в основному напрямку. При натисканні на S2 – виконує 20 кроків зі швидкістю в 4 рази меншої в основному напрямку, а потім обертається у зворотному напрямку
10	Виконати обертання КД у повнокроковому режимі при двофазній комутації струму в обмотках. При натисканні на кнопку S2 двигун виконує крок у прямому напрямку. При досягненні 10 крока двигун починає обертатися протягом 1 сек. Потім робить зупинку і повільно виконує 10 кроків у зворотному напрямку. Світлодіоди D2-D5 вказують на поточний крок двигуна (D2 – перший… D5 – четвертий).
11	Виконати обертання КД у напівкроковому режимі при двофазній комутації струму в обмотках. Двигун виконує 20 кроків у прямому напрямку, потім 20 кроків у зворотному. Натискання кнопки S3 додає 2 кроку, S2 – зменшує на 2 кроку. Світлодіод D4 – рух у прямому напрямку; D5 – у зворотному
12	Виконати обертання КД у повнокроковому режимі при однофазній комутації струму в обмотках. По натисканню на кнопку S3 двигун повільно виконує 5 кроків з інтервалом 0,25 сек., потім обертається протягом 5 сек., виконує 5 кроків з інтервалом 0,25 сек. у зворотному напрямку й зупиняється. Натискання кнопки S2 додає число повільних кроків на 2

Продовження таблиці 13

1	*2*
13	Виконати обертання КД у повнокроковому режимі при двофазній комутації струму в обмотках. Двигун постійно перебуває в режимі реверсу. Інтервал між кроками – 0,2 сек. Утримання кнопки S3 виконує обертання двигуна в прямому напрямку з високою швидкістю, а утримання S2 – у зворотному. Світлодіоди вказують стан: D3 – повільний крок у прямому напрямку; D4 – більша швидкість обертання незалежно від напрямку; D5 – повільний крок у зворотному напрямку
14	Виконати обертання КД у напівкроковому режимі при двофазній комутації струму в обмотках. Натискання кнопки S3 виконує 1 крок двигуна в прямому напрямку. При досягненні 10 кроку, двигун починає швидко обертатися протягом 2 сек., потім зупиняється. Натискання S2 виконує ті ж дії, тільки у зворотному напрямку. Світлодіод D3 вказує натискання S3; D4 – S2
15	Виконати обертання КД у повнокроковому режимі при однофазній комутації струму в обмотках. Двигун виконує обертання протягом 1 сек., і паузу в 1 сек. циклічно. Натискання S3 зменшує паузу на 0,2 сек.; S2 – збільшує на 0,2 сек. Світлодіод D3 вказує, що двигун працює без пауз; D4 – пауза менше 2 сек.; D5 – пауза більше 2 сек.

ЛІТЕРАТУРА

1 Микропроцессорные системы: Учебное пособие для ВУЗов / Е. К. Александров, Р. И. Грушвицкий, М. С. Куприянов, О. Е. Мартынов, Д. И. Панфилов, Т. В. Ремизевич, Ю. С. Татаринов, Е. П. Угрюмов, И. И. Шагурин; Под общ. ред. Д. В. Пузанкова. – СПб.: Политехника, 2002. – 935 с.

2 Локазюк В. М. Мікропроцесори та мікроЕОМ у виробничих системах: Посібник. – К.: Академія, 2002. – 368 с.

3 Каспер Э. Программирование на языке Ассемблера для микроконтроллеров семейства i8051. – М.: Горячая линия – Телеком, 2004. – 191 с. – ISBN 5-93517-104-X.

4 Предко, М. Справочник по PIC-микроконтроллерам / М. Предко, пер с англ. – М.: ДМК Пресс, 2002, ООО Издательский дом «Додэка-XXI», 2002. – 512 с.

ДОДАТКИ
ДОДАТОК А

Система команд мікроконтролера I8051

Мікро-ЕОМ розглянутого сімейства є типовими мікропроцесорними пристроями з архітектурою CISC – зі стандартним набором команд. Тому їх система команд досить велика й містить у собі 111 основних команд. Їхня довжина – один, два або три байти, причому більшість із них (94%) – одно- або двобайтні. Усі команди виконуються за один або два машинні цикли, виключення – команди множення й ділення, які виконуються за чотири машинні цикли. Мікро-ЕОМ сімейства 8051 використовують пряму, безпосередню, непряму й неявну, адресацію даних

У якості операндів команд мікро-ЕОМ сімейства 8051 можуть використовувати окремі біти, чотирибітні цифри, байти й двобайтні слова.

Усі ці риси звичайні для набору команд будь-якого Cisc-процесора й у порівнянні з RISC набором команд забезпечує більшу компактність програмного коду й збільшення швидкодії при виконанні складних операцій.

Типи команд

Усього мікро-ЕОМ виконують 13 типів команд, вони наведені в таблиці А.1 нижче. Як випливає із неї, перший байт команди завжди містить код операції (КОП), а другий і третій (якщо вони присутні в команді) – адреси операндів або їх безпосередні значення.

Таблиця А.1 – Типи команд I8051

Тип команди	Перший байт D7...D0	Другий байт D7...D0	Третій байт D7...D0
1	*2*	*3*	*4*
Тип 1	коп		
Тип 2	коп	#d	
Тип 3	коп	ad	
Тип 4	коп	bit	
Тип 5	коп	rel	
Тип 6	коп	a7...a0	

Продовження таблиці А.1

Тип 7	коп	ad	#d
Тип 8	коп	ad	rel
Тип 9	коп	ads	add
Тип 10	коп	#d	rel
Тип 11	коп	bit	rel
Тип 12	коп	ad16h	ad16l
Тип 13	коп	#d16h	#d16l

Групи команд

Усі команди мікро-ЕОМ сімейства 8051 можна розбити на п'ять функціональних груп:

– Пересилання даних.

– Арифметичних операцій.

– Логічних операцій.

– Операцій над бітами.

– Передачі керування.

Позначення, використовувані при описі команд

Rn (n = 0, 1,..., 7) – регістр загального призначення в обраному банку регістрів;

@Ri(i = 0, 1) – регістр загального призначення в обраному банку регістрів, використовуваний у якості регістру непрямої адреси;

ad – адреса прямоадресованого байта;

ads – адреса прямо адресованого байта-джерела;

add – адреса прямо адресованого байта-одержувача;

ad11 – 11-розрядна абсолютна адреса переходу;

ad16 – 16-розрядна абсолютна адреса переходу;

rel – відносна адреса переходу;

#d – безпосередній операнд;

#d16 – безпосередній операнд (2 байта);

bit – адреса прямо адресованого біта;

/bit – інверсія прямо адресованого біта;

A – акумулятор;

PC – лічильник команд;

DPTR – регістр покажчик даних;

( ) – уміст комірки пам'яті або регістру.

Команди пересилання даних мікроконтролера 8051

У таблиці А.2 зазначені тип команди (Т) відповідно до таблиці А.1, її довжина в байтах (В) і час виконання в машинних циклах (С).

Таблиця А.2 – Команди пересилання даних

Мнемокод	КОП	Т У С	Опис
MOV A, Rn	11101rrr	1 1 1	(A) <– (Rn)
MOV A, ad	11100101	3 2 1	(A) <–(ad)
MOV A, @Ri	1110011i	1 1 1	(A) <– ((Ri))
MOV A, #d	01110100	2 2 1	(A) <– #d
MOV Rn, A	11111rrr	1 1 1	(Rn) <– (A)
MOV Rn, ad	10101rrr	3 2 2	(Rn) <– (ad)
MOV Rn, #d	01111rrr	2 2 1	(Rn) <– #d
MOV ad, A	11110101	3 2 1	(ad) <–(A)
MOV ad, Rn	10001rrr	3 2 2	(ad) <– (Rn)
MOV add, ads	10000101	9 3 2	(add) <– (ads)
MOV ad, @Ri	1000011i	3 2 2	(ad) <– ((Ri))
MOV ad, #d	01110101	7 3 2	(ad) <– #d
MOV @Ri, A	1111011i	1 1 1	((Ri)) <– (A)
MOV @Ri, ad	0110011i	3 2 2	((Ri)) <– (ad)
MOV @Ri, #d	0111011i	2 2 1	((Ri)) <– #d
MOV DPTR, #d16	10010000	3 3 2	(DPTR) <– #d16
MOVC A, @A+DPTR	10010011	1 1 2	(A) <– ((A)+(DPTR))
MOVC A, @A+pc	10000011	4 1 2	(PC) <– (PC+1), (A) <– ((A)+(PC))
MOVX A,@Ri	11100011	1 1 2	(A) <– ((Ri))
MOVX a, @DPTR	11100000	1 1 2	(A) <– ((DPTR))
MOVX @Ri, A	1111001i	1 1 2	((Ri)) <– (A)
MOVX @DPTR, A	11110000	1 1 2	(DPTR)) <– (A)

Продовження таблиці А.2

PUSH ad	11000000	3 2 2	(SP) <– (SP)+1, ((SP)) <– (ad)
POP ad	11010000	3 2 2	(ad) <– ((SP)), (SP) <– (SP)-1
XCH A, Rn	11001rrr	1 1 1	(A) <-> (Rn)
XCH A, ad	11000101	3 2 1	(A) <–>(ad)
XCH A, @Ri	11000111	1 1 1	(A) <->((@Ri))
A, @Ri	11010111	1 1 1	(A0-3)<->((@Ri0-3))

За командою MOV виконується пересилання даних із другого операнда до першого. Ця команда не має доступу ні до зовнішньої пам'яті даних, ні до пам'яті програм. Для цих цілей призначені команди MOVX та MOVC відповідно. Перша з них забезпечує читання/запис байт із зовнішньої пам'яті даних, друга – читання байт із пам'яті програм.

За командою XCH виконується обмін байтами між акумулятором і коміркою РПД, а за командою XCHD – обмін молодшими тетрадами.

Команди PUSH і POP призначені відповідно для запису даних у стек і їх читання зі стека. Розмір стека обмежений лише розміром резидентної пам'яті даних.

Група команд пересилань мікроконтролера має наступну особливість – у ній немає спеціальних команд для роботи зі спеціальними регістрами: PSW, таймером, портами вводу-виводу. Доступ до них, як і до інших регістрів спеціальних функцій, здійснюється завданням відповідної прямої адреси, тобто це команди звичайних пересилань, у яких замість адреси можна ставити назву відповідного регістру.

Крім того, слід зазначити, що в мікро-ЕОМ акумулятор має два різних імені залежно від способу адресації: A – при неявній адресації (наприклад, MOV A,R0) і ACC – при використанні прямої адреси. Перший спосіб переважніше, однак, не завжди застосуваний.

Команди арифметичних операцій

Таблиця А.3 – Команди арифметичних операцій

Мнемокод	КОП	Т	В	С	Опис
ADD A, Rn	00101rrr	1	1	1	(A)<-(A)+(Rn)
ADD A, ad	00100101	3	2	1	(A)<-(A)+(ad)
ADD A, @Ri	0010011i	1	1	1	(A)<-(A)+((Ri))
ADD A, #d	00100100	2	2	1	(A)<-(A)+#d
ADDC A, Rn	00111rrr	1	1	1	(A)<-(A)+(Rn)+(C)
ADDC A, ad	00110101	3	2	1	(A)<-(A)+(ad)+(C)
ADDC A, @Ri	0011011i	1	1	1	(A)<-(A)+((Ri))+C)
ADDC A, #d	00110100	2	2	1	(A)<-(A)+#d+(C)
DA A	11010100	1	1	1	Десяткова корекція акумулятора
SUBB A, Rn	10011rrr	1	1	1	(A)<-(A)-(Rn)-(C)
SUBB A, ad	10010101	3	2	1	(A)<-(A)-(ad)-(C)
SUBB A, @Ri	1001011i	1	1	1	(A)<-(A)-((Ri))-(C)
SUBB A, #d	10010100	2	2	1	(A)<-(A)-#d-(C)
INC A	00000100	1	1	1	(A)<-(A)+1
INC Rn	00001rrr	1	1	1	(Rn)<-(Rn)+1
INC ad	00000101	3	2	1	(ad)<-(ad)+1
INC @Ri	0000011i	1	1	1	((Ri)<-((Ri))+1
INC DPTR	10100011	1	1	2	(DPTR)<-(DPTR)+1
DEC A	00010100	1	1	1	(A)<-(A)-1
DEC Rn	00011rrr	1	1	1	(Rn)<-(Rn)-1
DEC ad	00010101	3	2	1	(ad)<-(ad)-1
DEC @Ri	0001011i	1	1	1	((Ri))<-((Ri))-1
MUL AB	10100100	1	1	4	(B)(A)<-(A)*(B)
DIV AB	10000100	1	1	4	(A).(B)<-(A)/(B)

За результатом виконання команд ADD, ADDC, SUBB, MUL і DIV установлюються прапори PSW.

Прапор C установлюється при переносі з розряду D7, тобто у випадку, якщо результат не міститься у вісім розрядів; прапор AC установлюється при переносі з розряду D3 у командах додавання й

вирахування й служить для реалізації десяткової арифметики. Ця ознака використовується командою DA A.

Прапор OV установлюється при переносі з розряду D6, тобто у випадку, якщо результат не міститься в сім розрядів і восьмий не може бути інтерпретований як знаковий. Ця ознака служить для організації обробки чисел зі знаком.

Прапор P установлюється й скидається апаратно. Якщо кількість одиничних біт в акумуляторі непарне, то P=1, а якщо ні, то P=0.

Команди логічних операцій

У цій групі 25 команд, їх короткий опис наведений у таблиці А.4. Неважко бачити, що ці команди дозволяють виконувати операції над байтами: логічне І (/\), логічне АБО (\/), що виключає АБО (+), інверсію (NOT), скидання в нульове значення й зрушення. Команди, що оперують окремими бітами, описані далі.

Таблиця А.4 – Команди логічних операцій

Мнемокод	**КОП**	**Т В З**	**Опис**
1	*2*	*3*	*4*
ANL A, Rn	01011rrr	1 1 1	(A) <– (A) /\ (Rn)
ANL A, ad	01010101	3 2 1	(A) <– (A) /\ (ad)
ANL A, @Ri	01010111	1 1 1	(A) <– (A) /\ ((Ri))
ANL A, #d	01010100	2 2 1	(A) <– (A) /\ #d
ANL ad, A	01010010	3 2 1	(ad) <– (ad) /\(A)
ANL ad, #d	01010011	7 3 2	(ad) <– (ad) /\ #d
ORL A, Rn	01001rrr	1 1 1	(A) <– (A) \/ (Rn)
ORL A, ad	01000101	3 2 1	(A) <– (A) \/ (ad)
ORL A, @Ri	0100011i	1 1 1	(A) <– (A) \/((Ri)
ORL A, #d	01000100	2 2 1	(A) <– (A) \/ #d
ORL ad, A	01000010	3 2 1	(ad) <– (ad) \/ A
ORL ad, #d	01000011	7 3 2	(ad) <– (ad) \/ #d
XRL A, Rn	01101rrr	1 1 1	(A) <– (A) (+) (Rn)
XRL A, ad	01100101	3 2 1	(A) <– (A) (+) (ad)

Продовження таблиці А.4

1	*2*	*3*	*4*
XRL A, @Ri	0110011i	1 1 1	(A) <– (A) (+) ((Ri))
XRL A, #d	01100100	2 2 1	(A) <– (A) (+) #d
XRL ad, A	01100010	3 2 1	(ad) <– (ad) (+) A
XRL ad, #d	01100011	7 3 2	(ad) <– (ad) (+) #d
CLR A	11100100	1 1 1	(A) <– 0
CPL A	11110100	1 1 1	(A) <– NOT(A)
SWAP A	11000100	1 1 1	(A0-3) <–> (A4-7)
RL A	00100011	1 1 1	Циклічне зрушення вліво
RLC A	00110011	1 1 1	Зрушення вліво через перенос
RR A	00000011	1 1 1	Циклічне зрушення вправо
RRC A	00010011	1 1 1	Зрушення вправо через перенос

Команди операцій над бітами

Група складається з 12 команд, короткий опис яких наведено в таблиці А.5. Ці команди дозволяють виконувати операції над окремими бітами: скидання, установку, інверсію біта, а також логічні І (/\) і АБО (\/). У якості «логічного» акумулятора, що використовується у всіх операціях із двома операндами, виступає ознака переносу С (розряд D7 PSW), у якості операндів можуть використовуватися 128 біт з резидентної пам'яті даних і регістри спеціальних функцій, що допускають адресацію окремих біт.

Таблиця А.5 – Команди операцій над бітами

Мнемокод	**КОП**	**Т У С**	**Опис**
1	*2*	*3*	*4*
CLR З	11000011	1 1 1	(C) <– 0
CLR bit	11000010	4 2 1	(bit) <– 0
SETB З	11010011	1 1 1	(C) <– 1
SETB bit	11010010	4 2 1	(bit) <– 1
CPL З	10110011	1 1 1	(C) <– NOT(3)

Продовження таблиці А.5

1	*2*	*3*	*4*
CPL bit	10110010	4 2 1	(bit) <– NOT (bit)
ANL З, bit	10000010	4 2 2	(C)<– (C) /\ (bit)
ANL З, /bit	10110000	4 2 2	(C) <– (C) /\ NOT(bit)
ORL З, bit	01110010	4 2 2	(C) <– (C) \/ (bit)
ORL З, /bit	10100000	4 2 2	(C) <– (C) \/ NOT(bit)
MOV З, bit	10100010	4 2 1	(C) <– (bit)
MOV bit, C	10010010	4 2 2	(bit) <– (C)

Команди передачі керування

Група подана командами безумовного й умовного переходів, командами виклику підпрограм і командами повернення з підпрограм (табл..А.6).

Таблиця А.6 – Команди передачі керування

Мнемокод	**КОП**	**Т У С**	**Опис**
LJMP ad16	00000010	12 3 2	Довгий безумовний перехід по всій пам'яті
AJMP ad11	00001	6 2 2	Безумовний перехід у межах сторінки 2 Кбайт
SJMP rel	10000000	5 2 2	Безумовний перехід у межах сторінки 256 байт
JMP @A+DPTR	01110011	1 1 2	Безумовний перехід за непрямою адресою
JZ rel	01100000	5 2 2	Перехід, якщо нуль
JNZ rel	01110000	5 2 2	Перехід, якщо не нуль
JC rel	01000000	5 2 2	Перехід, якщо біт переносу встановлений
JNC rel	01010000	5 2 2	Перехід, якщо біт переносу не встановлений
JB bit, rel	00100000	11 3 2	Перехід, якщо біт установлений
JNB bit, rel	00110000	11 3 2	Перехід, якщо біт не встановлений
JBC bit, rel	00010000	11 3 2	Перехід, якщо біт установлений зі скиданням біта
DJNZ Rn, rel	11011rrr	5 2 2	Декремент і перехід, якщо не нуль
DJNZ ad, rel	11010101	8 3 2	Декремент і перехід, якщо не нуль

Продовження таблиці А.6

CJNE: A, ad, rel	10110101	8 3 2	Порівняння акумулятора з байтом і перехід, якщо нерівно
CJNE A, #d, rel	10110100	10 3 2	Порівняння акумулятора з константою й перехід, якщо нерівно
CJNE: Rn, #d, rel	10111rrr	10 3 2	Порівняння регістру з константою й перехід, якщо нерівно
CJNE: @Ri, #d, rel	1011011i	10 3 2	Порівняння байта пам'яті з константою й перехід, якщо нерівно
LCALL ad16	00010010	12 3 2	Довгий виклик підпрограми у всій пам'яті
ACALL ad11	10001	6 2 2	Виклик підпрограми в межах сторінки 2 Кбайт
RET	00100010	1 1 2	Повернення підпрограми
RETI	00110010	1 1 2	Повернення підпрограми обробки переривання
NOP	00000000	1 1 1	Порожня операція

Команда безумовного переходу LJMP (long – довгий) здійснює перехід за абсолютною 16-бітною адресою, зазначеною в тілі команди, тобто команда забезпечує перехід у будь-яку частину пам'яті програм.

Дія команди AJMP (absolute – абсолютний) аналогічно команді LJMP, однак у тілі команди зазначені лише 11 молодших розрядів адреси. Тому перехід здійснюється у межах сторінки розміром 2 Кбайт, при цьому треба мати на увазі, що спочатку вміст лічильника команд збільшується на 2 і, тільки потім, заміняються 11 розрядів адреси.

На відміну від попередніх команд у команді SJMP (short – короткий) зазначена не абсолютна, а відносна адреса переходу. Величина зсуву rel розглядається як число зі знаком, а, отже, перехід можливий у межах -128...+127 байт щодо адреси команди, що випливає за командою SJMP.

Команда непрямого переходу JMP @A+DPTR дозволяє обчислювати адресу переходу в процесі виконання самої програми.

Командами умовного переходу можна перевіряти наступні умови:

JZ – акумулятор містить нульове значення;

JNZ – акумулятор містить не нульове значення;

JC – біт переносу C установлений;

JNC – біт переносу C невстановлений;

JB – прямо адресований біт рівний 1;

JNB – прямо адресований біт рівний 0;

JBC – прямо адресований біт дорівнює 1 і скидається в нульове значення при виконанні команди.

Усі команди умовного переходу розглянутих ОЕОМ містять коротку відносну адресу, тобто перехід може здійснюватися у межах -128... +127 байт щодо наступної команди.

Команда DJNZ призначена для організації програмних циклів. Регістр Rn або байт за адресою ad, зазначені в тілі команди, містять лічильник повторень циклу, а зсув rel – відносна адреса переходу до початку циклу. При виконанні команди вміст лічильника зменшується на 1 і перевіряється на 0. Якщо значення вмісту лічильника недорівнює 0, то здійснюється перехід на початок циклу, а якщо ні, то виконується наступна команда.

Дія команд виклику процедур повністю аналогічна дії команд безумовного переходу. Єдина відмінність полягає в тому, що вони зберігають у стеку адресу повернення.

Команда повернення з підпрограми RET відновлює зі стека значення вмісту лічильника команд, а команда повернення із процедури обробки переривання RETI, крім того, дозволяє переривання обслуженого рівня. Команди RET і RETI не розрізняють, якою командою LCALL або ACALL була викликана підпрограма, тому що й у тому, і в іншому випадку в стеку зберігається повна 16-розрядна адреса повернення.

На закінчення необхідно відзначити, що більшість трансляторів допускають узагальнену мнемоніку JMP – для команд безумовного переходу й CALL – для команд виклику підпрограм. Конкретний тип команди визначається транслятором, виходячи з «довжини» переходу або виклику.

ДОДАТОК Б

Система команд мікроконтролера PIC16F877

Довжина кожної інструкції становить 14-бітне слово, розподілене на OPCODE (частина коду команди), яка вказує на тип команди, що виконується, і один або декілька операндів, які у свою чергу надалі визначають подальше виконання інструкції.

Для опису команд сімейства Picmicro використовують наступні скорочення (табл.. Б.1).

Таблиця Б.1 – Використовувані скорочення системи команд Pictmicro

Поле	**Опис**
1	*2*
f	Адреса файлового регістру (від 0x00 до 0x7F)
W	Робочий регістр (акумулятор)
b	Адреса біта усередині 8-бітного регістру
k	Символьне поле, константа або мітка
x	Будь-яке значення(0 або 1). Компілятор згенерує код з x=0 – це потрібно для сумісності з усіма програмними продуктами Microchip
d	Вибір, де зберігати результат: d=0 (зберігати в W); d=1 (зберігати в f). За замовчуванням d=1
label	Ім'я мітки
TOS	Вершина стека
PC	Лічильник команд (програмний лічильник)
PCLATH	Записуваний буфер для старших 5 біт PC
GIE	Прапор дозволу глобальних переривань
WDT	Сторожовий таймер
$\overline{TO}$	Біт Таймауту (Time-out bit)
$\overline{PD}$	Біт зниження живлення (Power-Down bit)

Продовження таблиці Б.1

dest	Призначення, або регістр W, або інший регістр зазначений в описі команди
[ ]	Опціонально, тобто необов'язкове використання запису, який укладений у квадратні дужки
( )	Уміст
–>	Занести в
< >	Бітове поле в регістрі
∈	Належить
Курсив	Визначається користувачем

Команди підрозділяються на байт-орієнтовані, біт-орієнтовані, символьні й команди керування. На рисунку Б.1 зазначене, з яких складових частин полягає код команди залежно від її приналежності до вище певних груп.

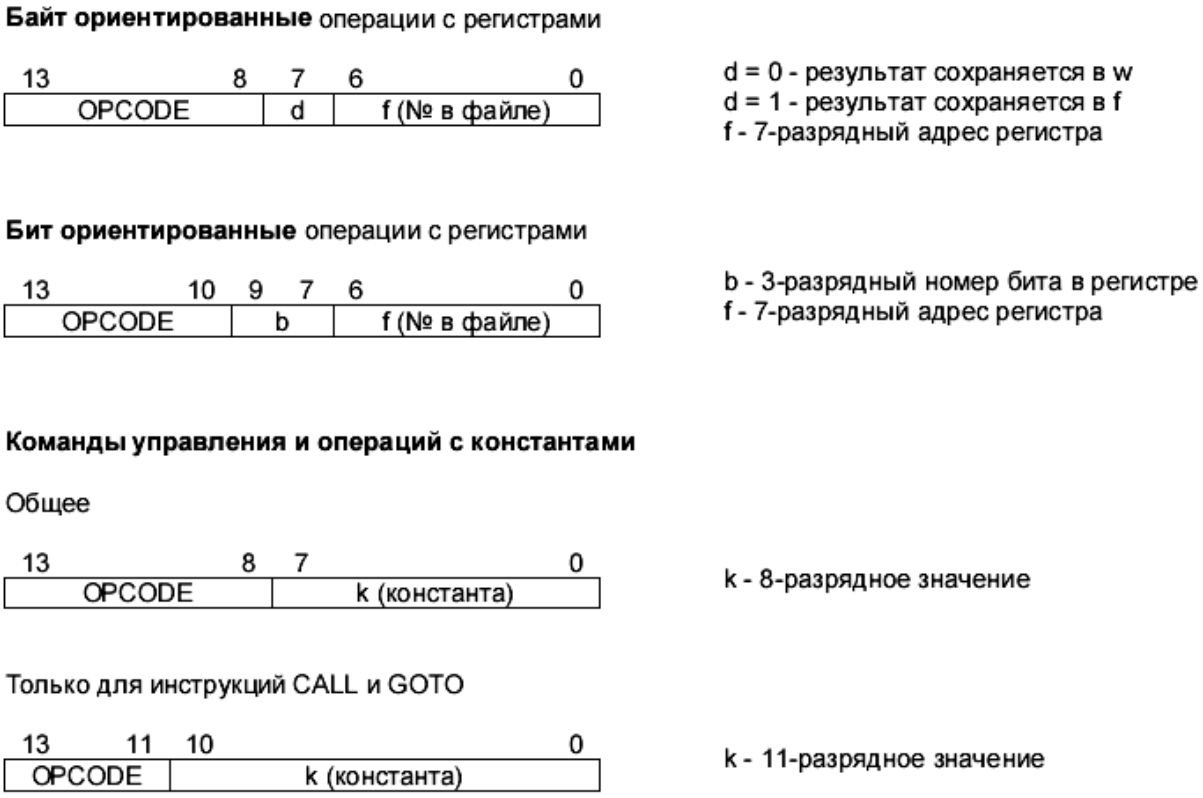


Рисунок Б.1 – Формати команд укр.!

Таблиця Б.2 – Система команд Pictmicro

Мнемоніка команди	**Опис**	**Циклів**	**14-розрядний код**		**Зм. прапори**	**Прим.**
			Біт 13	**Біт 0**		
1	*2*	*3*	*4*		*5*	*6*
БАЙТ-ОРІЄНТОВАНІ КОМАНДИ						
ADDWF f, d	Додавання W і f	1	00 0111 dfff ffff		C,DC, Z	1.2
ANDWF f, d	Побітне «І» W і f	1	00 0101 dfff ffff		Z	1.2
CLRF f	Очистити f	1	00 0001 lfff 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 і 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 в f	1	00 0000 1fff ffff			
NOP	Ні операції	1	00 0000 0xx0 0000			
RLF f, d	Циклічне зрушення f вліво через перенос	1	00 1101 dfff ffff		C	1.2
RRF f, d	Циклічне зрушення f вправо через перенос	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 напівбайти в регістрі f	1	00 1110 dfff ffff			1.2
XORWF f, d	Побитное «, що виключає АБО» W і f	1	00 0110 dfff ffff		Z	1.2
БІТ-ОРІЄНТОВАНІ КОМАНДИ						
BCF f, b	Очистити біт b у регістрі f	1	01 00bb bfff ffff			1.2
BSF f, b	Установити біт b у регістрі f	1	01 01bb bfff ffff			1.2
BTFSC f, b	Перевірити біт b у регістрі f, пропустити якщо 0	1(2)	01 10bb bfff ffff			3

Продовження таблиці Б.2

1	*2*	*3*	*4*	*5*	*6*
BTFSS f, b	Перевірити біт b у регістрі f, пропустити, якщо 1	1(2)	01 11bb bfff ffff		3
КОМАНДИ КЕРУВАННЯ І ОПЕРАЦІЇ З КОНСТАНТАМИ					
ADDLW k	Скласти константу з W	1	11 11 lx kkkk kkkk	C,DC, Z	
ANDLW k	Побітне «І» константи й 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 й 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 вW	2	11 01xx kkkk kkkk		
RETURN	Повернення з підпрограми	2	00 0000 0000 1000		
SLEEP	Перейти у 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 «АБО, що виключає» константи й W	1	11 1010 kkkk kkkk	Z	

Примітки. Для команд умовних переходів, якщо змінюється Програмний Лічильник (PC) або умова дійсна, тоді команда виконується за 2 командними циклами. Другий командний цикл виконується як команда NOP.

Навчальне видання

Мікропроцесорні пристрої

Методичні вказівки
до самостійної роботи
для студентів спеціальності 7.092203
«Електромеханічні системи автоматизації»

усіх форм навчання

НАЛИВАЙКО Олександр Михайлович
ПОНОМАРЬОВ Дмитро Сергійович

Редактор І.І.Дьякова
Комп'ютерна верстка О. П. Ордіна

131/2008. Підп. до друку . Формат 60 х 84/16.
Папір офсетний. Ум. друк. арк. Обл.-вид. арк.
Тираж прим. Зам. №

Видавець і виготівник
«Донбаська державна машинобудівна академія»
84313, м. Краматорськ, вул. Шкадінова, 72.
Свідоцтво про внесення суб'єкта видавничої справи
до Державного реєстру
серія ДК №1633 від 24.12.03.