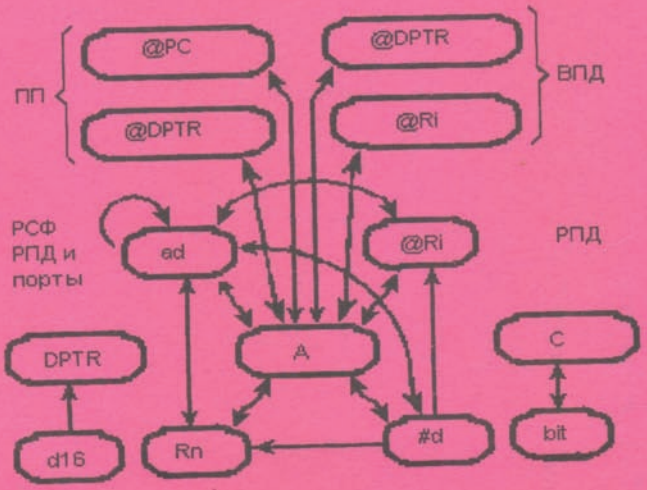
М.М. Биков, Г.Л. Лисенко, С.М.Москвіна,
В.А.Лужецький, Т.Б. Мартинюк

ОСНОВИ МІКРОПРОЦЕСОРНОЇ ТЕХНІКИ



Міністерство освіти і науки України
Вінницький національний технічний університет

М.М. Биков, Г.Л. Лисенко, С.М.Москвіна,
В.А.Лужецький, Т.Б. Мартинюк

ОСНОВИ МІКРОПРОЦЕСОРНОЇ ТЕХНІКИ

Затверджено Вченою радою Вінницького державного технічного університету як лабораторний практикум для студентів спеціальності "Комп'ютеризовані системи управління і автоматики ". Протокол № 5 від 26 грудня 2002 р.

Вінниця ВНТУ 2003

УДК 681.3
Б 60

Рецензенти:

А.М.Пєтух, доктор технічних наук, професор
В.К. Задорожний, кандидат технічних наук, доцент
Ю.А.Скидан, кандидат технічних наук, доцент

Рекомендовано до видання Вченою радою Вінницького державного технічного університету Міністерства освіти і науки України

М.М.Биков, Г.Л. Лисенко, С.М. Москвіна, В.А. Лужецький, Т.Б. Мартинюк.

Б 60 **Основи мікропроцесорної техніки.** Лабораторний практикум. - Вінниця: ВНТУ, 2003.-85 с.

Лабораторний практикум стане в нагоді студентам спеціальностей "Комп'ютеризовані системи управління і автоматики" та "Оптоелектроніка та лазерна техніка", а також для широкого кола спеціалістів, що вивчають схемотехнічні та програмні засади функціонування мікропроцесорних пристроїв в системах автоматики і оптоелектронної інформаційної техніки.

УДК 681.3

ЗМІСТ

ВСТУП

Одним з важливих досягнень мікроелектроніки та обчислювальної техніки є створення мікропроцесорів. Унікально малі розміри мікропроцесорів та створених на їх основі мікро ЕОМ, їх дешевизна, висока надійність та значні обчислювальні і логічні можливості дозволили зробити якісний стрибок в розвитку обчислювальної техніки та інформаційних технологій. В галузі автоматики та автоматизованих систем управління використання мікропроцесорів та їх різновидності – мікроконтролерів зробило можливим створення цифрових приладів і систем контролю і керування, безпосередньо вбудованих в машини, технологічні установки і процеси.

Використання мікропроцесорів і мікроконтролерів докорінно змінило традиційні методи проектування систем автоматики і автоматизованого управління, замінивши в багатьох випадках розробку схем розробкою програм налагодження мікропроцесорної апаратури на виконання певних функцій.

Проектування систем управління на базі мікропроцесорної техніки є одним із складних напрямків інженерної діяльності фахівця зі спеціальності 7.091401 "Комп'ютеризовані системи, управління і автоматика". Поряд із знаннями широкого кола питань з цифрової електроніки та програмування, які необхідно постійно обновлювати і поповнювати, для успішного проектування потрібні досвід і творчий підхід, знання внутрішньої структури мікропроцесорів і мікроконтролерів різних типів та особливостей програмування на асемблерних мовах, а також уміння на практиці використовувати наявні інструментальні засоби програмування та відлагодження асемблерних програм. Лабораторний практикум з дисципліни "Основи мікропроцесорної техніки" входить до складу комплексу навчальної літератури, що забезпечує вивчення даної дисципліни студентами спеціальності "Комп'ютеризовані системи, автоматика і управління". Практикум відіграє важливу роль для отримання практичних навичок програмування та відлагодження програм мовою Асемблер за допомогою лабораторних стендів та емуляторів.

Лабораторний практикум містить 9 лабораторних робіт, призначених для фронтального виконання на мікропроцесорних стендах типу "Електроніка 580" та емуляторі мікроконтролера MKS 51. Може бути корисним для інших спеціальностей, що вивчають мікропроцесорну техніку.

ЛАБОРАТОРНА РОБОТА №1

Вивчення будови та принципів роботи навчально-відлагоджувального пристрою "Електроніка-580"

Мета роботи - придбання навичок роботи з навчально-відлагоджувальним пристроєм "Електроніка-580 ", практичне вивчення принципів побудови мікро-ЕОМ на базі мікропроцесора КР 580 (Intel 8080).

1.1 Порядок виконання роботи

1.1.1 Ознайомитися з наведеними теоретичними відомостями і з будовою НВП "Електроніка-580".

1.1.2 Ознайомитися з режимами роботи НВП, його командними клавішами та принципами введення інформації з клавіатури.

1.1.3 Включити НВП. Для цього підключити НВП вилкою до електромережі 220 В і 50 Гц і перевести тумблер "мережа" в положення "вкл.". Натиснути клавішу **RST**, при цьому на індикаторі появиться значення 8200**??

Примітка: а) знаком "*" позначається пропуск, знаком "?" позначається якась раніше записана в ОЗП випадкова інформація.

1.1.4 Встановити перемикач "прогін-відлагодження" в положення "відлагодження".

1.1.5 Вивчити назви командних клавіш по табл.1.1. Зафіксувати та проаналізувати отримані результати.

1.1.6 Оформити звіт про виконання лабораторної роботи.

1.2 Описання лабораторного стенда

1.2.1 Органи керування мікропроцесорним НВП

Навчально-відлагоджувальний пристрій (НВП), виконаний на базі універсальної мікро-ЕОМ, призначений для створення мікропроцесорних систем різного призначення і для відлагодження їх програмного забезпечення. Крім того, він може бути використаний для навчання роботі з мікропроцесорним набором КР580.

В НВП застосований восьмирозрядний мікропроцесор (МП) типу КР580ІК80А (Intel 8080) і оперативний запам'ятовувальний пристрій (ОЗП) на інтегральній схемі (ІС) типу КР565РУ2А місткістю 2 Кбайта з адресним полем 8000_{16} - $87FF_{16}$ в шістнадцятковому коді, в залежності від об'єму вбудованого ОЗП.

Для здійснення діалогу користувача з НВП передбачені клавіатура і цифровий дисплей, дія яких забезпечується програмою-монітором, об'ємом 1 Кбайт з адресацією 0000_{16} + $03FF_{16}$, записаною в перепрограмовуваний постійний запам'ятовувальний пристрій (ППЗП) типу К573РФ2 місткістю 2кбайта.

Клавіатура містить 25 клавіш. За допомогою верхньої правої клавіші

(**RST**) формується сигнал скидання для МП, опитування інших клавіш проводиться за допомогою ІС інтерфейсу типу КР580ИК55, яка крім того, може використовуватися для реалізації операцій введення-виведення. Верхній і правий ряди клавіш містять командні клавіші НВП. Інші 16 клавіш служать для введення в НВП шістнадцяткових цифр. Назви і призначення клавіш наведені в таблиці 1.1.

Таблиця 1.1 - Призначення клавіатури НВП

Назва клавіші	Позначення клавіш	Призначення клавіш
Скидання	**RST**	Служить для формування сигналу скидання НВП
Адреса	**ADDR**	Служить для переведення НВП в режим задання адреси комірки пам'яті
Пам'ять	**MEM**	Служить для переведення НВП в режим запису
Наступний	**NEXT**	Служить для збільшення на одиницю адреси показаної комірки пам'яті чи регістра МП
Відновлення	**CLR**	Служить для відновлення початкового значення адреси чи даних, якщо після їх введення не натискали інші командні клавіші
Регістр	**REG**	Служить для відображення восьмирозрядного регістра МП
Крок	**STEP**	Служить для виконання чергової команди МП
Прогін	**RUN**	Служить для запуску програми на виконання із зупинкою на введеній контрольній точці або команді зупинки МП
Контрольна точка	**BRK**	Служить для задання адреси контрольної точки до програми

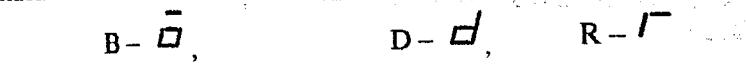
Клавіші даних використовуються також для задання імен регістрів і регістрових пар МП КР580:

- клавіші **A, B, C, D, E, 8/H, 9/L, F** - для позначення регістра-акумулятора **A** і регістрів загального призначення (РЗН) **B ÷ L** та регістра ознак **F**,
- клавіша **1/P** - для позначення покажчика стека (**SP**),
- клавіша **2/T** - для позначення вмісту вершини стека (**ST**).

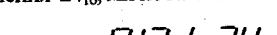
Старші розряди вершини стека зберігаються за адресою **SP+1**, молодші розряди за адресою **SP**.

Цифровий дисплей виконаний на восьми світлодіодних семисегментних індикаторах. Його дія основана на принципі прямого доступу до пам'яті. На

індикаторах відображається вміст комірок ОЗП з адресами від 8200_{16} до $83FF_{16}$. Кожний розряд є семисегментною коміркою на світлодіодах. Для відображення алфавітно-цифрової інформації, тобто цифр $0 \div 9$, літер $A \div F$, R, недостатньо семи сегментів, тому для букв B, D, R використовують стилізоване позначення:



Приклад 1.1. Під час читання вмісту комірки пам'яті з адресою $817D_{16}$, якщо там зберігається значення 24_{16}, ми побачимо на індикаторі



Приклад 1.2. Під час читання вмісту регістра A, якщо там зберігається значення CO_{16}, ми побачимо на індикаторі:

Для контролю стану тригерів (прапорців) нуля і перенесення в МП передбачені два світлодіоди "Z" і "C" відповідно.

Перемикач "Сеть" призначений для увімкнення напруги живлення НВП, а перемикач "Прогон" - "Отладка" - для виконання програм в автоматичному або покроковому режимах відповідно.

На панелі НВП також зображена таблиця шістнадцяткових кодів команд Асемблера МП КР580.

1.2.2 Основні функції монітора

Монітор дозволяє завантажити в ОЗП НВП програму користувача, переписувати її на побутовий магнітофон, прочитувати з магнітофона в ОЗП НВП, виконати програму користувача в режимі відлагодження (в некерованому режимі, або із зупинкою по заданих умовах), здійснити прогін програми користувача.

Для розширення функціональних можливостей НВП передбачена можливість підключення до нього різних зовнішніх пристроїв, наприклад, додаткової пам'яті, інтерфейсів і т.п. Додаткові плати пам'яті або інтерфейси можуть бути вбудовані також і в середину НВП, де передбачені додаткові гнізда для установки друкованої плати.

При експлуатації НВП потрібно враховувати, що час безперервної роботи не повинен перевищувати 8 годин, як правило повинен використовуватися наступний режим - безперервна робота -1,5 години, перерва (вимкнення) - 0,25÷ 0,3 години.

1.2.3 Будова та технічні характеристики МП КР580 (Intel 8080)

Структурна схема МП КР580 зображена на рис. 1.1.

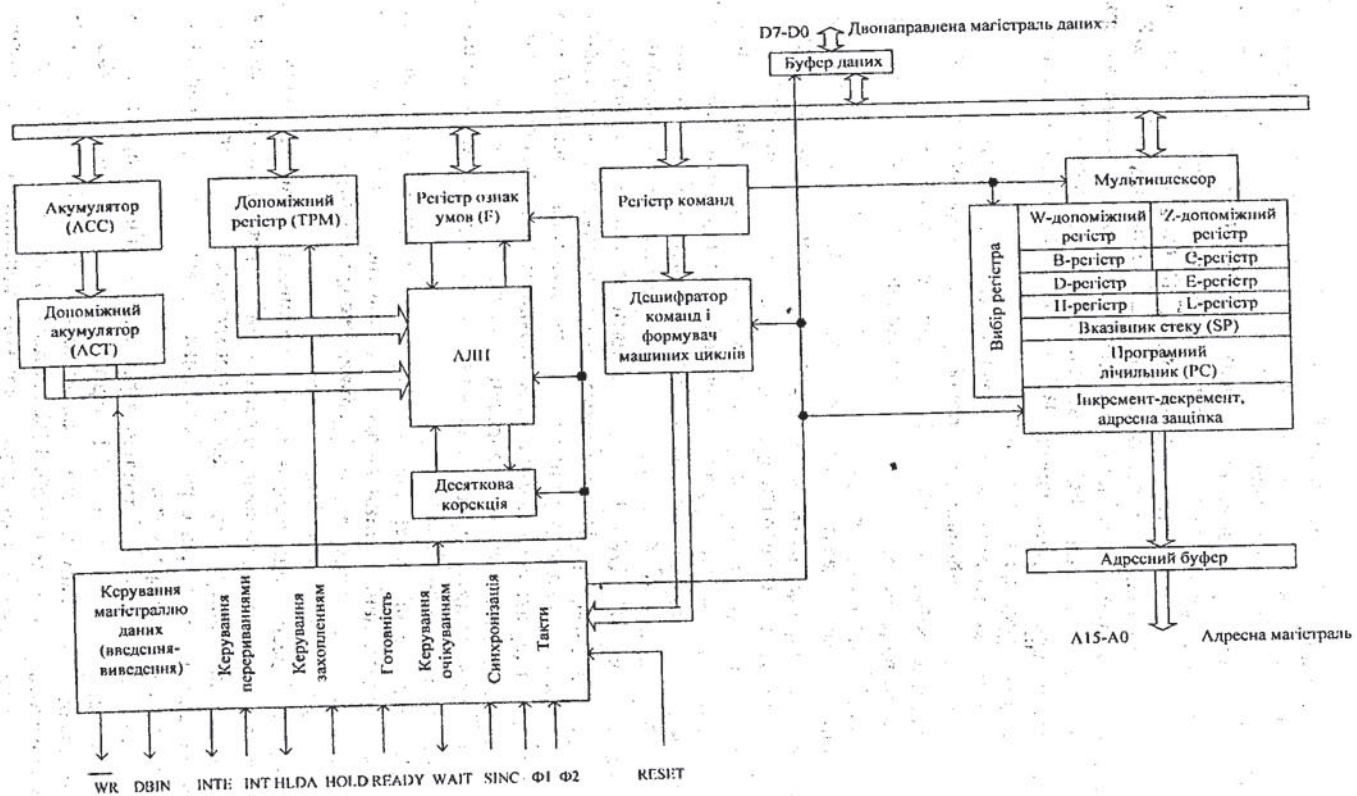


Рисунок 1.1 – Структурна схема мікропроцесора КР580 (Intel 8080)

Як видно з рис.1.1, основними складовими МП є арифметико-логічний пристрій АЛП, пристрій керування процесора у складі дешифратора команд і формувача машинних циклів, схема керування магістраллю даних, службові восьмирозрядні регістри **ТРМ, АСТ, W** і **Z**, регістр-акумулятор результатів **А (АСС)**, регістр прапорців ознак F, регістр команд, шість регістрів даних **B, C, D, E, H, L,** які можна об'єднувати в шістнадцятирозрядні регістрові пари **BC, DE, HL**, шістнадцятирозрядний регістр-покажчик стека **SP**, шістнадцятирозрядний регістр-лічильник команд **PC**, двонаправлений буфер даних і шістнадцятирозрядний буфер адреси. Всі пристрої об'єднані між собою внутрішньою восьмирозрядною магістраллю.

Тип центрального процесора - паралельний. Арифметико-логічний пристрій здатен здійснювати обробку восьмирозрядних слів, представлених в двійковому або двійково-десятковому кодах. Двійково-десяткові коди використовуються під час виконання операцій десяткової арифметики. В останньому випадку виконується корекція результатів за допомогою схеми десяткового коректора. Сигнали керування виробляються мікропрограмним пристроєм керування, який дешифрує команду, прочитану з оперативної пам'яті і записану на період її виконання в регістр команд. В регістрі команд зазвичай записується перший байт команди, який є кодом операції, другий і третій байти (для двобайтових та трибайтових команд) записуються в допоміжних регістрах **W** і **Z**. В регістрах **B, C, D, E, H, L** записуються восьмирозрядні, а регістрових парах **BC, DE, HL** – шістнадцятирозрядні дані і операнди. Крім того, регістрова пара **HL** може містити шістнадцятирозрядну адресу комірки пам'яті для команд з опосередковано-регістровою адресацією, а також служити додатковим шістнадцятирозрядним акумулятором. Окрім названого існують ще такі види адресації: неявна, пряма, безпосередня, регістрова. Адресованою одиницею пам'яті є байт, а максимальний обсяг адресованої пам'яті - 64 Кбайт.

Двонаправлений буфер даних – це схема з трьома станами, яка дозволяє вводити дані з шини даних в мікропроцесор, виводити дані з мікропроцесора на шину даних і гальванічно роз'єднувати його від магістралі в разі відсутності обміну даними.

Буфер адреси є однонаправленим і виконаний у вигляді регістра-защіпки, який видає код адреси на адресну шину на період читання даних з пам'яті чи їх запису. Код адреси формується інкрементацією або декрементацією вмісту регістра лічильника команд, яка здійснюється схемою інкремента-декремента.

Схема керування магістраллю забезпечує обмін інформацією між мікропроцесором і шиною МП системи. Сигнали **Ф1** і **Ф2** є взаємоінверсними імпульсними сигналами тактової частоти, які синхронізують роботу пристроїв МП системи. З них формується сигнал **SINC**, який видається МП на початку кожного машинного циклу. Сигнал

WAIT на виході МП говорить про те, що він знаходиться в режимі очікування обміну даними. Сигнал **READY** на вході МП говорить про готовність зовнішнього пристрою до передачі даних в МП. Сигнал **HOLD** – запит до МП від зовнішнього пристрою на захоплення шини прямого доступу до пам'яті, а сигнал **HLDA** – дозвіл від МП на захоплення шини. Сигнали **INT** і **INTE** – запит на переривання і його дозвіл відповідно. В даному МП дозволяється до восьми різних переривань. Сигнал **DBIN** від МП свідчить про те, що він приймає дані, а сигнал $\overline{\mathbf{WR}}$ - що він видає дані на шину даних.

Максимальне число внутрішніх пристроїв -180 пристроїв введення і 180 пристроїв виведення. Тактова частота f_t=2,0 мГц.

Кількість команд - 78. Система команд відповідає системі команд мікропроцесора КР580 ИК80А. Приклади команд МП:

STC – команда з неявною адресацією (встановити біт **C** в "1");

STA 8240 – команда з прямою адресацією (записати вміст акумулятора в комірку пам'яті з адресою 8240);

ADI 12 – команда з безпосередньою адресацією (додати до вмісту акумулятора операнд 12, який знаходиться безпосередньо в команді);

MOV D, E – команда з регістровою адресацією (переслати вміст регістра **E** в регістр **D**);

LXI H, 8235 – завантажити в регістрову пару адресу 8235;

MOV A, M - переслати дані з комірки пам'яті, адреса якої записана в регістровій парі **HL**, в акумулятор (команда з опосередковано-регістровою адресацією).

Введення і виведення інформації здійснюється в шістнадцятковому коді. Для тривалого збереження програм користувача передбачена можливість запису їх на побутовий магнітофон з подальшим переписуванням в ОЗП НВП.

Режими роботи:

а) покроковий;

б) прогін програми користувача в автоматичному режимі;

в) прогін програми користувача із зупинкою за заданою адресою і числом проходів.

Доступними користувачеві для використання в командах є такі регістри МП:

а) восьмирозрядний регістр-акумулятор - **A**;

б) восьмирозрядні регістри загального призначення - **B, C, D, E, H, L**;

в) регістр прапорців ознак - **F**;

г) шістнадцятирозрядні регістрові пари **BC, DE, HL**, покажчик стека **SP**, верхівка стека **ST**.

1.3 Приклад виконання роботи

Для відображення елемента пам'яті в розрядах 1÷4 індикатора в шістнадцятковій системі числення висвічується адреса, в розрядах 7÷8 - дані, що

зберігаються за цією адресою. В інших випадках в адресних розрядах (1÷4) відображається, наприклад, вміст лічильника команд, а в розрядах даних (7÷8) - чергова команда, або вміст регістра МП. В останньому випадку в п'ятому розряді індикатора з'являється найменування регістра.

Основним режимом роботи НВП є режим відлагодження програм. Програми розміщуються в ОЗП НВП в області 8000_{16} до $87FF_{16}$. Для завантаження програм в пам'ять НВП треба освоїти такі дії:

а) Читання вмісту комірок пам'яті.

Для читання вмісту комірок пам'яті з адресою **NNNN** потрібно натиснути клавіші:

ADDR NNNN

Після цього в розрядах 1÷4 індикатора відобразиться задана адреса комірки пам'яті, а в розрядах 7÷8 - її вміст.

Натискання на клавішу **NEXT** виведе на індикатор інформацію про адресу (значення, вміст) наступної комірки пам'яті. Повторне натискання на клавішу **MEM** виведе на індикатор інформацію з попередньої комірки пам'яті.

б) Читання вмісту регістрів МП.

Для читання вмісту одного з регістрів потрібно натиснути такі клавіші:

REG X,

де **X** - клавіша даних з найменуванням відповідного регістра (**A,B,C, D, E, H, L, F**).

Після натискання клавіші в розряді 5 індикатора відобразиться ім'я регістра, в розрядах 7÷8 - його вміст. Натискання на клавішу **NEXT** виведе на індикатор вміст наступного регістра МП в послідовності **A, B, C, D, E, F, H, L, A** і т.д.

Для відображення на індикаторі інформації, яка зберігається в регістрових парах R мікропроцесора, потрібно натиснути такі клавіші:

ADDR R MEM

де **R** - позначення однієї з клавіш, наведених в таблиці 1.2.

Таблиця 1.2 - Клавіші для позначення регістрових пар

Клавіша	Регістрова пара
1/P	Покажчик стека
8/H	**HL**
B	**BC**
D	**DE**
2/T	Вершина стека, **ST**

Після натискання клавіші в розрядах 5÷6 індикатора відобразиться ім'я регістрової пари, а в розрядах 1÷4 - її вміст.

в) Запис інформації в пам'яті НВП

Встановити адресу потрібного елемента пам'яті. Для цього необхідно

натиснути такі клавіші:

ADDR NNNN MEM

де N - шістнадцяткові клавіші, наприклад, **ADDR 8200 MEM**

Після цього в чотирьох лівих розрядах дисплея висвітиться адреса комірки пам'яті, а її вміст - в двох крайніх правих розрядах дисплея, а також загориться одна десяткова точка.

Якщо клавіша **MEM** була натиснена, введення даних в пам'ять здійснюється натисканням однієї або двох шістнадцяткових клавіш. Цим заміщається вміст комірки пам'яті, який висвічений на дисплеї. Нові дані з'являться на двох правих індикаторах дисплея.

Якщо під час введення даних була допущена помилка, її можна виправити натисканням додаткових цифрових клавіш. Буде записаний в пам'ять і відображений на дисплеї тільки код двох останніх цифрових клавіш. Натискання клавіші **CLR** відновлює початковий вміст комірки пам'яті (за умови, що інші командні клавіші перед цим не натискалися).

Для переходу до адреси наступної комірки пам'яті треба натиснути клавішу **NEXT**. При цьому немає необхідності натискати клавішу ще раз.

Повторне натискання клавіші **MEM** зменшить на одиницю адресу комірки пам'яті.

Висвічування десяткової точки шостого зліва розряду індикатора вказує на те, що клавіша **MEM** була натиснена і введення даних в пам'ять дозволено. Якщо вона не світиться - дані вводитися не будуть.

При спробі ввести дані без попереднього натискання клавіші **MEM**, а також, якщо на дисплеї встановлена адреса ПЗП або фактично відсутня в ОЗП, то на дисплеї висвітиться сигнал помилки "**Err**". В цьому випадку для того, щоб відновити попередню адресу і дозволити введення даних в пам'ять, необхідно натиснути клавішу **MEM**,

Приклад. В комірку 8210_{16} записати число $1A_{16}$, збільшити адресу, зменшити адресу (табл. 1.3).

Таблиця 1.3 - Приклад запису інформації з клавіатури

Клавіші	Інформація на дисплеї	Примітки
ADDR 8210	8210**,??	Установка адреси
MEM	8210**, ??	Дозволити введення даних
1	8210**, 01	Запис
A	8210**, 1A	числа 1A
NEXT	8211**,??	Наступна адреса
NEXT	8212**, ??	Наступна адреса
MEM	8211**,??	Вихідна адреса
MEM	8210**, 1A	Попередня адреса

г) Запис інформації в регістри МП.

Вибрати потрібний регістр, натиснувши клавішу **REG X,** і для введення даних в регістр натиснути одну або дві шістнадцяткові клавіші даних.

д) Введення контрольних точок.

Монітор НВП надає можливість виконання програми користувача з введенням контрольних точок, тобто адрес, на яких необхідно перервати виконання програми для перевірки проміжних результатів.

Якщо були введені контрольні точки, то під час виконання програм в режимі з зупинкою на контрольних точках перевіряються такі умови:

- змінився чи ні вміст комірки пам'яті, що адресується будь-якою контрольною точкою;
- чи відповідає вміст лічильника команд будь-якій контрольній точці.

Якщо жодна з цих умов не виконується, то продовжується виконання програми, інакше монітор зменшує на одиницю вміст числа проходів даної контрольної точки, і якщо він дорівнює нулю, то відбувається зупинка програми користувача і викликаються підпрограми клавіатури і дисплея.

Примітки:

1. Найбільше число проходів контрольної точки до зупинки дорівнює $FF_{16}=256_{16}$.

2. Програма зупиняється до виконання команди, що адресується контрольною точкою, але після зміни вмісту комірки пам'яті, що адресується цією точкою.

3. До виконання команди, яка йде за контрольною точкою, контроль інших контрольних точок не проводиться.

Введення контрольної точки за адресою NNNN з числом проходів NN здійснюється натисканням таких клавіш: **ADDR NNNN BRK NN.**

Після цього в розрядах 5-6 індикатора відображається символ контрольної точки, в розрядах 1÷4 - її адреса, в розрядах 7÷8 - число проходів. Клавіша **CLR** виключає дану контрольну точку (табл. 1.4)

Таблиця 1.4 - Приклад установки контрольних точок

Клавіші	Інформація на дисплеї	Примітки
ADDR 8220 BRK	8220 BR, 00	Початковий стан
10	8220 BR, 10	Установка нового числа переходів
CLR	8220 BR, 01	Виключення контрольної точки за адресою 8220, індикація наступної контрольної точки
CLR	****BR,**	Виключення контрольної точки за адресою 8320, інших контрольних точок немає

Послідовним натисканням на клавішу **NEXT** можна проглянути всі контрольні точки по колу з переходом від останньої до першої. Під час появи на індикаторі інформації про контрольну точку її можна змінити або виключити. Натискання на клавішу **RST** виключає всі контрольні точки. Є можливість припинити виконання програм не тільки за адресою команди, але і після виконання заданої кількості команд, для чого необхідно в комірку пам'яті з адресою 83Е6 записати число команд **NN**, яке потрібно виконати. Для цього необхідно натиснути такі клавіші:

ADDR 83E6 BRK NN

Індикація помилок.

Введення даних в пам'ять НВП дозволяється тільки в тому випадку, коли в розряді 6 індикатора висвічується точка (після натискання клавіші **MEM**). Якщо вона не світиться, дані вводитися не будуть!

Якщо під час введення даних допущена помилка, її можна виправити натисканням клавіші **CLR**, яка відновлює початковий вміст комірки пам'яті (за умови, що інші командні клавіші після цифрових не натискалися). Під час невірних дій на індикаторі з'явиться код помилки:

Err * * * * *

Він висвічується в таких випадках:

- під час спроби записати в неіснуючу комірку ОЗП або в ПЗП, а також якщо була блокована можливість введення даних в пам'ять (не натиснута клавіша **MEM**);
- під час спроби встановити неіснуюче найменування регістра:
- під час спроби встановити на місці **N** символу відмінний від символу регістрових пар **B, D, H, P, T** для операції **ADDR N MEM**;
- під час операції **ADDR Y BRK**, якщо на місці **Y** не символ регістрової пари (як в попередньому випадку) або не нуль;
- під час спроби ввести дані в лічильник проходів неіснуючої контрольної точки;
- якщо перед натисканням цифрової клавіші не була натиснута одна з клавіш: **ADDR, MEM, REG, BRK**;
- під час спроби запустити програму на виконання клавішами або **RUN**, якщо введено менше чотирьох цифр адреси (після клавіші **ADDR**). Якщо з'явився символ **Err**, натисканням **CLR** або **ADDR** можна відновити попередній стан лічильника команд і саму команду. Натисканням клавіші **MEM** відновлюється попереднє значення комірки пам'яті і її адреси.

1.4 Контрольні запитання та завдання

1. Для чого призначений навчально-відлагоджувальний пристрій (НВП)?
2. Які види адресації в пам'ять ви знаєте, які існують в даному НВП? Що є мінімальною адресованою одиницею пам'яті?

3. Які існують режими роботи НВП?
4. Назвіть регістри МП, що є доступними користувачеві.
5. Охарактеризуйте цифровий дисплей (виконання, принцип дії).
6. Для чого передбачені світлодіоди "Z" і "C"?
7. Перерахуйте основні функції монітора.
8. Поясніть призначення командних клавіш.
9. Поясніть призначення клавіш даних.
10. Розкажіть про структуру відображення інформації на індикаторі адреси і даних.
11. Для чого служить введення контрольних точок?
12. У яких випадках на індикаторі з'явиться код помилки?
13. Прочитайте вміст комірок пам'яті за адресами 5656, 820E, 86FC, 0001.
14. Використовуючи клавіші **NEXT**, **MEM**, виведіть на індикатор вміст комірок пам'яті в послідовності: 8200, 8201, 8202, 8200, 81FF.
15. Виведіть на індикатор вміст регістрів A, B, C, H, L, використовуючи клавішу NEXT.
16. Прочитайте вміст регістрової пари **HL** і вершини стека.
17. В комірки з адресами $820E_{16}$, $837A_{16}$, $837B_{16}$, $837D_{16}$ запишіть відповідно дані FC_{16}, 31_{16}, $D8_{16}$, 03_{16}.
18. Здійсніть введення даних в регістри МП: в регістр **A** – 31_{16}, **B** – CE_{16}, **E** – 84_{16};.
19. Введіть контрольні точки за адресами 8210, 8220, 8230 з числом проходів відповідно 01_{16}, 02_{16}, 10_{16}.
20. Використовуючи клавішу **NEXT** прогляньте всі контрольні точки по колу. Виключіть контрольну точку за адресою 8220.

ЛАБОРАТОРНА РОБОТА №2
Введення програм у пам'ять НВП і їх виконання у різних режимах

Мета роботи – навчитися складати програми, використовуючи систему команд МП, записувати їх в пам'ять НВП та виконувати в різних режимах.

2.1 Порядок виконання роботи

2.1.1 Ознайомитися з теоретичними відомостями, наведеними в даних методичних вказівках.

2.1.2 Для заданого викладачем варіанта задачі скласти програму в мнемокодах мови Асемблер, записати її в машинних кодах та задокументувати.

2.1.3 Ввести програму в пам'ять НВП "Електроніка 580" та виконати в покроковому режимі.

2.1.4 Зафіксувати та проаналізувати отримані результати.

2.1.5 Оформити звіт про виконання лабораторної роботи.

2.2 Теоретичні відомості

2.2.1 Відомості про роботу програми-монітора

Для виконання основних програм монітора використовується система переривань. Вона автоматично спрацьовує під час виконання кожної команди користувача, якщо перемикач вибору режиму встановлений в положенні "отладка".

Користувач має можливість викликати монітор, залучивши в свою програму команду **RST 32($E7_{16}$)**, а також впливати на функцію монітора, змінюючи адреси та прапорці вимог за допомогою своєї програми.

Доступ до програм монітора здійснюється звичайним їх викликом(**CALL**).

Коли монітор вводиться по перериванню (**RST 56**) або програмним викликом (**RST 32**), вміст регістра, лічильника команд і покажчика стека користувача зберігається в ОЗП і до нього можливий доступ за командами монітора.

Натискання на клавішу **RST** приводить до формування сигналу скидання. При цьому зазвичай втрачається вміст регістрів і стека користувача. В лічильнику команд користувача записується адреса 9200_{16}, а в покажчику стека користувача – $83E0_{16}$. За всіма адресами дозволяються переривання, а інформація про контрольні точки знищується. При цьому деякі дані, наприклад, вміст лічильника команд користувача, дістаються зі стека і зберігаються в комірці пам'яті **PCADR**. Вміст регістрів МП заноситься в стек. Якщо була використана клавіша

STEP чи сталася зупинка в контрольній точці, стан прапорців нуля і перенесення показується двома спеціальними світлодіодами.

Монітор записує такі дані у фіксованих комірках ОЗП:

BKPCW $83E2_{16}$ – однобайтовий покажчик контрольної точки, введеної раніше за інші. Якщо такі точки не вводились, він містить код $E2_{16}$.

BKPOS ($83E3_{16}$) – покажчик останньої контрольної точки.

MADDR (83E4, $E5_{16}$) – адреса останньої комірки пам'яті, до якої мало місце звертання натиском клавіші **MEM** чи **NEXT**.

PCADDR(83E6, $E7_{16}$) – лічильник команд (ЛК) користувача.

SFLAG($83F6_{16}$) – байт прапорців. Якщо програма запущена клавішею **RUN**, його вміст дорівнює нулю і перевіряються точки зупинок. Якщо він не дорівнює нулю, то програма була запущена клавішею **STEP** і під час виклику монітора розблоковується клавіатура та дисплей.

RGNAM($83F7_{16}$) – символ регістра, введений клавішею **REG**.

Якщо вміст цієї комірки пам'яті не дорівнює нулю, під час виклику програми-монітора, вона показує найменування регістра. Під час натискання клавіші **MEM** комірка **RGNAM** заповнюється нулями.

Коли програма-монітор очікує подання команди або даних, у парі регістрів **HL** зазвичай зберігається показана дисплеєм адреса (лічильника команд (ЛК) користувача), адреса точки зупинки або адреса, яка вводиться після клавіші **ADDR**.

Здебільшого дії монітора обмежені звертанням до комірок **PCADDR**, **MADDR** і зберіганням висвітленої індикатором адреси у парі регістрів **HL**.

2.2.2 Принципи програмування мовою Асемблер

Програмування на мові Асемблера, на відміну від програмування на алгоритмічних мовах, має низку особливостей. Перша з них – це необхідність знати і враховувати структуру і принцип взаємодії програмно доступних елементів конкретного мікропроцесора. Друга – необхідність знати способи адресації, які використовуються в системі його команд. Третя – необхідність самому програмісту турбуватись про розподіл пам'яті для розміщення команд і операндів. І, нарешті, четвертою особливістю є високий ступінь деталізації алгоритму – кожний крок алгоритму повинен відповідати одній команді Асемблера.

Процедура програмування задачі складається з таких етапів:

1. Розробка схеми алгоритму розв'язання задачі.
2. Запис програми в командах Асемблера.
3. Запис або трансляція програми в машинних кодах.
4. Введення програми в пам'ять мікропроцесорної (МП) системи.
5. Розв'язання програми.
6. Аналіз результатів і відлагодження програми в разі необхідності.
7. Документування програми.

За наявності програми-транслятора з Асемблера (наприклад Masm або Tasm) кроки 3 і 4 міняються місцями, тобто програма в кодах Асемблера вводиться в пам'ять МП системи, після чого транслятор автоматично генерує об'єктний модуль в машинних кодах. За відсутності транслятора програміст сам кодує програму, а потім вводить її в пам'ять ЕОМ.

Розглянемо вказані особливості програмування на прикладі задачі пошуку максимального елемента MAX в масиві C[N], де N – число елементів в масиві. Алгоритм розв'язання задачі, орієнтований на алгоритмічну мову Турбо Паскаль, зображений на рис. 2.1.

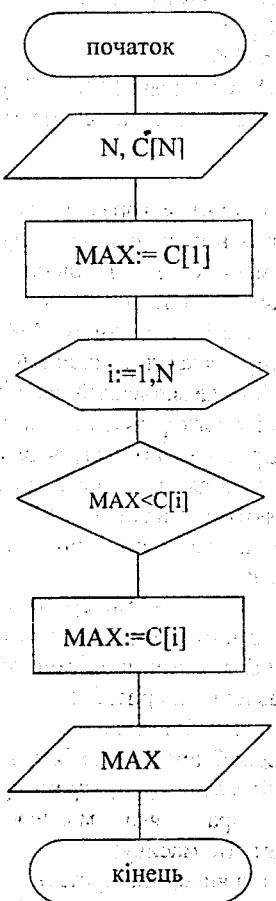


Рисунок 2.1- Схема алгоритму пошуку максимального елемента масиву, орієнтована на алгоритмічну мову

Програма на алгоритмічній мові, яка реалізує даний алгоритм, має такий вигляд:

```
program MAXMAS;
    const N=100;
    var
        I: integer;
      MAX: real;
        C: array [1..N] of real;
    begin
      writeln ('Введіть масив C[N]');
      for i:=1 to N do read (C[i]);
      MAX:= C[1];
      for i:=1 to N do
        if MAX<C[i] then MAX:= C[i];
      writeln ('Максимальний елемент Cmax =', MAX)
    end.
```

Як видно з блок-схеми алгоритму та наведеної програми, ідея розв'язання задачі полягає в послідовному перегляді масиву і порівнянні чергового елемента масиву із найденим на поточний момент максимальним елементом, який зберігається в комірці пам'яті під назвою MAX. Якщо черговий елемент більший MAX, то він заміщає його. Спочатку максимальним вважається перший елемент масиву. В кінці перегляду в комірці MAX буде записаний найбільший елемент масиву. Під час реалізації цієї ідеї ні в алгоритмі, ні в самій програмі програміст не вказує в якій області адрес пам'яті будуть розміщуватись масив чисел і команди програми, де знаходяться числа, що порівнюються між собою, яким чином і де виробляються ознаки "0" і "1" в блоці перевірки умов для організації розгалуження, де організовується лічильник циклів для адресації елементів масиву.

Програмування мовою Асемблер вимагає враховувати всі ці моменти як під час складання алгоритму, так і під час запису програми. Алгоритм розв'язання цієї ж самої програми, орієнтований на можливості мови Асемблер, представлений на рис. 2.2.

В алгоритмі прийняті такі позначення:

NNNN – адреса першої комірки пам'яті, з якої починається масив;

N – число елементів масиву (в програмі прийнято N=12);

HL – регістрова пара, яка містить шістнадцяткову адресу чергового елемента масиву;

B – ім'я регістра, що використовується як лічильник;

A – акумулятор, використовується для зберігання MAX;

E – регістр для тимчасового зберігання поточного елемента масиву, з яким порівнюється MAX;

D – допоміжний регістр для обміну даними між **A** і **E**.

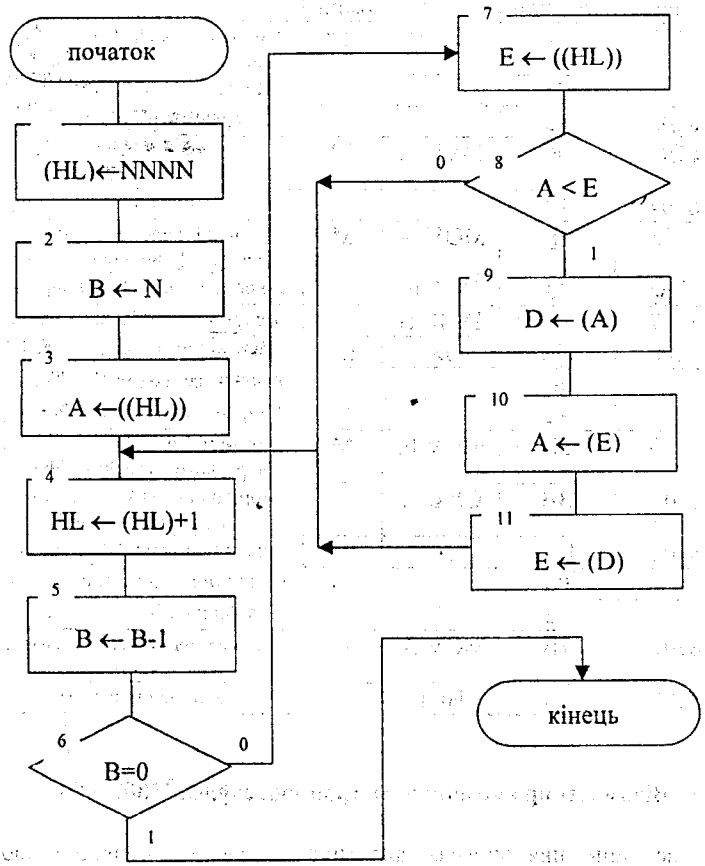


Рисунок 2.2 – Алгоритм пошуку максимального елемента масиву, орієнтований на мову Асемблер

Розподіл пам'яті перед початком запису програми виконаємо таким чином, щоб області адрес кодів програми і елементів масиву не перетинались між собою: початкова адреса масиву – 8270, початкова адреса програми – 8200.

Лістинг програми на мові Асемблер має такий вигляд:

Мітка	Адреса команди	Код команди	Мнемокод	Операнд	Коментар
	8200	21	LXI H,	8270	Завантаження адреси І елемента масиву в регістрову пару HL
	8201	70			Молодший байт адреси 8270
	8202	82			Старший байт адреси 8270
	8203	06	MVI B,	12	В регістр В завантажується кількість циклів
	8204	12			Число циклів
	8205	7E	MOV A,	M	Перший елемент масиву завантажуємо в акумулятор
L1:	8206	23	INX H		Інкремент регістрової пари HL
	8207	05	DCR B		Декремент лічильника циклів
	8208	CA	JZ	L2	Перехід по нулю на мітку L2
		0D			Молодший байт мітки переходу
		82			Старший байт мітки переходу
	8209	5E	MOV E,	M	Читання з пам'яті в регістр Е чергового елемента масиву
	820A	BB	CMP E		Порівняння MAX з черговим елементом масиву
	820B	DA	JC	L1	Перехід на мітку L1, якщо A<C
		06			Молодший байт мітки переходу
		82			Старший байт мітки переходу
	820C	7B	MOV A,	E	Заміщення MAX елементом масиву
L2:	820D	76	HLT		Зупинка програми

2.2.3 Відомості про команди мікропроцесора КР580ИК80

Під час написання програм для мікропроцесора необхідно добре знати його систему команд. Це означає, що потрібно пам'ятати весь перелік команд та добре уявляти собі ті дії, які будуть виконуватись мікропроцесором під час виконання кожної з них.

Код операції в будь-якій команді (для однобайтової команди – це просто код команди) в ЗП мікро-ЕОМ подається двійковим восьмирозрядним числом. Наприклад, код команди пересилання з регістра С в регістр А (MOV A,C) буде мати вигляд 01111001, код операції команди безпосереднього запису восьмирозрядного операнда записується так: 00110110. Двійковим кодом можна представити 2^8=256 різноманітних комбінацій. Майже стільки ж команд мають і МП (деякі комбінації двійкових восьмирозрядних чисел не використовуються і тому команд менше, ніж 256).

Звісно ж, запам'ятати понад 200 кодів команд, представлених у вигляді восьмирозрядних чисел, тобто у вигляді набору одиниць і нулів,

майже неможливо. Тому кожному коду команди ставиться в відповідність мнемонічна назва (мнемоніка) команди, яка є скороченням від англійських слів, які описують її дії. Мнемокод команд дозволяє легше запам'ятати їх функції і значно спрощує написання програм.

Так, перша з вище згаданих команд (пересилання з регістра **C** в регістр **A**) в мнемокоді має вигляд **MOV A,C (MOVE REGISTER)**, друга- **MVI M,D8 (MOVE TO MEMORY IMMEDIATE)**), де **M** – комірка пам'яті, адресована у відповідності з описом команди, **D8** – однобайтовий операнд.

Назви регістрових пар в мнемоніці команд наводяться в скороченому вигляді за допомогою перших літер їх назви: так, замість **BC, DE** і **HL** записується відповідно **B, D і H.**

Усі команди МП описані в таблиці на передній панелі НВП. За допомогою цієї таблиці можна легко та швидко порівняти мнемоніку команди з її кодом операції. Код операції кожної команди наведений тут у крайньому лівому та правому стовпцях (молодші розряди) та в верхньому і нижньому горизонтальних рядках (старші розряди) в шістнадцятковому вигляді. Таким чином, знаючи мнемоніку команди, наприклад, **ORA C**, за допомогою таблиці можна визначити її шістнадцятковий код операції **B1**, що буде відповідати двійковому коду 10110001.

При виконанні МП деяких команд в регістрі ознак виробляються ознаки стану, при цьому встановлюються в 1 такі біти регістра **F**:

Біт **Z** – ознака нуля, встановлюється "1", якщо результат виконання команди дорівнює 0.

Біт **S** – знак результату, встановлюється в "1", якщо результат виконання команди від'ємний. Під час виконання арифметичних команд кожен двійковий операнд подається у вигляді семирозрядного двійкового числа зі знаком записаним у старшому розряді. Одиниця у восьмому розряді відповідає від'ємному числу в доповнювальному коді.

Біт **P** – ознака парності, встановлюється, якщо кількість одиниць у двійковому коді результату парна.

Біт **C** – ознака перенесення, встановлюється, якщо в результаті додавання двох восьмирозрядних чисел з'являється перенесення із старшого розряду чи в результаті віднімання виникає позика.

Біт **AC** – ознака допоміжного перенесення, встановлюється коли з'являється перенесення з четвертого розряду двійкового числа (з розряду D3). Ця ознака використовується в різних операціях з чотирирозрядними операндами.

Докладніше з арифметичними командами мікропроцесора Intel 8080 можна ознайомитися в [2, 7].

Формат регістра прапорців F:

D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
S	**Z**	**0**	**AC**	**0**	**P**	**1**	**C**

Функціональне описання команд наведено в таблиці 2.

Таблиця 2.1 – Команди МП Intel 8080

Однобайтові пересилання	Двобайтові пересилання
MOV R1,R: R→R1	LXI YZ,D16: D16→Y2
MVI R,D8: D8→R	SHLDADR: HL→M(ADR)M(ADR+1)
STAX YZ^{+}: A→M(YZ)	LHLD ADR: M(ADR)M(ADR+1) →HL
STA ADR: M(ADR) →A	MVI
SPHL: HL→SP	POP YZ^{++}: M(SP)M(S'P+1) →YZ
	(POP' PSW): SP+2→SP
LDAX YZ^{+}: M(YZ) →A	PUSH YZ^{++}: YZ→M(SP-1)M(SP-2)
	: SP-2
Команди введення та виведення	Обмін байтами
IN N: (N) →A	XCHG: HL ↔ DE
OUT N: A→N	XTHL: H ↔ M(SP+1),L ↔ M(SP)

Арифметичні та логічні операції з одним операндом	
CMC": C→C	INR'"R: R+1→R
STC": 1→C	DCR'"R: R-1→R
CMA: A→A	INX YZ: YZ+1→YZ
DAA: десяткова корекція	DCX YZ: YZ-1→YZ

Арифметичні і логічні 8-бітові операції	Операції з двома операндами
ADD' R: A+R→A	ADI' D8: A+D8→A
SUB' R: A+R+C→A	ACI' D8: A+D8+C→A
SBB' R: A-R-C→A	SVI' D8: A-D8→A
ANA' R: AΛR→A	ANI' D8: AΛD8→A
ORA' R: AVR→A	ORI' D8: AVD8→A
XRA' R: A+→A	XRI' D8: A+D8→A
CPI' D8: Встановлення ознак	
CMP' R: У відповідності з A-D8 чи A-R 16-бітові операції	
DAD" YZ: HL+YZ→HL	

Команди зсуву вмісту акумулятора	Команди передачі управління
RLC": Зсув вліво	PCHL: HL→PC
RAL": Зсув вліво через біт ознаки C	JMP ADR: ADR→PC
RRC": Зсув вправо	J-CON ADR: ADR→PC
RAR": Зсув вправо через біт ознаки C	

Спеціальні команди	Команди виклику і повернення з підпрограми
EI: Дозвіл на переривання	CALL ADR: PC→M(SP-1)M(SP-2)
	C-CON ADR: ADR→PC
DI: Заборона на переривання	RST X: PC→M(SP-1)M(SP-2), :ADR→PC, де x=0,1,…,7 :ADR відповідно дорівнює : 0H,8H,10H,18H,20H,28H, :30H,38H
HLT: зупинка NOP: холоста операція	RET: M(SP)M(SP+1) →PC, R-CON: SP+2→SP

Умовні позначення:

' – команда діє на усі ознаки регістра **F**,

" – команда діє на ознаку **C**,

"' – команда діє на усі ознаки, окрім **C**.

R,R1 – вміст регістрів **A, B, C, D, E, H, L** чи комірки пам'яті **M(HL)**,

YZ – вміст регістрової пари **BC, DE, HL** чи регістра **SP**,

YZ^{+} – вміст регістрової пари **BC** чи **DE**,

YZ^{++} – вміст регістрової пари **BC, DE, HL** чи **PSW**,

SP – вміст покажчика стека перед виконанням програми,

D8 – восьмирозрядний операнд (вміст другого байта двобайтової команди),

(N) – вміст порту введення чи виведення з номером N (N=0,1,…,255),

D16 – шістнадцятирозрядний операнд (вміст другого та третього байта команди),

ADR – шістнадцятирозрядна адреса у трибайтовій команді,

M( ) – вміст комірки пам'яті (адреса комірки вказана у дужках),

-CON – частина мнемоніки команди, яка визначає умову передачі керування, виклику і повернення з підпрограми (- **CON** в мнемоніці замінюється на **NZ, Z, NC, C, PO, PE, P** чи **M**).

Замість декількох однотипних команд в таблиці розташована одна узагальнена команда. В такій команді замість позначення конкретного регістра чи регістрової пари використовується узагальнене позначення декількох регістрових пар. Стрілкою → позначається напрямок передачі даних під час виконання команди.

2.2.4 Коротка характеристика деяких груп команд МП

Група команд однобайтових пересилань даних

З їх допомогою проводиться обмін даними між внутрішніми регістрами МП, а також між внутрішніми регістрами і комірками пам'яті. **MOV R1, R** – вміст регістра R пересилається в регістр **R1**, при цьому в регістрі R зберігається попереднє значення даних.

В якості **R1** або **R** може бути визначена комірка пам'яті, адреса якої знаходиться в парі **HL**. Наприклад, по команді **MOV C,M** відбудеться пересилання даних з комірки пам'яті з адресою, записаною в регістровій парі HL, у внутрішній регістр **C** мікропроцесора. Вміст комірки пам'яті **M** при цьому не змінюється.

Для пересилання даних між акумулятором і коміркою пам'яті в якості адреси пам'яті може бути використаний також вміст регістрових пар **BC** і **DE**. Тоді для запису в пам'ять даних використовуються однобайтові команди **STAX B** або **STAX D**, а для зворотного пересилання **LDAX B** або **LDAX D**.

Адресу комірки пам'яті для обміну з акумулятором можна задати також за допомогою трибайтових команд з безпосередньою адресацією. У цьому випадку для запису даних з акумулятора в пам'ять є команда **STA ADR**, для зворотного пересилання **LDA ADR**.

За допомогою двобайтової команди **MVI R,D8** (**R**- літер. наймен. регістра або комірки пам'яті, адресованої по вмісту **HL**) можна записати операнд в будь-який восьмирозрядний регістр МП або комірку пам'яті. Операндом тут буде вміст другого байту команди.

Команда **SPHL** дозволяє занести адресу з регістрової пари HL у покажчик стека **SP**.

Група команд двобайтових пересилань даних

Трибайтові команди **LXI B, D16;LXI D,D16; LXI H, D16** слугують для безпосереднього запису у відповідні регістрові пари шістнадцяти-розрядного операнда **D16**.

Використовуючи команди **SHLD ADR, LHLD ADR**, можна організовувати пересилання даних між регістровою парою **HL** і коміркою пам'яті, безпосередньо адресованою за вмістом другого і третього байта команди. Інші команди цієї групи виконують пересилання з адресуванням за покажчиком стека **SP**. За допомогою команд **PUSH B, PUSH D** і **PUSH**

H вміст регістрових пар **BC, DE** і **HL** посилаються в стек. За командою **PUSH PSW** в стек посилаються дані з акумулятора і регістра ознак. Команди **POP B, POP D, POP H** слугують для пересилання шістнадцятирозрядного слова з комірки пам'яті, адресованих покажчиком стека **SP** у відповідну пару регістрів. Командою **POP PSW** дані із стека пересилаються в акумулятор і регістр ознак **F**. Таким чином, команда **POP PSW** може змінювати усі біти регістра ознак **F**.

Група команд обміну даними.

XCHG – обмін вмістом між регістровими парами **HL** і **DE**,

XTHL - обмін вмістом між регістровою парою **HL** і комірками пам'яті, адресованих за покажчиком стека **SP**.

Група команд арифметичних команд і операцій з одним операндом.

CMC – інвертує ознаку перенесення.

STC – встановлює значення ознаки перенесення в 1.

CMA – інвертує усі біти в регістрі **A**.

DAA – виконує двійково-десяткове додавання.

Дуже часто під час написання програм використовуються команди **INR R, DCR R, INX YZ, DCX YZ,** які слугують для збільшення чи зменшення значення вмісту регістра, комірки пам'яті чи регістрової пари на одиницю. Багато з команд цієї групи діють на різні біти регістра ознак F.

Група команд арифметичних і логічних операцій з двома операндами.

Перед початком виконання будь-якої команди з цієї групи один з операндів повинен бути розміщений в регістрові **A**, а другий операнд (якщо команда однобайтова) в одному з восьмирозрядних внутрішніх регістрів МП, або в комірці пам'яті, адресованої вмістом регістрової пари **HL**. В двобайтовій команді значення другого операнда безпосередньо задається в другому байті команди. Результат виконання команди записується в регістр **A**.

Команди **ADD R** і **ADI D8** дозволяють додати два операнди. Додавання двох операндів зі значенням біта перенесення **C** приходить за командою **ADC R** або **ACI D8**. Віднімання із акумулятора другого операнда та врахування значення біта позички **C** виконується відповідно командами **SUB R, SUI D8, SBB R** або **SBI D8**.

Операції порозрядного логічного добутку (операція **I**, кон'юнкція, позначається Λ) вмісту акумулятора з другим операндом відбувається під час виконання команди **ANA R** або **ANI D8**.

ORA R або **ORI D8** – порозрядне логічне додавання (операція **АБО**, диз'юнкція, позначається V).

XRA R або **XRI D8** – виконання операції "Виключне АБО" (позначається знаком ⊕). Результат – байт, окремі розряди якого дорівнюють "1" тільки тоді, коли відповідні розряди операндів мають протилежні значення. Після виконання розглянутих команд логічної обробки двох операндів значення ознак **C** і **AC** регістра ознак **F** завжди дорівнюють 0.

CMP R або **CPI D8** – порівняння двох операндів. Порівняння виконується відніманням від першого операнда, що зберігається в акумуляторі, другого. Якщо в результаті операції віднімання буде встановлено, що операнди однакові, то ознака нуля **Z** встановлюється в 1, якщо ж значення операнда, який зберігається в акумуляторі, менше значення другого операнда, то встановлюється в 1 ознака перенесення **C**.

DAD B, DAD D, DAD H, DAD SP – команди, які дозволяють додати два 16-розрядні числа. Одне з цих чисел повинне бути записано в регістрову пару **HL**, а інше – в регістрову пару **BC**, **DE**, **HL** або **SP**. Результат додавання розташовується в регістровій парі **HL**.

Група команд зсувів вмісту акумулятора.

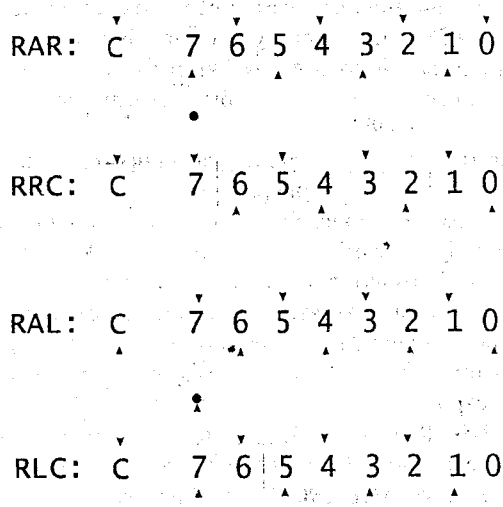


Рисунок 2.3 – Схеми реалізації зсувів командами МП 8080

На рис. 2.3 схематично показано, як відбувається зсув вмісту акумулятора ліворуч або праворуч командами зсуву **RAL** і **RAR**, і командами циклічного зсуву **RLC** і **RRC**. В операціях зсуву бере участь біт перенесення **C** регістра ознак **F**.

Під дією кожної з цих команд відбувається зсув вмісту акумулятора тільки на один розряд. Якщо необхідно зсунути вміст акумулятора на більше число розрядів, то команду зсуву необхідно повторити стільки раз, скільки вимагається.

Група команд передачі керування.

Ці команди грають особливу роль в організації виконання програм мікро-ЕОМ. Доти, доки в програмі не зустрінуться команди цієї групи, лічильник команд РС постійно збільшує своє значення, і МП виконує команду за командою в порядку їх розташування в пам'яті.

Порядок виконання програми може бути змінений, якщо занести в регістр лічильника команд МП адреси, які відрізняються від адреси чергової команди. Це викличе передачу керування роботою МП іншої частини програми.

Така передача керування (чи перехід у програмі) може бути виконана за допомогою трибайтової команди безперечного переходу **JMP ADR**. Як тільки ця команда зустрічається у програмі, в регістр лічильника команд РС мікропроцесора запишеться величина **ADR**. Таким чином, наступною командою, яку буде виконувати МП слідом за командою **JMP ADR**, буде команда, код операції якої записаний в комірці з адресою, яка дорівнює значенню **ADR**.

PCHL - також безпосередня передача керування (за адресою, яка зберігається в регістровій парі **HL**).

МП має вісім трибайтових команд умовного переходу. Передача управління за адресою, зазначеною в команді, здійснюється тільки у випадку виконання зазначеної умови. Якщо умова не задовольняється, то виконується команда, яка безпосередньо йде за командою умовного переходу.

Умови, з якими оперують команди умовної передачі керування, визначаються станом бітів (розрядів) ознак **F**:

NZ(NOT ZERO) – нульовий результат, Z=0;
Z(ZERO) – нульовий результат, Z=0;
NC(NO CARRY) – відсутність, C=0;
C(CARRY) – перенесення, C=1;
PO(PARITY ODD) – непарний результат, P=0;
PE(PARITY EVEN) – парний результат, P=1;
P(PLUS) – від'ємне число, S=0;
M(MINUS) – невід'ємне число, S=1.

Ці умови перевіряються відповідними командами умовного переходу: **JNZ ADR, JZ ADR, JNC ADR, JC ADR, JPO ADR, JPE ADR, JP ADR, JM ADR**.

2.3 Приклади виконання роботи

1. Вмикаємо НВП перемиканням тумблера «Сеть» в положення «Вкл». Натискаємо клавішу **RST**, при цьому на індикаторі з'явиться значення 8200 **??
2. Встановлюємо перемикач «Прогон - отладка» в положення «Отладка».
3. Записуємо програму додавання чисел згідно таблиці 2.2 прикладу 1, вводимо її в пам'ять згідно з таблицею 2.3 і виконуємо послідовність дій згідно з таблицею 2.4:

   1) Задаємо початкову адресу програми NNNN, натиснувши клавіші **ADDR NN N N**.
   2) Натискаємо клавішу **STEP**.
      Після виконання чергової команди здійснюється зупинка.
      На індикаторі буде відображено нове значення лічильника команд у розрядах 1-4, вміст комірки пам'яті за цією адресою в розрядах 7-8.
   3) Повторюємо пункт 2) для всіх команд програми.
   4) Після виконання останньої команди переглядаємо вміст регістрів чи комірок пам'яті, в яких зберігаються результати.

Приклад 2.1. Програма додавання двох чисел 17_{16} і $B5_{16}$ наведена в таблиці 2.2.

Таблиця 2.2 – Програма додавання двох чисел мовою Асемблер

Адреса команди	Код команди	Мнемоніка	Операнд	Коментар
8200	3E	MVI A,	17	Запис в акумулятор А числа 17
8201	17			
8202	06	MVI B,	B5	Запис в регістр В числа B5
8203	B5			
8204	80	ADD B		Додавання чисел 17_{16} і $B5_{16}$
8205	76	HLT		Зупинка

Попередньо необхідно записати програму в пам'ять НВП за допомогою таких дій (табл. 2.3).

Таблиця 2.3 – Приклад введення програми в пам'ять НВП з клавіатури

Клавіші	Інформація на індикаторі		Коментар
	1 2 3 4	5 6 7 8	
RST	8 2 0 0	* * ? ?	Скидання НВП
ADDR	8 2 0 0	* * ? ?	Установка адр. 8200
MEM	8 2 0 0	* *, ? ?	
3	8 2 0 0	* *, 0 3	Програми з адреси 8200_{16}
E	8 2 0 0	* *, 3 E	Запис коду 3E
NEXT	8 2 0 1	* *, ? ?	
1	8 2 0 1	* *, 0 1	
7	8 2 0 1	* *, 1 7	Запис коду 17
NEXT	8 2 0 2	* *, ? ?	
0	8 2 0 2	* *, 0 0	
6	8 2 0 2	* *, 0 6	Запис коду B6
NEXT	8 2 0 3	* *, ? ?	
B	8 2 0 3	* *, 0 B	
5	8 2 0 3	* *, B 5	Запис коду B5
NEXT	8 2 0 4	* *, ? ?	
8	8 2 0 4	* *, 0 8	
0	8 2 0 4	* *, 8 0	Запис коду 80
NEXT	8 2 0 5	* *, ? ?	
7	8 2 0 5	* *, 0 7	
6	8 2 0 5	* *, 7 6	Запис коду 76

Таблиця 2.4 – Послідовність дій для покрокового виконання програми

Клавіші	Інформація на індикаторі		Примітки
	1 2 3 4	5 6 7 8	
ADDR	8 2 0 0	* * 3 E	Встановлення початкової адреси програми
STEP	8 2 0 2	* * 0 6	По кроках виконуємо команди до адреси 8205_{16} де закінчується програма
STEP	8 2 0 4	* * 8 0	
STEP	8 2 0 5	* * 7 6	
STEP	8 2 0 6	* * ? ?	
REG A	8 2 0 6	A-CC	Перегляд значення регістра А в розрядах 7-8 індикатора

Результат додавання чисел 17_{16} і $B5_{16}$ дорівнює CC_{16}. Під час виконання програми, яка записана з адреси 8200_{16}, в покроковому режимі можна переглянути за зміною вмісту будь-якого регістра МП. Наприклад, в розглянутій програмі додавання двох чисел результат формується в акумуляторі А:

ADDR 8200
REG A
STEP
STEP
STEP

2.4 Контрольні запитання та завдання

1. Для чого використовується система переривання?
2. Яким чином користувач може впливати на функції монітора?
3. Що трапляється під час формування сигналу скидання?
4. Пояснити структуру подання в ЗУ мікро-ЕОМ, як операції команди?
5. Що таке мнемоніка команди? Привести приклад декількох команд в мнемокоді, пояснити їх роботу.
6. Дати характеристику регістра ознак (прапорців) F, призначення його бітів.
7. Використовуючи таблицю 1, дати характеристику групі команд:
   а) однобайтових пересилань даних;
   б) двобайтових пересилань даних;
   в) арифметичних і логічних операцій з одним операндом;
   г) арифметичних і логічних операцій з двома операндами;
   д) зсувів вмісту акумулятора;
   е) передачі управління.
8. Вказати послідовність дій під час виконання програми:
   а) в покроковому режимі;
   б) в режимі з зупинкою в контрольних точках;
   в) в неперервному режимі.
9. Вказати послідовність дій складання програми і запису її в НВП.
10. Вивчити дії по виконанню програми в режимі з зупинкою в контрольних точках. Режим виконання програми з зупинкою в контрольних точках дозволяє зупинятися тільки при досягненні заданої адреси.

    Для виконання програми у режимі із зупинкою за заданими умовами (контрольним точкам) необхідно:
    1) встановити тумблер режиму в положення «отладка»,
    2) ввести контрольні точки – послідовність адрес, в яких ви бажаєте зупинятися,

3) задати початкову адресу програми **N N N N** , натиснувши клавішу **ADDR N N N N**,

4) запустити програму на виконання, натиснувши клавішу **RUN**.

Після виконання частини програми на індикаторі відобразиться адреса контрольної точки на якій трапилась зупинка. В цьому режимі програма автоматично переривається монітором після виконання кожної команди для контролю точок зупинки.

Клавіатура і індикатор будуть блоковані до моменту, поки поточна адреса програми не буде дорівнювати адресі контрольної точки і число проходів даної контрольної точки не буде дорівнювати нулю. Після цього буде викликана програма монітора і на індикаторі з'явиться значення лічильника команд, яке відповідає контрольній точці, і вміст пам'яті за цією адресою.

Виконати програму прикладу 1 в режимі з зупинкою в контрольних точках.

11. Вивчити дії по виконанню програми в неперервному режимі.

В цьому режимі НВП працює без вмикання монітора. Для того, щоб після виконання програми здійснилось переривання і звертання до монітора, який включає в себе індикатор, необхідно у якості команди зупинки використовувати не **HLT**, а команду **RST32**(її код $E7_{16}$).

Для виконання програми у неперервному режимі потрібно:

1) встановити тумблер режиму в положення "прогон",

2) задати початкову адресу програми **N N N N : ADDR N N N N,**

3) запустити програму клавішею **RUN**.

Після виконання програми на індикаторі відобразиться значення адреси команди, на якій трапилася зупинка і дані за цією адресою. Виконати програму прикладу 1 в неперервному режимі.

Варіанти завдань.

1.Записати програму віднімання двох восьмирозрядних операндів з індексацією результату на дисплеї. Виконати програму в покроковому режимі. Перший операнд помістити в акумулятор.

	1	2	3	4	5	6
Значення 1/2 операндів	3F/20	2A/19	0E/03	20/14	31/14	10/0A
Початкова адреса програми	8210	8000	82FF	8010	8220	8200
2-й операнд помістити в	рег. B	D	H	8300	80FE	810A

2. Записати розгалужену програму з будь-якої адреси. Виконати її в неперервному режимі для двох значень *x*. Прочитати результат виконання операцій у відповідному регістрі.

	Варіанти					
	1	2	3	4	5	6
Умова	x≥12	x=3C	x-парне	x<2A	x+7D перенесення	x - непарне
Операції	(x+10)Λ03	(x+02)-1	(2A-x) в акумул., зсув вліво	(xVC1) +1	(x+7D) в акумул., зсув вправо	x в акумул., зсув вправо через біт ознаки C, інвертув. всіх бітів
Умова	x<12	x≠3C	x-непарне	x≥2A	x+7D відсутність переносу	x - парне
Операція	x+1	00→H	3A→E	2A→D	x→C	02→B
Значення x_1	20	3C	03	10	90	0B
x_2	10	31	01	30	80	0C
Результат занести в регістр	L	H	E	D	C	B

ЛАБОРАТОРНА РОБОТА № 3.

Укладання циклічних програм і їх відлагодження на НВП "Електроніка – 580"

Мета роботи – отримання навичок складання і відлагодження циклічних програм на мові Асемблер МП 8080

3.1 Порядок виконання роботи

3.1.1 Ознайомитися з теоретичними відомостями, наведеними в даних методичних вказівках.

3.1.2 Для заданого викладачем варіанту задачі скласти програму в мнемокодах мови Асемблер, записати її в машинних кодах та задокументувати.

3.1.3 Ввести програму в пам'ять НВП "Електроніка 580" та виконати в покроковому режимі.

3.1.4 Зафіксувати та проаналізувати отримані результати.

3.1.5 Оформити звіт про виконання лабораторної роботи.

3.2 Теоретичні відомості

3.2.1 Особливості організації циклів мовою Асемблер

Циклічним називається алгоритм, в якому передбачається багаторазове виконання однієї і тієї ж послідовності дій для різних значень величин, що підлягають обробці цими діями. Така послідовність дій називається циклом, а величини, що змінюють свої значення під час кожного повторення циклу, називаються параметрами циклу. Використання циклів дозволяє значно скоротити програму.

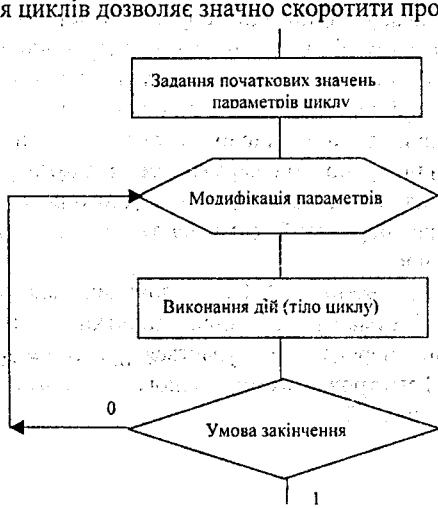


Рисунок 3.1 – Основні етапи циклічної дільниці алгоритму

Як видно з рис.3.1, для організації циклу потрібно задати початкові значення параметрів циклу, закон зміни параметрів циклу перед кожним новим його повторенням, вигляд виконуваних дій (тіло циклу), умову виходу з циклу. В разі невиконання вказаної умови цикл повторюється спочатку. Для організації циклу в програмі на мові Асемблер використовуються команди порівняння (**CMP, CPI**), інкрементування (**INR, INX**) або декрементування (**DCR, DCX**) даних в регістрах або комірках пам'яті, і команди умовного переходу (**JNZ ADR, JZ ADR, JNC ADR, JC ADR, JPO ADR, JPE ADR, JP ADR, JM ADR**).

3.2.2 Моделювання процесу множення в АЛП на НВП "Електроніка-580".

В ЕОМ операція множення чисел з фіксованою комою за допомогою відповідних алгоритмів зводиться до операцій додавання і зсуву. Особливості виконання операції множення чисел з фіксованою комою можна визначити на конкретному прикладі. Виконаємо, наприклад множення.

```
   1 1 0      - множене
 × 1 1 0      - множник
 -------
   0 0 0      - I  частковий добуток
   1 1 0      - II частковий добуток
 1 1 0        - III частковий добуток
 -------
 1 0 0 1 0 0  - добуток
```

З наведеного прикладу видно, що добуток двох **n**-розрядних чисел може мати **2n** розрядів. Тому під час виконання операції множення цілих чисел необхідно передбачити можливість формування в АЛП добутку, який має подвійну в порівнянні зі співмножниками довжину. Операція множення має **n** циклів, де **n** – число розрядів множника. Також з наведеного прикладу видно, що при нерухомому множнику потрібно на кожному кроці множення зсувати вміст дворозрядного регістра часткових добутків вправо, а при нерухомій сумі часткових добутків потрібно зсувати співмножник вліво.

Для виконання множення АЛП повинен містити регістри множеного, множника і суматор часткових добутків, в якому шляхом відповідної організації передач відбувається послідовне додавання часткових добутків. Регістрова схема одного з можливих методів множення зображена на рис.3.2.

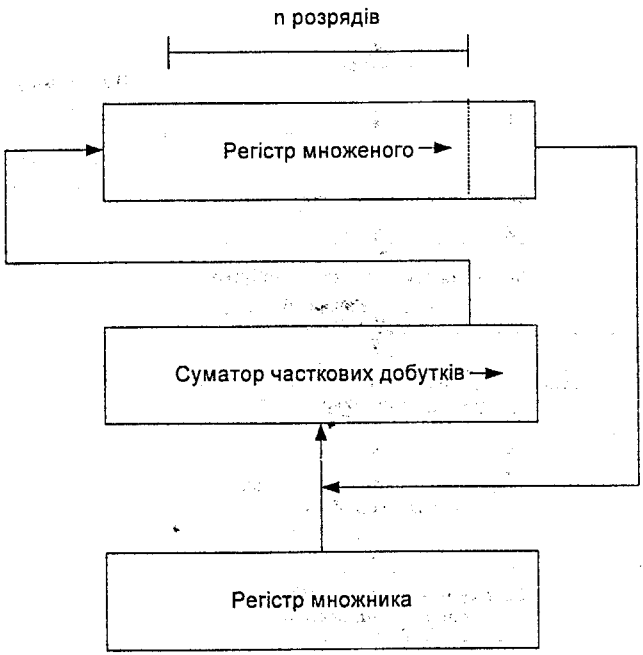


Рисунок 3.2 - Регістрова схема множення зі зсувом суми часткових добутків вправо коли нерухомий множник, починаючи з молодших розрядів множеного

Даний метод множення реалізується за допомогою такого алгоритму:

1. Беруться модулі співмножників.
2. Встановлюється початкове значення лічильника циклів (рівне розрядності числа).
3. Обнуляється значення суми часткових добутків.
4. Якщо цифра, що аналізується, множеного дорівнює 1, то до суми часткових добутків додаємо множник; якщо ця цифра дорівнює 0, додавання не виконується.
5. Здійснюється зсув часткових добутків вправо на один розряд.
6. Здійснюється зсув множеного на 1 розряд вправо.
7. Вміст лічильника циклів зменшується на одиницю.
8. Якщо значення лічильника не дорівнює 0, то повторити все з пункту 4, інакше на пункт 9.
9. Добутку присвоюється знак "+" якщо знаки співмножників однакові, в протилежному випадку - знак "-".
10. Кінець.

Блок-схема алгоритму множення зображена на рис. 3.3

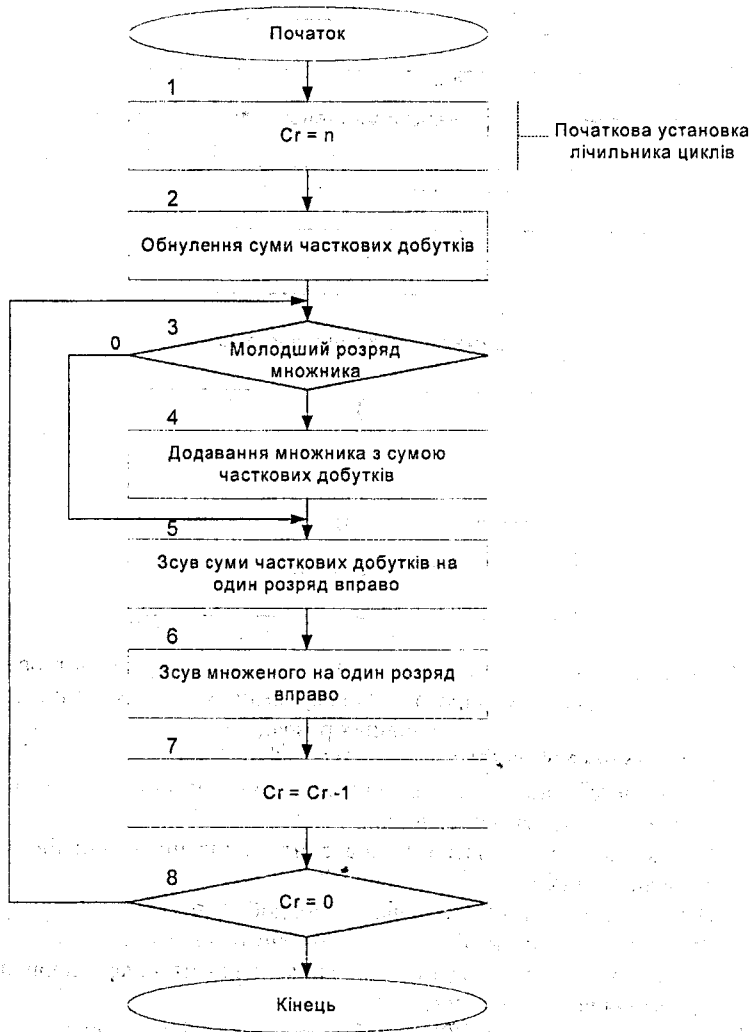


Рисунок 3.3 – Схема алгоритму множення цілих чисел

3.3 Приклад запису циклічної програми

За приклад програми з використанням циклу розглянемо задачу сортування масиву чисел на додатні і від'ємні. Нехай початковий масив містить 25 чисел, розміщених в пам'яті за адресами 8200_{16} ÷ 8218_{16}.

Додатні числа потрібно розмістити в пам'яті з адреси 8220_{16}, а від'ємні – з адреси 8240_{16}.

Схема алгоритму для розв'язання поставленої задачі зображена на рис. 3.2, а програма наведена в табл.3.1. Ідея алгоритму полягає в циклічному перегляді масиву, в процесі якого на кожному кроці черговий елемент перевіряється на рівність нулю, а потім на додатність. В разі рівності нулю елемент не змінює свого місцеположення, в разі додатності або від'ємності записується в чергову комірку заданої області пам'яті.

З метою скорочення довжини команд використаємо опосередковано-регістровий спосіб адресації елементів масиву в пам'яті. При цьому регістрова пара **HL** слугує покажчиком адреси чергового елемента початкового масиву, який підлягає сортуванню. Регістрова пара **BC** є покажчиком адреси вільної комірки пам'яті для запису чергового додатного елемента масиву, а регістрова пара **DE** – для від'ємного. Лічильник циклу організуємо в комірці пам'яті з адресою 8260_{16}, в якій спочатку буде записано число елементів масиву (19_{16}). Її вміст в кожному циклі зменшується на одиницю. Для визначення знаку числа аналізується його знаковий розряд (старший біт двійкового слова). Для цього воно записується в акумулятор і зсувається вліво через біт переносу циклічним зсувом (команда **RLC**). За цих умов біт переносу стає рівним знаковому розряду – якщо він дорівнює "0", то число додатне, якщо "1" – від'ємне. Перед записом в пам'ять число відновлюється шляхом виконання циклічного зсуву акумулятора вправо. Елементи масиву, рівні нулю, залишаються на своєму місці.

Слід відзначити, що перші п'ять команд роблять програму придатною для розв'язання тільки даної конкретної задачі. Частина програми, що починається з адреси $810E_{16}$, є універсальною в тому розумінні, що вона може сортувати масив потрібного розміру (в межах до 255 елементів), розміщений в будь-якій області пам'яті, якщо у відповідні регістрові пари попередньо занесено інформацію про кількість елементів масиву та адреси його розміщення в пам'яті.

Для зберігання шістнадцятирозрядних чисел в МП можна використовувати три регістрові пари - **BC, DE, HL,** покажчик стека **SP** і лічильник команд **PC**. При цьому в регістрах **B, D** і **H** регістрових пар зберігаються старші байти чисел, а в регістрах **C, E** і **L** – їх молодші байти. В операціях зі стеком як шістнадцятирозрядне число розглядається також сукупність регістра **A** (старший байт) і регістра ознак **F** (молодший байт), яка іменується як **PSW**.

Шістнадцятирозрядним числа для їх зберігання в пам'яті завжди відводяться дві суміжні комірки. Запис чисел в ці комірки відбувається побайтно, причому у комірку з меншою адресою записується молодший байт, а в комірку з більшою адресою – старший байт числа. Це правило виконується за будь-яких способів адресації, а також під час запису в

пам'ять трибайтових команд, де другий і третій байти є відповідно молодшим і старшим байтами шістнадцятирозрядного числа.

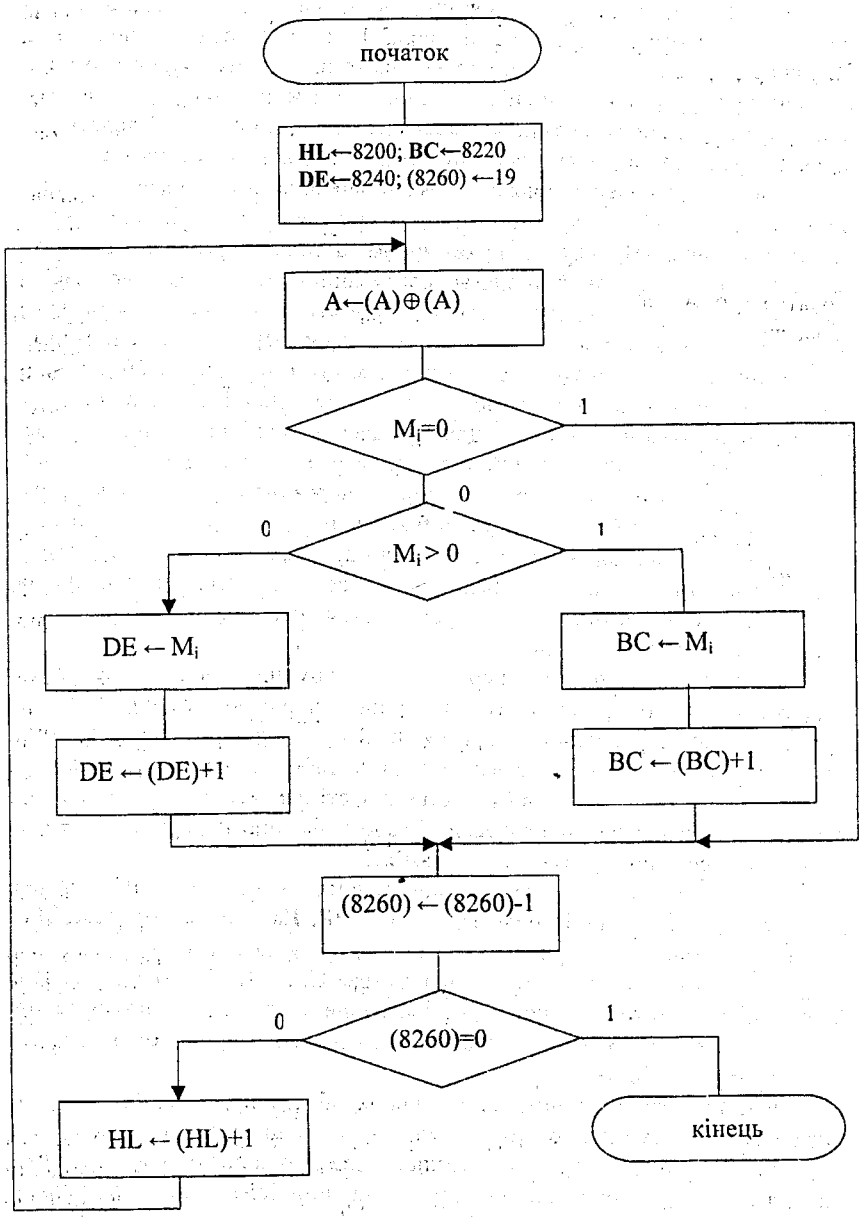


Рисунок 3.2 – Схема алгоритму сортування масиву

Мітка	Адреса команди	Код команди	Мнемокод	Операнд	Коментар
	8100	21	LXI H,	8200	Завантаження адреси 1-го елемента початкового масиву в регістрову пару HL
	8101	00			Молодший байт адреси 8200
	8102	82			Старший байт адреси 8200
	8103	01	LXI B,	8220	Завантаження адреси 1-го елемента додатного масиву в регістрову пару BC
	8104	20			Молодший байт адреси 8220
	8105	82			Старший байт адреси 8220
	8106	11	LXI D,	8240	Завантаження адреси 1-го елемента від'ємного масиву в регістрову пару BC
	8107	40			Молодший байт адреси 8240
	8108	82			Старший байт адреси 8240
	8109	3E	MVI A,	19	Завантаження в акумулятор числа циклів 19_{16}
	810A	19			Число 19_{16}
	810B	32	STA,	8260	(8260) ←19_{16}
	810C	60			Молодший байт адреси лічильника циклів
	810D	82			Старший байт адреси лічильника циклів
L4:	810E	AF	XRA	A	Обнулення акумулятора
	810F	BE	CMP	M	(A)=0
	8110	CA	JZ,	L1	Перейти на адресу 811B, якщо елемент масиву рівний нулю
	8111	1B			
	8112	82			
	8113	7E	MOV A,	M	Завантаження в акумулятор чергового елемента масиву
	8114	07	RLC		Зсув вмісту акумулятора вліво
	8115	DA	JC,	L2	Перехід на адресу 812A, якщо елемент масиву менший нуля
	8116	2A			
	8117	81			
	8118	0F	RRC		Зсув вмісту акумулятора вправо
	8119	02	STAX	B	Запис вмісту акумулятора в комірку пам'яті, адреса якої знаходиться в парі BC
	811A	03	INX	B	Інкремент вмісту регістрової пари BC
L1:	811B	3A	LDA,	8260	A←(8260) – Завантаження акумулятора вмістом комірки пам'яті з адресою 8260
	811C	60			
	811D	82			

	811E	3D	DCR A		Декремент акумулятора
	811F	C2	JNZ,	L3	Перехід за адресою 8123, якщо вміст акумулятора не дорівнює нулю
	8120	23			
	8121	81			
	8122	76	HLT		Зупинка програми
L3:	8123	23	STA,	8260	Завантаження акумулятора в комірку пам'яті з адресою 8260
	8124	60			
	8125	82			
	8126	INX	H		Інкремент пари HL
	8127	JMP,	L4		Перехід на адресу 810E
	8128	0E			
	8129	81			
L2:	812A	0F	RRC		Зсув вмісту акумулятора вправо
	812B	12	STAX	D	Запис вмісту акумулятора в комірку пам'яті, адреса якої знаходиться в парі DE
	812C	13	INX	D	Інкремент пари DE
	812D	C3	JMP,	L1	Перехід на адресу 811B
	812E	1B			
	812F	81			

3.4 Контрольні питання та завдання

1. Який алгоритм називається циклічним?
2. Які команди мікропроцесора КР580 використовуються для організації циклу?
3. Розказати про порядок запису шістнадцяткових чисел в пам'ять і внутрішні регістри МП.
4. Записати програму множення двох восьмирозрядних чисел. За множник вибирається номер бригади, в якості множеного - остання цифра в списку журналу першого студента бригади. Виконати програму в покроковому режимі.
5. Виконати програму в неперервному режимі.

Вказівка: для перевірки значення молодшого розряду множеного використати тригер переносу, для чого використати команди зсуву через біт ознаки (RRC) і переходу по одиничному значенню тригера переносу.

ЛАБОРАТОРНА РОБОТА №4

РОБОТА З ПІДПРОГРАМАМИ МОВОЮ АСЕМБЛЕР 8080

Мета роботи: вивчення принципів використання команд з стековою адресацією, а також принципів використання підпрограм в програмах на мові Асемблер-8080.

4.1 Порядок виконання роботи

4.1.1 Ознайомитись з теоретичними відомостями і рекомендованою літературою.
4.1.2 Скласти алгоритм і програму відповідно до заданого варіанта.
4.1.3 Записати програму в кодах.
4.1.4 Ввести програму в пам'ять МП пристрою "Електроніка –580".
4.1.5 Виконати програму і проаналізувати результати.
4.1.6 Оформити звіт.

4.2 Теоретичні відомості

Підпрограма – це частина програми, яка, зазвичай, використовується декілька раз в процесі виконання програми. Однак текст підпрограми записується програмістом тільки один раз. В разі потреби скористатися підпрограмою достатньо в основній програмі вказати відповідну команду виклику, що адресує область пам'яті, в якій розміщена підпрограма.

Можливість використання підпрограм під час програмування мікропроцесорів забезпечується наявністю стекової пам'яті, що реалізує безадресне визначення операндів. В загальному випадку стек являє собою групу послідовних регістрів або комірок пам'яті і покажчик стека. Покажчик стека автоматично встановлює адресу останньої зайнятої комірки стека під час читання або запису в ньому чергового слова інформації. Запис і читання слова в стекові здійснюється за принципом "останнім прийшов – першим вийшов" в верхню комірку стека. Тому в операціях зі стеком можливе безадресне визначення операнда – команда не містить адреси комірки стека, але містить адресу комірки пам'яті або регістра, з яких слово передається в стек або куди завантажується із стека.

В мікропроцесорі 8080 використовується так званий "зворотний стек", в якому під час передавання в стек слова вміст покажчика стека (регістр SP) зменшується, а під час вилучення слова зі стека – збільшується. Цей стек організований програмно, тобто безпосередньо на кристалі мікропроцесора з обладнання стекової пам'яті знаходиться тільки регістр-покажчик стека, а сам стек реалізується в оперативній пам'яті у вигляді групи послідовних комірок. Покажчик стека завантажується старшою адресою, яка визначає вершину стека в пам'яті (рис. 4.1).

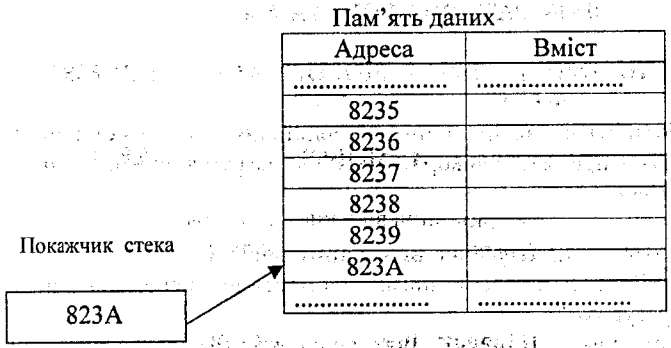


Рисунок 4.1- Розміщення стека в ОЗП

В наведеному на рис.4.1 прикладі покажчик стека містить адресу 823А, що на одиницю вища першої комірки пам'яті стека 8239.

Дані можна записати в стек командами **PUSH RP"**(занести) і **CALL A16** (викликати). Прочитати їх зі стека можна за командами **POP RP"** (вилучити) або **RETURN** (повернення). Сполучення літер **RP"** умовно означає одну з регістрових пар **BC, DE, HL** або слово стану мікропроцесора **PSW**.

На рис.4.2 наведено послідовність дій під час завантаження в стек за командою **PUSH HL** вмісту регістрової пари **HL**. Цифри над стрілками показують послідовність виконання дій.

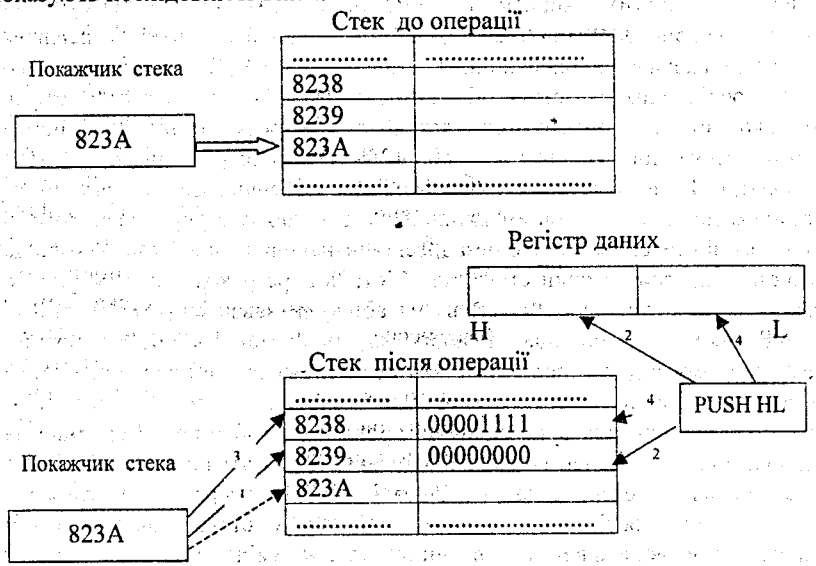


Рисунок 4.2 – Послідовність дій під час виконання операції зі стеком PUSH HL; H - старший байт регістра, L - молодший

1. Покажчик стека декрементується від 823А до 8239.
2. Покажчик стека вказує на комірку пам'яті 8239 і старший байт регістрової пари (00000000_2) завантажується в цю комірку стека.
3. Покажчик стека знову декрементується від 8239 до 8238.
4. Покажчик стека показує на комірку пам'яті 8238 (за адресною шиною системи) і молодший байт з регістра **HL** ($0000\ 11111_2$) завантажується в стек.

В цих командах умовне позначення **RP"** означає одну з регістрових пар **BC**, **DE**, **HL** або **PSW** (шістнадцятирозрядне слово стану МП, що складається з вмісту регістра-акумулятора та вмісту регістра прапорців).

Стек може продовжувати зростати, поки триває процес завантаження в нього. Єдиним обмеженням при цьому може бути зайнятість пам'яті іншими програмами.

Зазвичай кожній команді завантаження в стек (**PUSH**) пізніше повинна відповідати команда читання зі стека (**POP**), для якої дані беруться зі стека. Оскільки стек є пам'яттю типу LIFO (last input – first output: останнім прийшов – першим вийшов), то дані повинні вийматися зі стека в порядку, оберненому завантаженню. Розглянемо операцію читання даних зі стека в регістрову пару **HL** командою **POP HL**, зображену на рис.4.3.

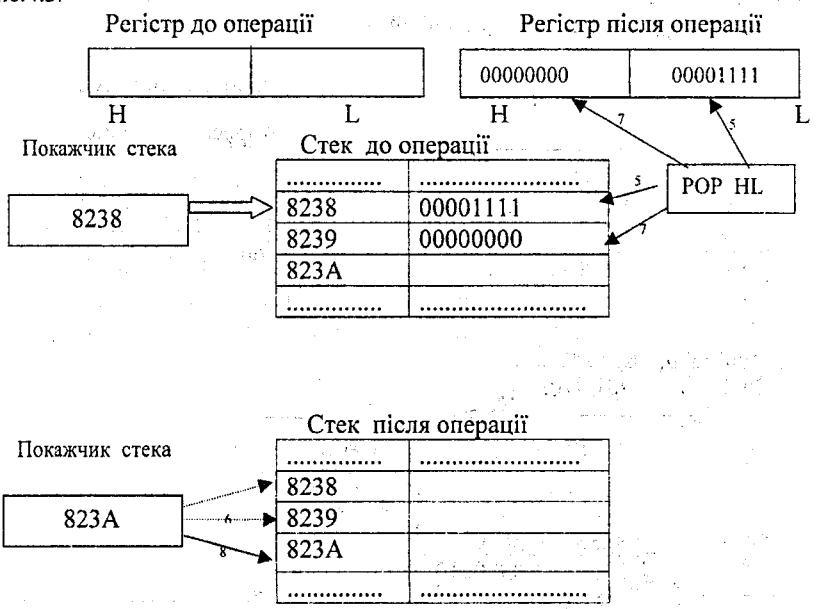


Рисунок 4.3 – Послідовність дій під час виконання операції зі стеком **POP HL**

5. Покажчик стека показує на вершину стека (адреса 8238). Вміст цієї комірки виймається і пересилається в молодший байт регістрової пари **HL**.

6. Покажчик стека інкрементується до 8239.

7. Вміст цієї комірки (00000000_2) пересилається в старший байт регістрової пари **HL**.

8. Покажчик стека інкрементується від 8239 до 893A .

Команди **PUSH** і **POP** використовуються завжди сумісно, але між ними розміщуються інші команди, які змінюють дані, що містяться в регістрах мікропроцесора.

Для виклику підпрограм в мові програмування для мікропроцесорів існує команда **CALL**. Вона складається з трьох байт: 1 байт – для коду операції і 2 байти - для вказання адреси першої команди підпрограми. В мнемокодах ця адреса задається символічним ім'ям, наприклад, **MULT1**. Воно є міткою першої команди підпрограми. Команда виклику суміщає операції завантаження в стек інформації, необхідної для повернення в основну програму, і переходу на підпрограму. Її використання показано на рис.4.4. Спочатку вона завантажує в стек вміст лічильника команд, в якому на момент переходу зафіксована адреса команди, наступної після команди виклику. Після цього в лічильник команд завантажується адреса переходу, що задається командою виклику.

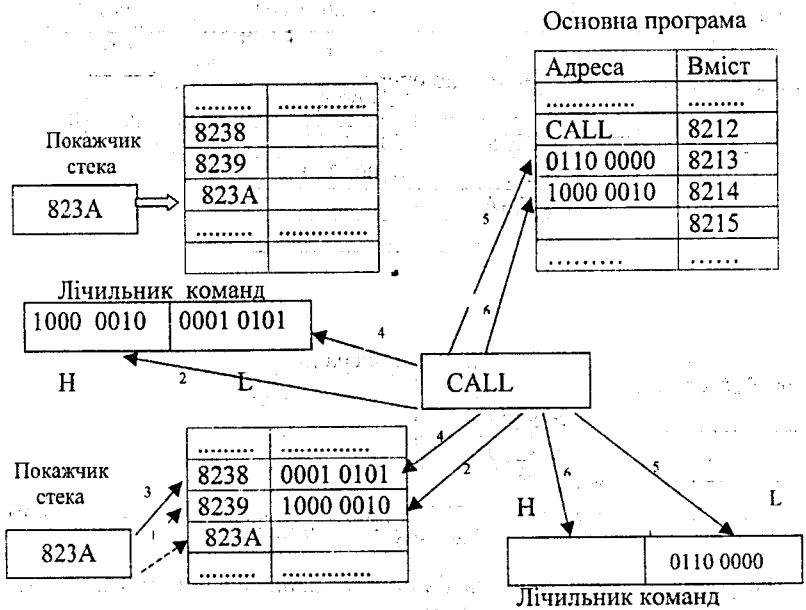


Рисунок 4.4 – Послідовність дій під час виконання команди **CALL**

Послідовність дій помічена на рис. 4.4 номерами над стрілками.

1. Покажчик стека декрементується від 823А до 8229.

2. Старший байт лічильника команд завантажується в стек за адресою 8229.

3. Покажчик стека декрементується від 8239 до 8228.

4. Молодший байт лічильника команд завантажується в комірку пам'яті за адресою 8228.

5. Молодший байт адреси переходу (комірка 8213 основної пам'яті, другий байт команди **CALL**) завантажується в молодший байт лічильника команд.

6. Старший байт адреси переходу (третій байт команди **CALL**) завантажується в старший байт лічильника команд.

Після цього мікропроцесор переходить на виконання команди, адресу якої показує лічильник команд (8260) і яка є адресою першої команди підпрограми.

Останньою командою підпрограми є команда **RET**. За цією командою виконується повернення в основну програму, перервану командою **CALL**. Команда повернення в основну програму **RET** читає послідовно зі стека молодший і старший байти адреси команди основної програми і завантажує їх відповідно в молодший і старший байти лічильника команд. Мікропроцесор продовжує виконання основної програми, починаючи з команди, яка йде після команди **CALL**.

Узагальнена схема взаємодії основної програми і підпрограми показані на рис. 4.5.

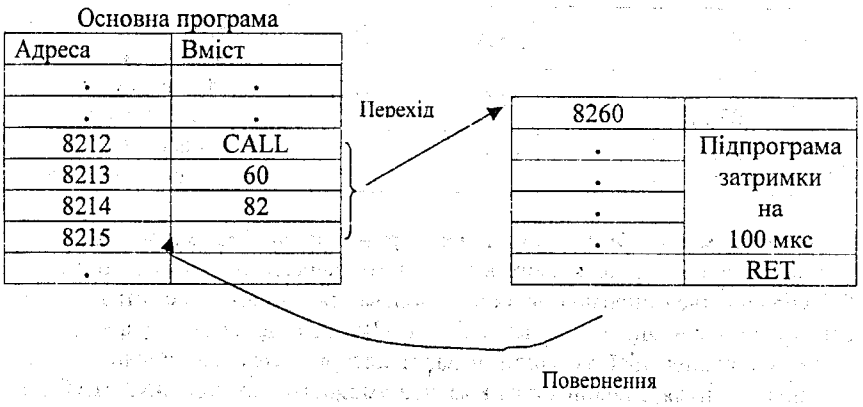


Рисунок 4.5 – Взаємодія основної програми з підпрограмою шляхом використання команд **CALL** і **RET**

4.3 Приклад виконання роботи

У вигляді підпрограми, як правило, оформлюють однотипні дії, що часто повторюються і зустрічаються в різноманітних частинах алгоритму, що реалізується. Такий підхід дозволяє значно зекономити необхідний обсяг пам'яті і зробити основну програму більш зрозумілою. Замість того, щоб неодноразово писати в різних місцях програми цілком однакові послідовності команд, використовують там тільки команди виклику підпрограми, а потрібну послідовність команд оформлюють одноразово у вигляді підпрограми.

Розглянемо підпрограму **ЧАС**, яка здійснює часову затримку в 0,5 сек. (таблиця 4.1).

Таблиця 4.1 - Підпрограма часової затримки

Адреса	Код	Помітка	Мнемоніка	Операнд	Коментар
8200	01,70, CB		LXI	B, КОНСТ	Задати величину затримки 0,5 сек
8203	CD, 07,82		CALL	ЧАС	Виклик підпрограм
8206	E7		RST 32		Виклик монітору
8207	OB	ЧАС	DCX	B	Зменшити на 1 вміст BC
8208	78		MOV A,	B	Переслати в A вміст B
8209	BI		ORA	C	B і C дорівнюють нулю?
820A	C2, 07,82		JNZ	ЧАС	Якщо не нуль, повторити цикл
820D	C9		RET		Повернення в основну програму

По команді **DCX** вміст регістрової пари **BC** мікропроцесора зменшується на 1, а після цього в акумулятор пересилається вміст регістра B і виробляється операція логічного додавання з вмістом регістра C цієї регістрової пари. Якщо в регістровій парі **BC** код ще не став рівним 0, то після виконання цієї команди в акумуляторі виявиться число, також відмінне від нуля, і виконається команда умовного переходу **JNZ ЧАС** до початку підпрограми, всі дії повторюються знов. При цьому говорять, що в підпрограмі організований цикл. Вихід з нього можливий тільки тоді, коли в результаті виконання команди **DCX B** у регістровій парі **BC** виявляться всіх нулі. Тоді робота підпрограми закінчиться виконанням команди **RET** і відбудеться повернення до виконання основної програми.

Мітка **ЧАС** є умовною позначкою початкової адреси підпрограми (місця входу в підпрограму) 8207_{16}.

Часова затримка, що забезпечується підпрограмою **ЧАС**, визначається, по-перше, часом, необхідним для однократного виконання всіх команд цієї підпрограми, і, по-друге, вмістом регістрової пари **BC**. Остання і визначає кількість програмних циклів.

Як визначити число (**КОНСТ**), що треба помістити в регістрову пару **BC** для задання часової затримки в 0.5 секунд?

Виконання будь-якої команди мікропроцесором займає певний час. Тому, знаючи тривалість виконання підпрограми, можна обчислити загальний час однократного виконання підпрограми **ЧАС**. Воно складає 9,6 мкс. Отже, для отримання часової затримки в 0,5 секунд підпрограма **ЧАС** повинна бути виконана $0.5/(9/6\cdot10^{-6}) = 52080$ раз. Отриманий результат необхідно присвоїти операнду **КОНСТ**, записавши його шістнадцятковим числом: $52080_{10} = CB70_{16}$.

В програмі часової затримки (таблиця 2) є одна характерна помилка: перед початком роботи програми необхідно було налагодити регістр **SP**. Налагодити – означає помістити в регістрову пару **SP** адресу пам'яті ОЗП, що не містить кодів команд. Стек використовується в розглянутій програмі (команди **CALL** і **RET**), і тому ми повинні були потурбуватися про вміст регістра покажчика стека.

Перша команда нашої програми повинна бути командою налагодження покажчика стека - **LXI SP, СТЕК** (**СТЕК** - двобайтовий операнд, адреса). Для стека відводяться три осередки пам'яті перед нашою програмою.

Самостійно внести зміни в програму. При цьому треба звернути увагу на те, як відбувається адресація під час роботи зі стеком.

4.4 Контрольні запитання та завдання

1. Розказати про команди виклику підпрограми і повернення з неї.
2. Яким чином управління з основної програми передається на підпрограму, що викликається, і як здійснюється повернення в основну програму?
3. Дати характеристику однобайтовим командам виклику підпрограм, розташованих за фіксованою адресою.
4. Як здійснюється безпосередня і опосередкована адресація пам'яті?
5. Дати означення стека для МП Intel 8080.
6. Як відбувається запис числа в стекову область пам'яті і читання з неї?
7. Розказати про порядок запису шістнадцяткових чисел в пам'ять і внутрішні регістри МП.
8. Скласти підпрограму часової затримки $t_з$=N x n, де N – номер в списку журнала, n – номер бригади.

ЛАБОРАТОРНА РОБОТА №5.
Вивчення принципів роботи з емулятором M8751 мікроконтролера КМ1816ВЕ51 (МК 51)

Мета роботи – вивчення принципів використання програми-емулятора M8751.exe для запису і відлагодження програм на мові Асемблер МК 51.

5.1 Порядок виконання роботи

5.1.1 Ознайомитися з можливостями та інтерфейсом емулятора.
5.1.2 Завантажити емулятор, вивчити усі режими роботи з програмами.
5.1.3 Записати текст запропонованої програми в режимі редагування.
5.1.4 Відтранслювати, відлагодити та завантажити програму в емулятор.
5.1.5 Занести в пам'ять емулятора потрібні початкові дані.
5.1.6 Виконати програму в покроковому та неперервному режимах.
5.1.7 Проаналізувати результати та стан кожного з пристроїв емулятора.

5.2 Теоретичні відомості

5.2.1 Характеристики емулятора M8751

Повноекранний налагоджувач-емулятор M8751 для програм, написаних мовою Асемблера однокристальних мікро-ЕОМ КМ1816ВЕ51/КР1816ВЕ35 (MKS 51) забезпечує повний набір можливостей для запису та логічного налагодження програм. Завантажену програму можна виконати по кроках і в безперервному режимі з зупинкою в контрольних точках, що задаються користувачем. Забезпечується трасування виконання програми, друк дизасемблерного тексту, дампів пам'яті даних і програм. По ходу виконання програми можна оперативно коректувати вміст ОЗП, регістрів і вносити зміни в саму програму в мнемонічних позначеннях мови Асемблера і машинних кодах. Налагоджувач має лічильник часу виконання завантаженої програми. Передбачається, що тактова частота дорівнює 12 Мгц. Для іншої частоти перерахування виконується простим множенням на коефіцієнт. Після запуску програми-емулятора M8751.exe на екрані з'являється його віконний інтерфейс, зображений на рис.5.1. В рамці під словом **"Программа"** відображено вміст пам'яті програм, а в рамках під написом **"Внутренние ресурсы ОЭВМ"** відображено вміст внутрішніх пристроїв мікроконтролера: портів введення-виведення **P0÷P3**, регістрів **R0÷R7** загального призначення простору пам'яті RSEG, 128-байтової резидентної пам'яті даних (РПД, простір DSEG), регістрів спеціального призначення **PC, PSW, Acc (A), B, IE, SP, IP, DPTR (DPH, DPL), PCON, TCON, TMOD, SCON, SBUF, T0**

(TH0,TL0), T1 (TH1,TL1), призначення яких описано в лабораторній роботі №6. Також відображено значення бітів регістра прапорців **F1: CY, AC, F0, F1, OV, P.**

В нижній рамці вікна відображені пункти меню екранного інтерфейсу, які дозволяють вибирати різні режими роботи емулятора.

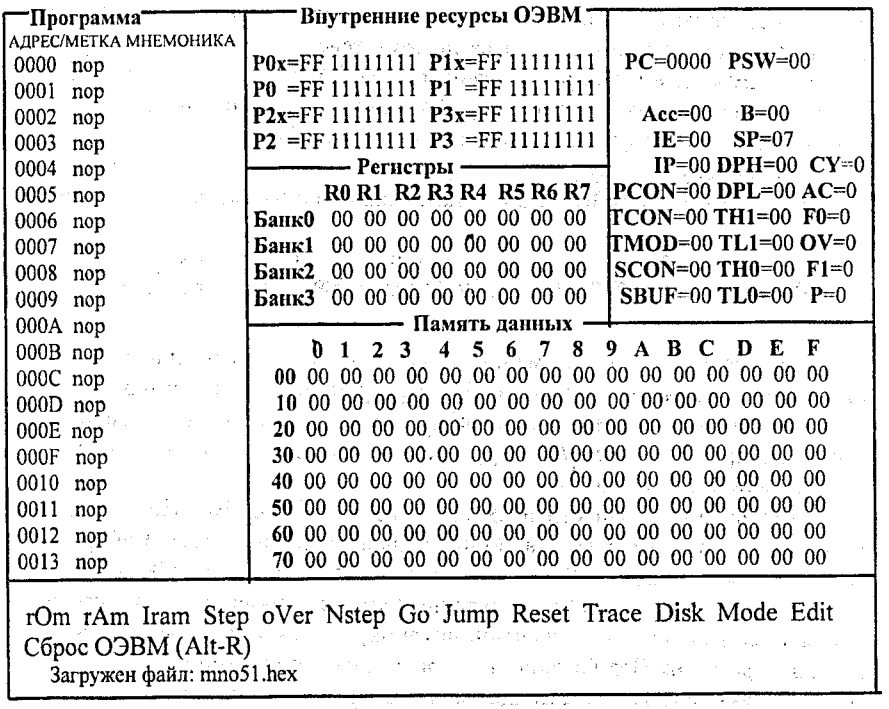


Рисунок 5.1 – Екранне вікно емулятора M8751

5.2.2 Інтерфейс емулятора

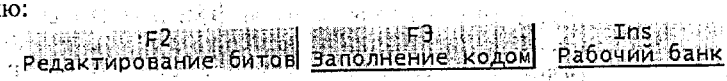
Робота з віконним інтерфейсом емулятора здійснюється через пункти його головного меню. Режим завантаження відкомпільованої програми обирається через пункт **Disk** наведенням на нього курсора (світлої рамки) за допомогою клавіш "←" і "→" клавіатури. Натискання клавіші *Enter* викликає допоміжне меню:

Enter	W	T	D	F6
Чтение файла	Запись файла	Запись текста	Выбор диска	Тип экрана

Вибираємо (якщо це не було зроблено раніше) пункт "Вибір диска" і пишемо (вручну і по пам'яті) шлях до каталогу, де знаходиться емулятор і програмні файли.

Після цього у вікні в центрі екрана з'явиться стандартний список файлів, з якого і вибираємо потрібну програму за допомогою клавіш "←" і "→". В разі успіху з'явиться повідомлення "програма завантажена", і "Ok" в рамці буде чекати підтвердження – натисканням клавіші *Enter*. Тепер на екрані емулятора бачимо в лівій рамці *.asm*-текст завантаженої програми, а внизу - її ім'я (текст "Загружен файл: mno51.hex"). В пам'яті даних будуть записані якісь випадкові числа, тому її потрібно очистити. Вихід з вибраного режиму роботи здійснюється натисканням клавіші *Esc*.

Для вибору режиму роботи з внутрішньою пам'яттю даних обираємо в головному меню пункт **Iram**, після натискання якого з'являється допоміжне меню:



Вибираємо в цьому меню пункт *F3*, з'являється діалог-бокс, де вибираємо "записати усюди нулі". Програма-емулятор дозволяє перед початком роботи встановити в потрібних комірках РПД задані значення. Для цього вибираємо пункт *F2*. Курсором заходимо у потрібні комірки і встановлюємо задані значення.

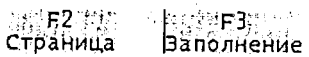
Тепер можемо запустити програму на виконання. Виходимо з режиму редагування бітів (*Esc* як і в інших аналогічних випадках). Вибираємо в головному меню пункт **Step**, дивимось на всі елементи МК - регістри, спеціальні регістри, натискаємо клавішу *Enter*, дивимося, що і як змінилося, порівнюємо з попередніми значеннями, знову натискаємо клавішу *Enter* і так до кінця програми. При бажанні повторити прогін програми вибираємо пункт головного меню **Reset**, який скидає мікроконтролер в початковий стан .

Для входу в режим редагування пам'яті програми використовуємо пункт головного меню **rOm**. Після його активізації з'являться такі пункти допоміжного меню:

Для вибору номера сторінки використовуємо клавішу *F2*. Для заповнення пам'яті програми використовуємо клавішу *F3*. Тепер можна редагувати пам'ять програми, яка відображається в правій частині екрану. Використовуючи кнопку *F5* можна очистити пам'ять, для чого потрібно

вказати початкову і кінцеву адреси, після цього натиснути бокс Ok. Використовуючи кнопку *F6* можна зміститися у відповідному діапазоні на потрібну адресу і підтвердити її натисканням боксу Ok. Пошук заданої адреси пам'яті у відповідному діапазоні здійснюється активізацією кнопки *F10*, і підтверджується натисканням на Ok.

Для входження в режим редагування пам'яті даних активізуємо в головному вікні емулятора пункт **rAm**. Після його активізації з'являться такі пункти допоміжного меню:



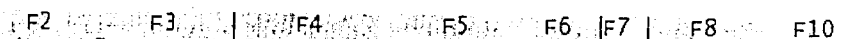
Для вибору номера сторінки використовуємо кнопку *F2*. Для заповнення пам'яті даних використовуємо кнопку *F3*. Тепер можна редагувати пам'ять даних, яка відображається в правій частині екрана.

Для простеження ходу виконання програми через декілька кроків вибираємо в головному меню пункт **Nstep**.

Для встановлення адреси пуску програми використовуємо в головному меню пункт **Go**, де вказуємо початкову адресу початку запуску програми.

Для встановлення адреси переходу використовуємо в головному меню пункт **Jump**.

Для наочного перегляду ходу виконання програми використовуємо в головному меню пункт **Trace**, після входу в який з'являються такі діалогові вікна:



F2 – встановлення відповідної адреси перегляду;

F3 – заповнення відповідного діапазону потрібним кодом;

F4 – виконання програми до рядка, на який вказує курсор;

F5 – очистка контрольних точок у відповідному діапазоні;

F6 – зсув діапазону;

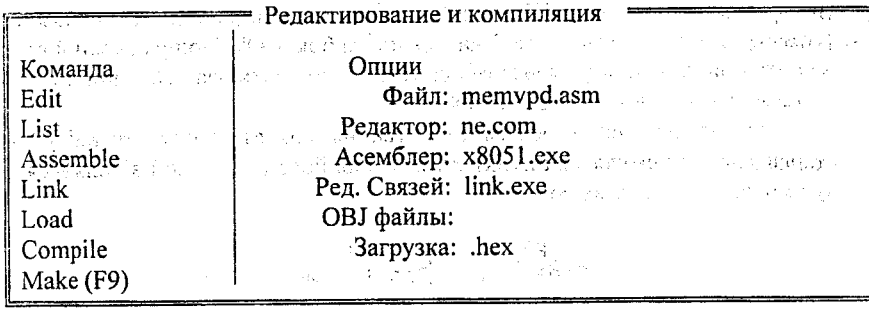
F7 – виконання програми по кроках;

F8 – виконання процедурно;

F10 – пошук заданого коду на відповідному діапазоні.

Запис тексту програми. Текст програми записується в довільному текстовому редакторі, що підтримує формат .txt, згідно з стандартом запису програм на Асемблері, який вимагає чітко розділяти мітки, команди, операнди, коментарі. Ім'я текстового файла повинне мати розширення .asm.

Програму можна безпосередньо написати і в емуляторі. Це робиться таким чином: вибираємо пункт головного меню **Edit**, який відкриє таке вікно:



Вибираємо в цьому вікні пункт **Edit**, який відкриє вікно редактора ne.com, в якому записується текст програми, натискається клавіша F2 (Save), після чого програмі присвоюється ім'я з розширенням .asm. Потім послідовно вибираємо у вікні пункти Assemble, Link, Load, Compile. Після виконання цих дій висвітиться знайоме нам вікно емулятора. В лівій верхній частині вікна буде завантажена відкомпільована програма.

Для завантаження уже наявної на диску програми за допомогою клавіші табуляції переходимо в праву частину вікна і вписуємо послідовно - ім'я програми, ім'я редактора, ім'я компілятора, ім'я лінковщика, якщо ці програми обробки не встановлені.

Потім знову ж клавішею табуляції повертаємось в ліву частину і послідовно вибираємо пункти Assemble, Link, Load, Compile.

5.3 Приклад запису і відлагодження програми на мові Асемблер МК 51 з допомогою емулятора М8751

Вивчення емулятора проведемо на прикладі роботи з готовою програмою MULT51.EXE множення багатобайтового числа на константу.

Подаємо її текст на асемблері. Через пункт меню Edit можна записати і відкомпілювати цей текст, але зараз ми для вивчення емулятора використаємо вже відлагоджену і відкомпільовану програму MULT51.HEX.

Приклад: Помножити ціле двійкове багатобайтове число **P1** на константу **K1**. Програма попередньо повинна розмістити вхідне число в РПД, починаючи з адреси **A1**, що також попередньо завантажена до регістра **R0**. Результат розмістити на місце заданого числа **P1**. Число байтів **N** в числі записано регістрі **R2**. Нехай **K1**=11H, а перший байт числа **P1** розміщений в комірці РПД з адресою 20H.

Текст програми:

```
mov PSW, #00H   ; вибір нульового банку регістрів
```

```
        mov R0, #20H    ;завантаження в регістр R0 1-го байта множеного
        mov R1, #03H    ;завантаження в регістр-лічильник число циклів
        mov A,#0        ; скидання акумулятора
LOOP: add A,@R0         ; завантаження множеного
        mov B,#11H      ; завантаження множника
        mul AB          ; множення
        mov @R0,A       ; запис молодшого байта часткового добутку
        inc R0          ; приростення адреси
        mov A,B         ; пересилання старшого байта часткового добутку в
                        ;акумулятор
        xch A,@R0       ; попереднє формування чергового байта добутку
        djnz R1, LOOP   ; перевірка умови продовження циклу множення
        .end
```

Отриманий добуток розміщується на місці множеного та займає в РПД на один байт більше.

Завантажуємо відкомпільовану програму в емулятор. Вибираємо пункт **Disk** в головному меню емулятора (нижній рядок на екрані). Натискуємо *Enter* і маємо відповідне меню:

Enter Чтение	*W* Запись	*T* Запись	*D* Выбор диска	*F6* Тип экрана

Вибираємо (якщо це не було зроблено раніше) пункт "Выбор диска" і пишемо (вручну і по пам'яті) шлях до каталогу, де знаходиться емулятор і програмні файли.

Після цього у вікні в центрі екрана з'явиться стандартний список файлів, з якого і вибираємо потрібну програму - в нашому випадку це MULT51.HEX. В разі успіху з'явиться повідомлення "програма завантажена", і "Ok" в рамці буде чекати підтвердження - натискання клавіші *Enter*.

Тепер на екрані емулятора бачимо в лівій частині .ASM-текст завантаженої програми, а внизу — її ім'я.

В пам'яті даних будуть записані якісь випадкові числа. Треба її очистити.

Вибираємо в головному меню пункт IRAM і тоді з'являється внизу таке меню:

F2 Редактирование битов	*F3* Записать везде нули	*Ins* Рабочий банк

Вибираємо в цьому меню пункт F3, з'являється діалог-бокс, де вибираємо "записать везде нули".

Задана програма спеціально написана так, щоб нам потрібно було перед початком її роботи встановити в потрібних комірках РПД потрібні значення. Для цього вибираємо пункт F2.

Курсором заходимо у потрібні комірки і встановлюємо потрібні значення, наприклад: в комірках **20, 21, 22** записуємо трибайтове число, наприклад **11, 0F, 0A**.

Тепер можемо запустити програму на виконання. Виходимо з режиму редагування бітів (*Esc*, як і в інших аналогічних випадках).

Вибираємо в головному меню пункт **Step**, дивимось на всі елементи МК - регістри, спеціальні регістри, натискуємо *Enter*, дивимося, що і як змінилося, порівнюємо, аналізуємо, знову натискуємо і так до кінця програми.

При бажанні повторити прогон програми вибираємо пункт **Reset**.

5.3 Контрольні запитання та завдання

1. Описати віконний інтерфейс емулятора M8751.
2. В яких режимах може працювати емулятор?
3. Яке призначення пунктів головного меню емулятора?
4. Розкажіть про порядок запису, компіляції, завантаження та відлагодження програми на емуляторі.
5. Які режими виконання програми на емуляторі вам відомі?
6. Записати текст наведеної вище програми множення в режимі редагування. Варіанти вхідних даних до задачі наведені в табл..5.1:

Таблиця 5.1 – Варіанти завдань

№ вар.	1	2	3	4	5	6
K1	120	17	28	56	71	84
N	2	3	2	3	3	4
Число	1F04 H	23DE10 H	1121 H	FFB121 H	11012C H	5DBC7611 H
A1	30 H	24 H	10 H	17 H	26 H	32 H

7. Відтранслювати, відлагодити та завантажити програму в емулятор.
8. Занести в пам'ять емулятора потрібні початкові дані (Табл. 5.1).
9. Виконати програму в покроковому та неперервному режимах.
10. Проаналізувати результати та стан кожного з пристроїв емулятора.

ЛАБОРАТОРНА РОБОТА №6
Архітектура мікроконтролера КМ1816ВЕ 51 (МК 51). Організація пам'яті, команди передачі даних

Мета роботи – вивчення структури мікроконтролера КМ1816ВЕ51, організації пам'яті і форматів його команд, а також придбання початкових навичок програмування в кодах мікроконтролера з використанням команд передачі даних.

6.1 Порядок виконання роботи

6.1.1 Ознайомитися з теоретичними відомостями, наведеними в даних методичних вказівках.

6.1.2 Для заданого варіанта завдання (таблиця 6.3) скласти програму в мнемокодах мови Асемблер МК-51, відлагодити її і обраховувати за допомогою емулятора.

6.1.2 Скласти звіт про виконання роботи.

6.2 Теоретичні відомості

6.2.1 Архітектура мікроконтролера

Мікроконтролер (МК) КМ1816ВЕ51 (аналог мікроконтролера МК51) відноситься до класу однокристальних мікро-ЕОМ і призначений для побудови нескладних цифрових систем керування. Мікроконтролер виконаний на базі **n**-МОН технології, упакований у корпус із дворядним розташуванням виводів НВІС (має 40 контактів). ВІС живиться від одного джерела напругою 5В. Структура МК приведена на рис.6.1.

Операційна частина. До складу операційної частини МК включені такі пристрої:

- АЛП, що виконує 51 різноманітну операцію над бітами, восьмирозрядними двійковими і десятковими числами;
- восьмирозрядні регістри тимчасового зберігання Т1, Т2;
- акумулятор А;
- розширювач акумулятора В (для виконання операцій множення і ділення);
- регістр слова стану програми PSW, призначення бітів якого наведено в табл.6.1

Пам'ять. Мікропроцесор містить:

RAM - резидентну пам'ять даних (РПД) об'ємом 128 байт, доступ до якої здійснюється через регістр адреси **RAR** або покажчик стека **SP**;

EPROM - вмонтовану перепрограмовувану пам'ять програм, під час вибірки команд із якої використовується 16-розрядний покажчик адреси **PC**. Якщо з пам'яті команд вибираються дані (константи), то для адресації використовується 16-розрядний покажчик адреси **DPTR** (молодший байт адреси заноситься в **DPL**, а старший у **DPH**).

Описання пам'яті приведено в п.2 даної лабораторної роботи.

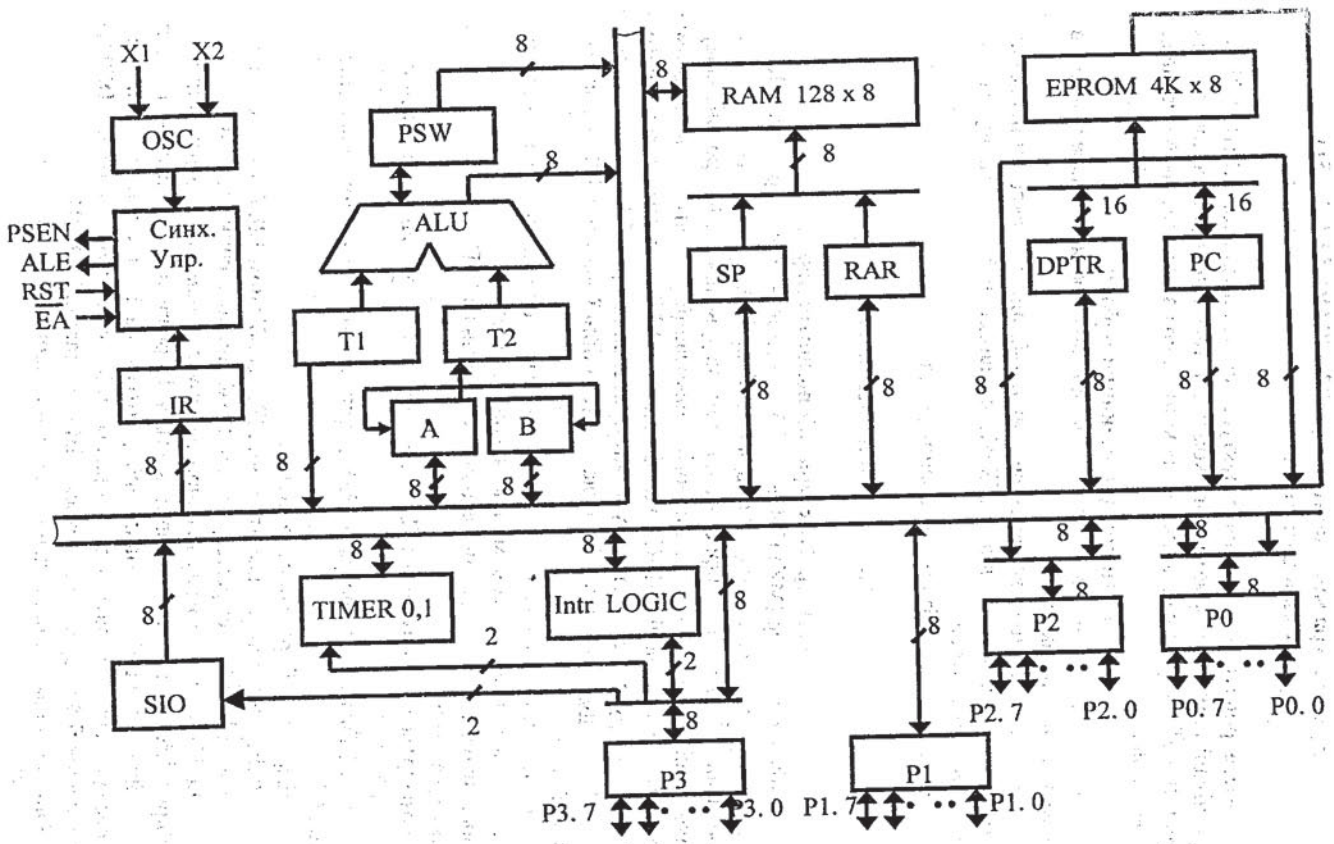


Рисунок 6.1 – Структура мікроконтролера

Таблиця 6.1 - Призначення бітів PSW

Розряд	Символ	Призначення
PSW. 7	**C**	Перенесення із старшого розряду АЛП
PSW. 6	**AC**	Десяткове перенесення з молодшої тетради АЛП
PSW. 5	**FO**	Прапорець користувача загального призначення
PSW. 4	**RS1**	Вибір банку регістрів
PSW. 3	**RS0**	Вибір банку регістрів
PSW. 2	**OV**	Арифметичне переповнення результату
PSW. 1		Не використовується
PSW. 0	**P**	Паритет - парність вмісту регістра А

Пристрій керування. До складу пристрою керування МК входять:
- **IR** - регістр команд, у якому зберігається код виконуваної команди;
- **OSC** - вмонтований генератор синхроімпульсів **X1**, **X2**;
- пристрій синхронізації і керування роботою МК.

Зовнішні пристрої . МК має 4 двонаправлених порти **P0÷P3** для підключення зовнішніх пристроїв, причому порт **P3** може виконувати альтернативні функції. До складу МК також входять:

- два шістнадцятирозрядних таймери/лічильники для часової синхронізації обчислювальних процесів;
- послідовний прийомо-передавач **SIO**;
- схема обробки внутрішніх і зовнішніх переривань.

6.2.2 Формати команд

Система команд МК нараховує 111 команд, серед яких: 49 однобайтових, 45 двобайтових і 17 трибайтових. Використовується чотири основних методи адресації:

- Регістрова адресація. Операнд знаходиться в одному з регістрів загального призначення **R0÷R7** вибраного банку регістрів. Регістрова адресація дозволяє в однобайтовій команді вказати код операції і адресу операнда. Оскільки операнд знаходиться у внутрішній комірці, то не потрібні цикли звертання до зовнішньої пам'яті даних. Наприклад, команда **ADD A, R1** додає вміст регістра **R1** до вмісту акумулятора;
- Пряма адресація. Операнд знаходиться у внутрішній пам'яті даних, адресу операнда визначає окремий байт команди. Таким чином, команда займає мінімум два байти, доступними для адресації є 256 комірок, звертання до зовнішньої пам'яті за операндом не потрібне. Наприклад, команда **ADD A, 2EH** додає вміст комірки внутрішньої пам'яті з адресою **2E** до вмісту акумулятора;

- Опосередковано-регістрова адресація. Цей метод адресації використовує регістри **R0** і **R1** як покажчики адреси. Регістри беруться з банку, визначеного розрядами **RS0, RS1** регістра **PSW**. Вміст цих регістрів використовується як адреса для звертання до комірок внутрішньої пам'яті даних, в яких зберігається операнд. Наприклад, команда **ADD A, @R0** додає до вмісту акумулятора операнд, який зберігається у внутрішній пам'яті даних за адресою, записаною в регістрі **R0**;

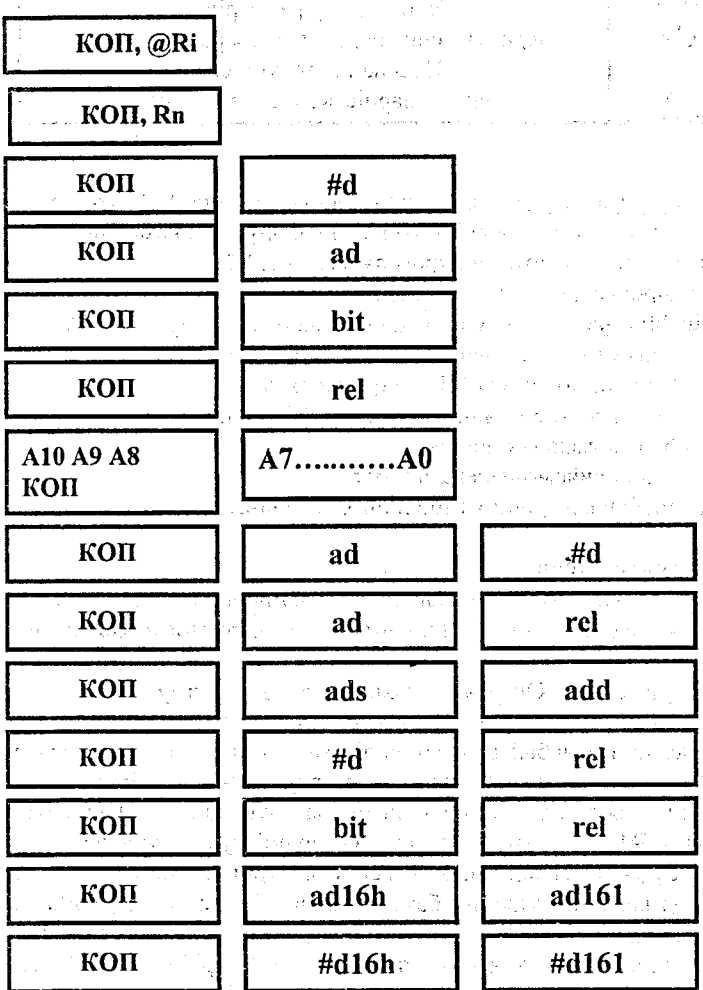


Рисунок 6.2 - Формати команд мікроконтролера

- Безпосередня адресація. Операнд у вигляді константи записаний в окремому байті команди. Таким чином, команда займає мінімум два байти, звертання за операндом до пам'яті не потрібне. Наприклад, команда **ADD A, #2FH** додає до вмісту акумулятора константу **2F**;

Формати команд МК приведені на рис.6.2. В наведених форматах використані такі скорочення:

#d - безпосередній операнд;

ad - адреса операнда в DSEG;

bit - адреса біта в BSEG;

rel - відносна адреса в CSEG;

ad16 – 16 розрядна адреса переходу в командах **LCALL** і **LJMP**;

#d16 - безпосередній 16-розрядний операнд;

ads, add – 8-розрядні безпосередні адреси джерела і приймача в DSEG.

6.2.3 Організація пам'яті

В архітектурі мікроконтролера МК 51 використано окремо пам'ять програм і пам'ять даних (гарвардська архітектура). Кожна з них має розмір 64 Кбайти. Пам'ять мікроконтролера можна подати як п'ять логічно адресованих просторів, відображених на 3 фізичних пристроях пам'яті (внутрішні ROM і RAM, зовнішні ЗП). Організація пам'яті в мікроконтролерах сімейства 8051 показана на рис.6.3.

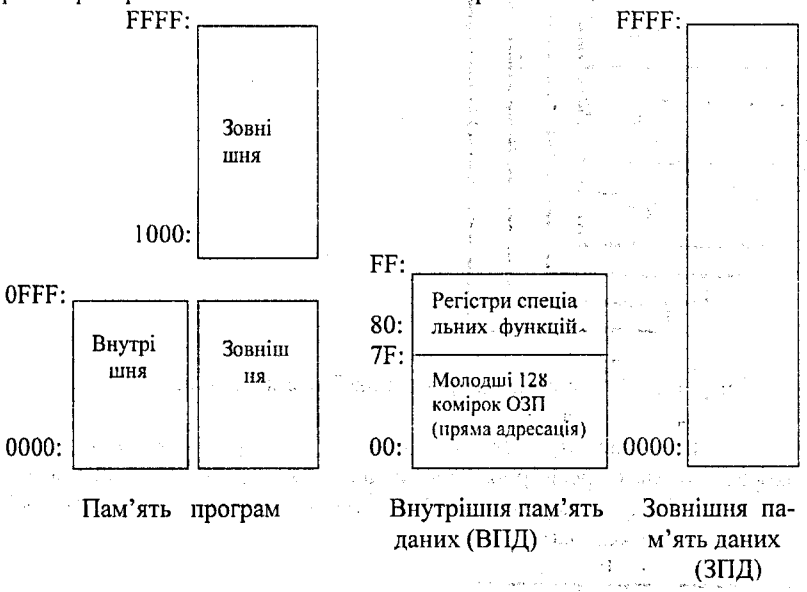


Рисунок 6.3 – Організація пам'яті в архітектурі МК 51

Простір RSEG. Простір внутрішніх регістрів включає 32 восьмирозрядних регістри, об'єднаних в 4 банки по 8 регістрів у кожному. Банки регістрів (рис.6.3) відповідно позначаються - **RBO, RB1, RB2, RB3** і займають область ВПД 00÷1F.

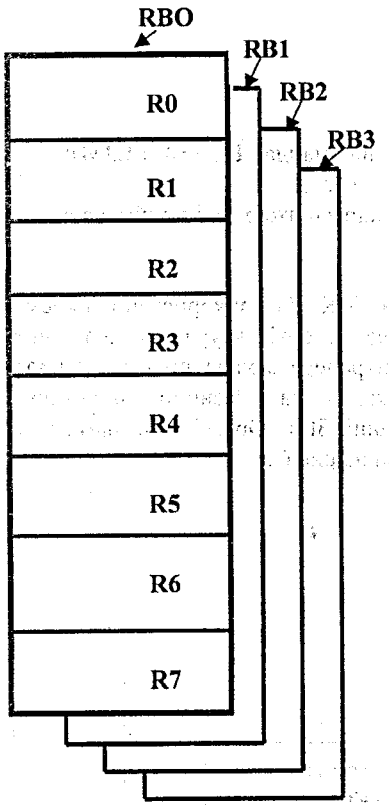
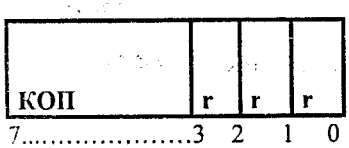


Рисунок 6.4 – Організація регістрової пам'яті даних

Для доступу до будь-якого регістра поточного банку використовується регістрова адресація, при цьому номер регістра поточного банку вказується в 3 молодших розрядах першого байта команди - в полі Rn.



rrr = Rn, n = 0,1,2,3,4,5,6,7

Поточний банк регістрів вибирається за допомогою 2-розрядного поля **RS** у **PSW**.

Простір DSEG.

Простір внутрішньої пам'яті даних має об'єм 256 байтів. Для доступу до комірки простору DSEG використовується два способи адресації. Перший спосіб - пряма адресація, при цьому в команді вказується пряма восьмирозрядна адреса **ad**. В прямій адресації доступні 128 молодших байтів простору DSEG або спеціальні регістри. Якщо старший (сьомий) розряд прямої адреси дорівнює 0, відбувається звертання до однієї з 128 комірок внутрішньої пам'яті. Якщо старший розряд прямої адреси дорівнює 1, відбувається звертання до одного із спеціальних регістрів. Якщо при цьому використовується адреса, що не приписана ні до одного спеціального регістра, то результат дії команди не визначений. В двоадресних командах використовується пряма адреса джерела операнда **ads** і пряма адреса приймача операнда **add**. Другий спосіб - непряма адресація через регістри **R0** або **R1** поточного банку регістрів.

@Ri, i = 0,1

В непрямій адресації до простору DSEG можливий доступ тільки до 128 молодших адрес простору. Якщо вміст регістра **R0** (**R1**) під час непрямого звертання до DSEG перевершує 7FH, то результат операції не визначений.

Простір BSEG.

Простір BSEG (пам'ять бітового процесора) уявляє собою однорозрядний лінійно-упорядкований простір пам'яті ємністю 256 біт, під який використовуються 16 комірок внутрішньої пам'яті даних з адресами 20÷2F (це складає 16 х 8=128 біт), а також 128 біт регістрів спеціального призначення.

У просторі BSEG використовується тільки пряма адресація, пряма восьмирозрядна адреса BSEG позначається **bit**. Робота BSEG докладно розглядається в лабораторній роботі № 8.

Простір зовнішньої пам'яті даних XSEG.

Простір зовнішньої пам'яті даних являє собою лінійний адресний простір обсягом 64 Кбайт. Адреса в XSEG може приймати значення від 0000H до FFFFH (рис.6.3).

Використовується два способи адресації.

Перший спосіб - непряма адресація, через шістнадцятирозрядний регістр-покажчик **DPTR**. Така адресація можлива тільки в двох командах **MOVX A, @DPTR** і **MOVX @DPTR, A**.

Другий спосіб - непряма сторінкова адресація, при цьому номер сторінки задається вмістом порту **P2**, а зсув у сторінці - вмістом регістра **R0** (або **R1**) поточного банку, тобто адреса в XSEG знаходиться як конкатенація **(P2) | (Ri)**. Для цієї мети можна використовувати тільки дві команди - **MOVX A, @Ri** або **MOVX @Ri, A**.

Простір пам'яті програм CSEG.

Об'єм простору пам'яті програм 64 Кбайт. Він побудований як однорідний лінійний простір із двома основними способами передачі керування. Перші (молодші) 4 Кбайт фізично належать EPROM усередині МК. Інші 60 Кбайт - реалізуються зовнішнім ЗП (передача керування докладно розглядається в лабораторній роботі № 7).

До простору CSEG можливо також звертання як до джерела операндів (констант). Для цього використовується безпосередня адресація і базово-індексна адресація через регістри **DPTR** або **PC** із змінним зміщенням.

Під час безпосередньої адресації операнд (константа) вказується в коді команди. Можливі команди з 8-розрядним операндом (**#d8**) і 16-розрядним операндом (**#d16**).

Непряма адресація можлива тільки за допомогою двох команд. Команда **MOVC A, @A+DPTR** зчитує в акумулятор із CSEG за адресою, яка дорівнює сумі вмісту **A** (зміщення, індекс) і **DPTR** (база). Команда **MOVC A, @A+PC** зчитує в акумулятор байт із CSEG за адресою, знайденій як сума вмісту **A** і **PC**.

6.2.4 Команди передачі даних

У МК широко подані команди пересилки даних, велику частину яких складають команди передачі й обміну байтів. Всі команди даної групи не модифікують прапорці результату, за винятком команд завантаження **PSW** і акумулятора. Мнемоніка і коди операцій команд передачі даних наведені в табл. 6.2. Літери Б і Ц в назвах третьої і четвертої колонок означають кількість байтів в команді і кількість машинних циклів, які витрачаються на виконання команди.

Таблиця 6.2 Група команд передачі даних

МНЕМОНІКА	КОП	Б	Ц	Операція
MOV A, Rn	11101rrr	1	1	(A) :=(Rn)
MOV A, ad	11100101	2	1	(A) :=(ad)
MOV A, @Ri	1110011i	1	1	(A) :=((R))
MOV A, #d	01110100	2	1	(A) :=#d
MOV Rn, A	11111rrr	1	1	(Rn) :=(A)
MOV Rn, ad	10101rrr	2	2	(Rn) :=(ad)
MOV Rn, #d	01111rrr	2	1	(Rn) :=#d
MOV ad, A	11110101	2	1	(ad) :=(A)
MOV ad, Rn	10001rrr	2	2	(ad) :=(Rn)
MOV add, ads	10000101	3	2	(add) :=(ads)
MOV ad, @Ri	0000011i	2	2	(ad) :=((Ri))
MOV ad, #d	01110101	3	2	(ad) :=#d
MOV @Ri, A	1111011i	1	1	((Ri)) :=(A)
MOV @Ri, ad	0110011i	2	2	((Ri)) :=(ad)
MOV @Ri, #d	0111011i	2	1	((Ri)) :=#d
MOV DPTR, #d16	10010000	3	2	(DPTR) :=#d16
MOVC A, @A+DPTR	10010011	1	2	(A) :=((A)+(DPTR))
MOVC A, @A+PC	10000011	1	2	(PC) :=(PC)+1, (A):=((A)+(PC))
MOVX A, @Ri	1110001i	1	2	(A) :=((P2)*(Ri))
MOVX A, @DPTR	11100000	1	2	(A) :=((DPTR))
MOVX @Ri, A	1111001i	1	2	((P2)*(Ri)) :=(A)
MOVX @DPTR, A	11110000	1	2	((DPTR)) :=(A)
PUSH ad	11000000	2	2	(SP):=(SP)+1,((SP)):=(ad)
POP ad	11010000	2	2	(ad):=((SP)), (SP):=(SP)-1
XCH A, Rn	11001rrr	1	1	(A) <-> (Rn)
XCH A, ad	11000101	2	1	(A) <-> (ad)
XCH A, @Ri	1100011i	1	1	(A) <-> ((Ri))
XCHD A, @Ri	1101011i	1	1	(A[0-3]) <-> ((Ri[0-3]))

6.3 Приклад виконання роботи

За приклад застосування команд передачі даних розглянемо таку задачу.

У регістрах банку **RB1** резидентної пам'яті даних (РПД) знаходяться К1 байтів даних. Потрібно переписати ці дані у зовнішню пам'ять даних (ЗПД), починаючи з адреси **ADDR**. Перед виконанням програми потрібно попередньо записати за довільними (допустимими) адресами у РПД К1=8

довільних однобайтових чисел. Виберемо початкову адресу для запису у ЗПД **ADDR=3000H** і банк пам'яті РПД - №1.

Версія програми

```
MOV PSW, #0100B      ; вибір банку регістрів RB1
MOV R0, #8           ; завантаження лічильника
MOV DPTR, #3000H     ; визначення початкової адреси ЗПД
MOV R1, #0           ; визначення початкової адреси РПД
LOOP: MOV A, @R1     ; завантаження в акумулятор з комірки, на яку
                     ; вказує число в регістрі R1
MOVX @DPTR, A        ; передача з акумулятора у ЗПД за адресою, на
                     ; яку вказує число в DPTPR
INC R1               ; інкремент адреси в РПД
INC DPTR             ; інкремент адреси ЗПД
DJNZ R0, LOOP        ; цикл, поки не перепишемо усі байти з банку
                     ; пам'яті РПД
.END                 ; кінець програми - директива для транслятора
```

Таким чином, після встановлення банку пам'яті, початкових адрес РПД, ЗПД та кількості байтів, починається цикл читання чергового байта в акумулятор і передачі його за допомогою команди **MOVX** у відповідну комірку ЗПД. Оскільки дані і знаходяться і передаються за опосередкованими адресами, то чергові адреси для читання та передачі формуються за допомогою команди інкрементування поточного змісту відповідних регістрів - **INC**.

6.4 Контрольні запитання та завдання

1. Опишіть структуру мікроконтролера МК 51.
2. Яке призначення портів введення-виведення P0÷P3?
3. Які пристрої входять до операційного блоку мікроконтролера?
4. Розкажіть про пристрої керування МК 51.
5. Розкажіть про призначення бітів слова стану програми PSW.
6. Як організована пам'ять мікроконтролера МК 51.
7. Як влаштована регістрова пам'ять МК?
8. Які способи адресації використано в системі команд МК 51?
9. Опишіть формати команд МК.
10. Запишіть групу команд передачі даних МК 51.
11. Завдання на лабораторну роботу: запишіть в кодах МК 51 програму, що:

- записує в комірку RAM1 константу CONST1;
- записує в комірку RAM2 константу CONST2;

- переписує вміст RAM1 у регістр із номером X банку регістрів номер Q;
- переписує вміст RAM2 у регістр із номером Y банку регістрів номер W;
- початкова адреса програми ADR1.

Таблиця 6.3 - Варіанти завдань

Номер	RAM1	RAM2	CONST1	CONST2	X	Bank Q	Y	Bank W	ADR1
01	71	43	FE	CA	0	0	2	3	0714
02	62	54	FF	AB	1	1	3	2	062F
03	53	62	FD	BC	2	2	4	1	053E
04	44	71	FC	CD	3	3	5	0	044A
05	35	12	FB	DE	4	1	6	3	0355
06	26	23	FA	EF	5	2	7	2	0266
07	17	34	F1	FF	6	3	0	0	0177
08	78	45	F2	1A	7	0	1	3	078A
09	69	56	F3	2B	3	1	2	2	069D
10	5A	67	F4	3C	0	2	3	1	05AF
11	4B	78	F5	4D	1	3	4	0	04Ba
12	3C	23	F6	5E	2	0	5	3	03C9
13	2D	3A	F7	6F	3	1	6	2	02D8
14	1E	4B	F8	7A	4	2	7	1	01E4
15	7F	5C	F8	8B	5	3	0	0	07F5
16	6E	6D	F9	9C	6	0	1	3	06E6
17	5A	7E	F1	2D	7	1	2	2	05A1
18	4D	1F	F2	5E	0	2	3	1	04D3
19	3C	2E	F3	6F	1	3	4	0	03C8
20	26	3D	F4	7A	2	0	5	3	026F
21	45	9A	FF	8B	3	1	6	3	04BE
22	67	8d	1F	9D	4	2	7	3	03CA
23	4F	4E	2E	A0	5	3	7	2	02D3
24	5D	3B	3D	B1	6	3	6	1	01E7
25	3A	71	4C	C2	7	2	5	0	07F3
26	9E	69	5B	D3	1	1	4	2	06E5
27	3F	9E	6A	E4	2	0	3	1	05AD

ЛАБОРАТОРНА РОБОТА №7

Команди передачі керування. Організація циклів на асемблері МК 51

Мета роботи - вивчення організації простору пам'яті програм мікроконтролера КМ 1816ВЕ51, програмних засобів керування ходом виконання програми, придбання навичок програмування циклічних алгоритмів у кодах мікроконтролера.

7.1 Порядок виконання роботи

7.1.1 Ознайомитися з теоретичними відомостями, наведеними в даних методичних вказівках.

7.1.2 Для заданого викладачем варіанта задачі скласти програму в мнемокодах мови Асемблер МК 51.

7.1.3 Завантажити програму в емулятор М8751 та виконати в покроковому режимі.

7.1.4 Зафіксувати та проаналізувати отримані результати.

7.1.5 Оформити звіт про виконання лабораторної роботи.

7.2 Теоретичні відомості

7.2.1 Простір пам'яті команд CSEG

Для збереження програм і незмінних даних у МК 1816ВЕ51 використовується логічний однорідний лінійний простір пам'яті CSEG об'ємом 64 Кб. Пам'ять програм адресується 16-розрядним лічильником **PC**. Молодші 4 Кб цього простору відповідають умонтованому EPROM мікроконтролера, інші 60 Кб реалізуються зовнішніми, щодо МК, схемами.

У просторі CSEG виділяються такі точки (адреси):

0000 H - **RESET** - стартова адреса при скиданні системи;

0003 H – **EXTI0** - зовнішнє переривання 0;

000B H - **TIMER0** - переривання таймера/лічильника 0;

0013 H - **EXTI1** - зовнішнє переривання 1;

001B H - **TIMER1** - переривання таймера/лічильника 1;

0023 H - **SINT** - переривання послідовного порта;

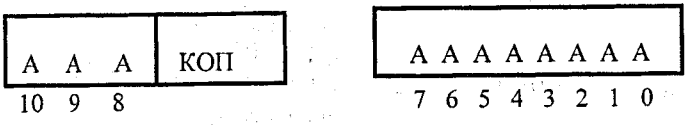
літера H в даних адресах означає шістнадцяткову систему

У CSEG визначені два способи передачі керування:

1) пряма адресація за допомогою 16-розрядної прямої адреси ad16;
2) відносна адресація, що має два варіанти: за допомогою 8-розрядного зсуву **rel** (ціле двійкове зі знаком) щодо **PC**, або за допомогою вмісту акумулятора **@A** щодо **PC [(A)+(PC)]** або **DPTR [(A)+(DPTR)]**.

Для двох команд (**AJMP, ACALL**) передбачена сторінкова адресація в CSEG, за допомогою 11-розрядної адреси ad11. У цьому випадку 8

молодших розрядів адреси розташовуються в другому байті команди, а 3 старших - у трьох старших розрядах першого байта команди:



Номер сторінки задається п'ятьма старшими розрядами програмного лічильника **PC**.

Необхідно підкреслити так само, що в групі пересилки існують команди **MOVC A, @+DPTR** і **MOVC A, @A+PC**, які дозволяють читати вміст пам'яті програм. Як правило, ця можливість використовується для організації таблиць констант у CSEG.

7.2.2 Команди передачі керування

Групу команд передачі керування утворять команди безумовного переходу, умовного переходу, команди виклику підпрограми і команди повернення з підпрограми.

У більшості команд використовується пряма адресація. Характеристики команд приведені в таблиці 7.1.

Таблиця 7.1 - Група команд передачі керування

МНЕМОНІКА		КОД	Б	Ц	Операція
LJMP	ad16	00000010	3	2	(PC) := ab16
AJMP	ad11	aaa00001	2	2	(PC):= (PC)+2 (PC{0-10}):=ad11
SJMP	rel	10000000	2	2	(PC) := (PC)+2 (PC) :=(PC)+rel
JMP	@A+DPIR	01110011	1	2	(PC) :=(A)+(DPTR)
JZ	rel	01100000	2	2	(PC) :=(PC)+2; якщо (A)= =0 то (PC):=(PC)+rel
JNZ	rel	01110000	2	2	(PC) :=(PC)+2; якщо (A):=0 то (PC):=(PC)+rel
JC	rel	01000000	2	2	(PC) :=(PC+2; якщо (C)= =1 то (PC):=(PC)+rel
JNC	rel	01010000	2	2	(PC) :=(PC)+2; якщо C)= =0 то (PC):=(PC)+rel
JB	bit, rel	00100000	3	2	(PC) :=(PC)+3; якщо (b)= =1 то (PC):=(PC)+rel

JNB	bit, rel	00110000	3	2	(PC) :=(PC)+3; якщо (b)= =0 то (PC):=(PC)+rel
JBC	dit, rel	00010000	3	2	(PC) :=(PC)+3; якщо (b)= =1 то (PC):=(PC)+rel, (b):=0
DJNZ	Rn, rel	11011rrr	2	2	(PC):=(PC)+2;(Rh):=(Rn)-1; якщо (Rn):=0, то (PC):=(PC)+rel
DJNZ	ad, rel	11010101	3	2	(PC):=(PC)+3, (ad):=(ad)-1; якщо (ad):=0, то (PC):=(PC)+rel
CJNZ	A,ad, rel	10110101	3	2	(PC):=(PC)+3, якщо (A):=(ad); то (PC):=(PC)+rel; якщо (A)<(ad), то(C):=1, інакше (C):=0
CJNE	A, #d, rel	10110100	3	2	(PC):=(PC)+3, якщо (A):= #d, то(PC):=(PC)+rel; якщо (A)< #d, то (C):=1, інакше (C):=0;
CJNZ	Rn, #d, rel	10111rrr	3	2	(PC):=(PC)+3, якщо (Rn):= #d, то (PC):=(PC)+rel; якщо (Rn):= (ad), то (C):=1, інакше (C):=0;
CJNZ	@Ri, #d, rel	1011011i	3	2	(PC):=(PC)+3, якщо ((Ri)) = #d, то (PC):=(PC)+rel; якщо ((Ri))< #d, то (C):=1, інакше (C):=0;
LCALL	ad16	00010010	3	2	(PC):= (PC)+3,(SP):=(SP)+1 ((SP)):=(PCL), (SP):=(SP)+1 ((SP)):=(PCH), (PC):=ad16
ACALL	ad11	aaa10001	2	2	(PC) :=(PC)+2,(SP):=(SP)+1 ((SP)):=(PCL); (SP):=(SP)+1 ((SP)):=(PCH), (PC{0-10}):=ad11
RET		00100010	1	2	(PCH):=((SP)),(SP):=(SP)-1,
RET 1		00110010	1	2	(PCL):=((SP)),(SP):=(SP)-1, (PCH):=((SP)),(SP):=(SP)-1,
NOP		00000000	1	1	(PCL):=((SP)),(SP):=(SP)-1, (PC):=(PC)+1

7.3 Приклад циклічної програми

У пам'яті команд з адреси **ADR2** = 0D80 розташовані **N** = ØC H шістнадцяткових кодів. Наприклад: FF, 00, 11, 22, 33, 44, 55, 66, 77, 88, 99, AA.

Необхідно переписати їх у пам'ять даних, починаючи з адреси **ADR3** = 65 H. Програма повинна починатися з адреси **ADR1** = 0F00 H.

Текст програми:

```
    MOV   DPTR,#0D7F H
    MOV   R0, #70 H
    MOV   R1, #0C H
L1: MOVC  A, R1
    MOV   A, @A+DPTR
    MOV   @RO, A
    DJNZ  R0, L2
L2: DJNZ  R1, L1
    .END
```

Ще один варіант програми:

```
    MOV  PSW, #00B
    MOV  R1, #65
    MOV  DPTR, (0D80-65)
    MOV  A, R1
    MOVC A, @A + DPTR
    MOV  @R1, A
    INC  R1
    CJNZ R1, 71, F9
    NOP
```

7.4 Контрольні запитання та завдання

1. Розкажіть про організацію пам'яті програм CSEG.
2. Які способи адресації в пам'ять програм використовуються в командах Асемблера МК 51?
3. Які команди використовуються для сторінкової адресації?
4. Чому в командах звертання до пам'яті програм в Асемблері МК 51 джерелом даних може бути тільки пам'ять програм?
5. На основі яких умов організовані команди переходу?
6. Охарактеризуйте команди виклику підпрограм.
7. Нехай у пам'яті програм, починаючи з комірки **ADR2**, розташована таблиця кодів довжиною **N**. Запишіть в кодах МК 1816ВЕ51 програму, що виконує пересилку даного масиву в RAM, починаючи з адреси **ADR3**. Програма повинна починатися з комірки **ADR1** (Табл. 7.2).

Таблиця 7.2. - Таблиця варіантів завдання

Номер	ADR1	ADR2	N	ADR3
01	714	431	E	4F
02	62F	541	F	23
03	53E	621	D	45
04	44A	711	C	56
05	355	121	B	48
06	266	236	A	3D
07	177	345	6	4F
08	78A	454	7	3A
09	69D	568	F	4C
10	5AF	677	E	44
11	4BA	781	5	23
12	3C9	231	6	18
13	2D8	3A1	7	4C
14	1E4	4B2	8	6C
15	7E2	5C2	8	6E
16	6E6	6D2	9	3A
17	5A1	7E2	F	4A
18	4D3	1F2	A	5B
19	8C8	2E2	B	3B
20	26F	3D2	C	4B
21	4BE	782	D	39
22	3CA	232	E	47
23	2D3	3A2	F	65
24	1E7	4B2	8	33
25	7F3	5C2	8	3D
26	6E5	6D2	9	55
27	5AD	7E4	C	3E

Коди задавати довільно.

ЛАБОРАТОРНА РОБОТА №8

Програмування задач на Асемблері МК 51 з використанням арифметико-логічних операцій

Мета роботи - придбання навичок програмування арифметико-логічних операцій в кодах мікроконтролера.

8.1 Порядок виконання роботи

8.1.1 Ознайомитися з теоретичними відомостями, наведеними в даних методичних вказівках.

8.1.2 Для заданого викладачем варіанта задачі скласти програму в мнемокодах мови Асемблер МК 51.

8.1.3 Завантажити програму в емулятор М8751 та виконати в покроковому та неперервному режимах.

8.1.4 Зафіксувати та проаналізувати отримані результати.

8.1.5 Оформити звіт про виконання лабораторної роботи.

8.2 Група команд арифметико-логічних операцій

Команди даної групи дозволяють виконувати такі операції над 8-розрядними цілими двійковими числами: додавання, додавання з урахуванням перенесення, десяткову корекцію, інкремент і декремент, віднімання, множення, ділення, диз'юнкцію, кон'юнкцію, "виключне АБО", інверсію, очистку, зсув. Описання команд наведений в табл. 8.1, 8.2 та на рис 8.1. У табл. 8.3 наведені умови установки і скидання прапорців.

Ознака паритету P змінюється будь-якими командами, результат яких змінює акумулятор (включаючи команди передачі даних).

Таблиця 8.1 - Група команд арифметичних операцій

МНЕМОНІКА	КОД	Б	Ц	Операція
ADD A, Rn	00101rrr	1	1	(A) :=(A)+(Rn)
ADD A, ad	00100101	2	1	(A) :=(A)+(ad)
ADD A, @Ri	0010011i	1	1	(A) :=(A)+((Ri))
ADD A, #d	00100100	2	1	(A) :=(A)+#d
ADDC A, Rn	00111rrr	1	1	(A) :=(A)+(Rn)+(C)
ADDC A, ad	00110101	2	1	(A) :=(A)+(ad)(C)
ADDC A, @Ri	0011011i	1	1	(A) :=(A)+((Ri))+(C)
ADDC A, #d	00110100	2	1	(A) :=(A)+#d+(C)
DA A	11010100	1	1	Десяткова корекція
SUBB A, Rn	10011rrr	1	1	(A) :=(A)-(C)-(Rn)
SUBB A, ad	10010101	2	1	(A) :=(A)-(C)-(ad)
SUBB A, @Ri	1001011i	1	1	(A) :=(A)-(C)-((Ri))

SUBB	A, #d	10010100	2	1	(A) :=(A)-(C)-#d
INC	A	00000100	1	1	(A) :=(A)+1
INC	Rn	00001rrr	1	1	(Rn): =(Rn)+1
INC	ad	00000101	1	1	(ad): =(ad)+1
INC	@Ri	0000011i	2	1	((Ri)): =((Ri))+1
INC	DPTR	10100011	1	2	(DPTR): =(DPTR)+1
DEC	A	00010100	1	1	(A): =(A)-1
DEC	Rn	00011rrr	1	1	(Rn): =(Rn)-1
DEC	ad	00010101	1	1	(ad): =(ad)-1
DEC	@Ri	0001011i	2	1	((Ri)): =((Ri))-1
MUL	AB	10100100	1	1	(B) (A): =(A)*(У)
MUL	AB	10000100	1	1	(A).(У): =(A)/(У)

Таблиця 8.2 - Група команд логічних операцій

МНЕМОНІКА		КОД	Б	Ц	Операція
ANL	A, Rn	01011rr	1	1	(A) :=(A)Λ(Rn)
ANL	A, ad	01010101	2	1	(A) :=(A)Λ(ad)
ANL	A, @Ri	0101011i	1	1	(A) :=(A)Λ((Ri))
ANL	A, #d	01010100	2	1	(A): =(A)Λ#d
ANL	ad, A	01010010	2	1	(ad): =(ad)Λ(A)
ANL	ad, #d	01010011	3	2	(ad): =(ad)Λ#d
ORL	A, Rn	01001rrr	1	1	(A): =(A)V(Rn)
ORL	A, ad	01000101	2	1	(A): =(A)V(ad)
ORL	A, @Ri	0100011i	1	1	(A): =(A)V((Ri))
ORL	A, #d	01000100	2	1	(A): =(A)V#d
ORL	ad, A	01000010	2	1	(ad): =(ad)V(A)
ORL	ad, #d	01000011	3	2	(ad): =(ad)V#d
XRL	A, Rn	01101rrr	1	1	(A): =(A)((Rn)
XRL	A, ad	01100101	2	1	(A): =(A) ((ad)
XRL	A, @Ri	0110011i	1	1	(A): =(A) (((Ri))
XRL	A, #d	01100100	2	1	(A): =(A) (#d
XRL	ad, A	01100010	2	1	(ad): =(ad) ((A)
XRL	ad, #d	01100011	3	2	(ad): =(ad) (#d
CRL	A	11100100	1	1	(A):=0
CPL	A	11110100	1	1	(A) :=($\overline{A}$) інверсія A
RL	A	00100011	1	1	Ліворуч циклічно (рис.8.1 а)
RLC	A	00110011	1	1	Зсув ліворуч через С(рис.8.1б)
RR	A	00000011	1	1	Праворуч циклічно (рис.8.1в)
RRC	A	00010011	1	1	Праворуч через С (рис.8.1г)
SWAP	A	11000100	1	1	Обмін тетрадами (рис.8.1 д)

Таблиця 8.3 -Установка прапорців

Мнемоніка	CY	OV	AC	Мнемоніка	CY	OV	AC
ADD	+	+	+	CRL C	0	-	-
SUBB	+	+	+	CPL C	+	-	-
ADDC	+	+	+	ANL C, bit	+	-	-
MUL	0	+	-	ANL C,/bit	+	-	-
DIV	0	+	-	ORL C, bit	+	-	-
DA	+	-	-	ORL C,/bit	+	-	-
RRC	+	-	-	MOV C, bit	+	-	-
RLC	+	-	-	CJNE	+	-	-
SET C	1	-	-				

(+) змінюється
(-) не змінюється

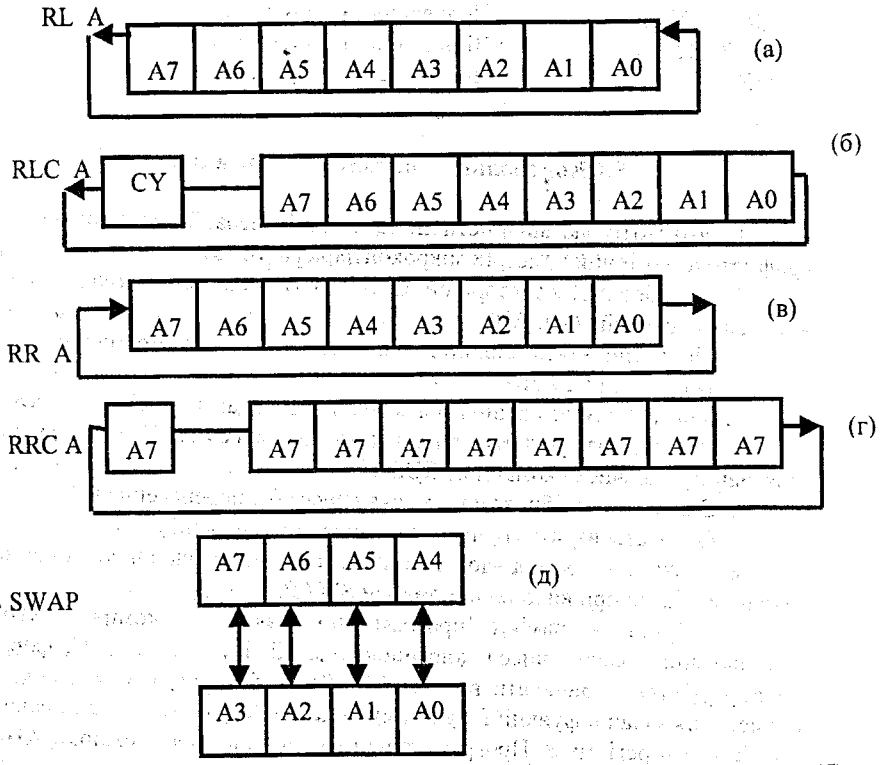


Рисунок 8.1 – Команди циклічного зсуву вліво (а) і вліво через перенос (б) та команди циклічного зсуву вправо (в) та вправо через перенос (г); команда обміну між старшою і молодшою тетрадами (д)

8.3 Приклад виконання роботи

Нехай в пам'яті програм, починаючи з комірки **ADR2** = 0F20, розташована таблиця 5-ти кодів десяткових чисел (21, 32, 43, 54, 65).

Необхідно скласти і налагодити програму, яка обчислює суму за модулем 2 цих кодів, результат помістити в комірку 7F RAM. Програма повинна починатися з комірки **ADR1** = 0200.

Текст програми

```
    MOV     PSW, #00B   ; Задаємо нульовий банк
    CRL     A           ; Очищуємо акумулятор
    MOV     7FH, A      ; Очищуємо результат
    MOV     R2,#05      ; Задаємо лічильник
    MOV  DPTR, #0F1     ; Задаємо початкову адресу масиву
L2: MOV     A, R2       ; Індекс елемента в А
    MOVC    A,@A+DPTR ; Елемент масиву в А
    XRL   7FH, A        ; Додавання за модулем 2
    DJNZ  R2, L1        ; Віднімання 1 і перевірка на 0
    NOP                 ; Вихід
```

8.4 Контрольні запитання та завдання

1. Розкажіть, які арифметичні та логічні операції може виконувати арифметико-логічний пристрій мікроконтролера МК 51.

2. Де зберігається старший байт добутку, а де молодший після виконання операції MUL AB?

3. Де зберігається старший байт добутку, а де молодший після виконання операції DIV AB?

4. В чому полягає різниця між командами додавання ADD і ADDC ?

5. Як можна усунути вплив біта перенесення С на результат віднімання двох чисел командою SUBB?

6. Для чого потрібні команди зсуву через біт перенесення С?

7. Охарактеризуйте групу команд логічних операцій.

8. Розкажіть про порядок обміну бітами між молодшою і старшою тетрадами байта при виконанні команди SWAP.

9. Нехай в пам'яті програм, починаючи з комірки **ADR2**, розташований масив чисел довжиною N (Xi, i = 1,2,……, N, формат числа – байт). Записати в кодах МК1816ВЕ51 програму, яка виконує обчислення заданої функції F над цими кодами. Результат обчислення розмістити в регістр **B**. Програма повинна починатися з комірки **ADR1**. Варіанти завдань наведені в табл. 8.4.

Таблиця 8.4 - Таблиця варіантів завдань

Номер	ADR1	ADR2	N	Функція F
01	741	431	E	Сума (Xi)/N
02	62F	541	F	Max(Xi)/N
03	53E	621	D	Min(Xi)/N
04	44A	711	3	Сума (Xi)\/X1
05	355	121	B	Max(Xi)\/X1
06	266	236	A	Min (Xi)\/X1
07	177	345	6	Сума (Xi)/\X1
08	78A	454	7	Max(Xi)/\X1
09	69D	568	F	Min (Xi)/\X1
10	5AF	677	E	Сума (Xi)/\X2
11	4BA	781	5	Max(Xi)\/X2
12	3C9	231	6	Min (Xi)/\X2
13	2D8	3A1	7	Сума (Xi)/\X3
14	1E4	4B2	8	Max(Xi)/\X3
15	7F5	5C2	8	Min (Xi)/\X3
16	6E6	6D2	9	Сума (Xi)/\X4
17	5A1	7E2	F	Max(Xi)\/X4
18	4D3	1F2	A	Min (Xi)\/X4
19	3C8	2E2	B	Сума (Xi)(X1
20	26F	3D2	3	Max(Xi)(X1
21	4BE	782	D	Min (Xi)(X1
22	3CA	232	E	Сума (Xi)/\Xn
23	2D3	3A2	F	Max(Xi)/\Xn
24	1E7	4B2	8	Min (Xi)/\Xn
25	7F3	5C2	8	Сума (Xi)\/X1n
26	6E5	6D2	9	Max(Xi)\/Xn
27	5AD	7E4	3	Min (Xi)\/Xn

Коди даних задати довільно. Виконати контрольний розрахунок.

ЛАБОРАТОРНА РОБОТА №9

Програмування бітових операцій на мові Асемблер МК 51

Мета роботи - вивчення організації бітового простору пам'яті мікроконтролера КМ1816ВЕ51, програмних засобів обробки біт і отримання навичок програмування бітових операцій у кодах мікроконтролера.

9.1 Порядок виконання роботи

9.1.1 Ознайомитися з теоретичними відомостями, наведеними в даних методичних вказівках.

9.1.2 Для заданого варіанта задачі (табл. 9.3) скласти програму в мнемокодах мови Асемблер МК 51.

9.1.3 Завантажити програму в емулятор М8751 та виконати в покроковому режимі.

9.1.4 Зафіксувати та проаналізувати отримані результати.

9.1.5 Оформити звіт про виконання лабораторної роботи.

9.2 Теоретичні відомості

9.2.1 Організація простору пам'яті BSEG

Дана група команд оперує з однобітовими операндами. В якості операндів можуть виступати окремі біти деяких регістрів спеціальних функцій, біти портів і біти 16 осередків внутрішньої пам'яті даних. Всі біти, що адресуються, утворюють однорозрядний лінійний впорядкований простір BSEG ємністю 256 біт.

У просторі BSEG використовується тільки пряма адресація, пряма восьмирозрядна адреса в просторі BSEG позначається **bit**.

Адресація в просторі BSEG ілюструється табл. 9.1.

Таблиця 9.1 - Адресація в просторі бітової пам'яті BSEG

Адреса в DSEG	Розряд комірки внутрішньої пам'яті даних							
	7	6	5	4	3	2	1	0
20	07	06	05	04	03	02	01	00
21	0F	0E	0D	0C	0B	0A	09	08
22	17	16	15	14	13	12	11	10
23	1F	1E	1D	1C	1B	1A	19	18
24	27	26	25	24	23	22	21	20
25	2F	2E	2D	2C	2B	2A	29	28

26	37	36	35	34	33	32	31	30
27	3F	3E	3D	3C	3B	3A	39	38
28	47	46	45	44	43	42	41	40
29	4F	4E	4D	4C	4B	4A	49	48
2A	57	56	55	54	53	52	51	50
2B	5F	5E	5D	5C	5B	5A	58	59
2C	67	66	65	64	63	62	61	60
2DF	6F	6E	6D	6C	6B	6A	68	69
2E	77	76	75	74	73	72	71	70
2F	7F	7e	7D	7C	7B	7A	79	78

Адреса в DSEG	Розряд регістра спеціального призначення							
	7	6	5	4	3	2	1	0
80	87	86	85	84	83	82	81	00
88	8F	8E	8D	8C	8B	8A	89	88
90	97	96	95	94	93	92	91	90
98	9F	9E	9D	9C	9B	9A	99	98
A0	A7	A6	A5	A4	A3	A2	A1	A0
A8	--	--	--	AC	AB	AA	9	8
B0	B7	B6	B5	B4	B3	B2	B1	B0
B8	--	--	--	BC	BB	BA	B8	B9
D0	D7	D6	D5	D4	D3	D2	D1	D0

E0	E7	E6	E5	E4	E3	E2	E1	E0
F0	F7	F6	F5	F4	F3	F2	F1	F0

9.2.2 Команди роботи з бітовими даними

Група команд операцій з бітами (табл. 9.2.) включає 6 операцій: три одномісних операції – установки (**CLR**), скидання (**SETB**), і інверсії (**CPL**), дві двомісних операції – кон'юнкції і диз'юнкції, і операцію пересилання. В якості акумулятора в бітових операціях використовується тригер (прапорець) переносу **C**. Характеристики бітових команд наведені в табл. 9.2.

Таблиця 9.2. - Група команд операцій з бітами

Мнемоніка	КІП	Б	Ц	Операція
CLR C	11000011	1	1	(C) := 0
CLR bit	11000010	2	1	(b) := 0
SETB C	11010011	1	1	(C) := 1
SETB bit	11010010	2	1	(b) := 1
CPL C	10110011	1	1	(C) :=/(C)
CPL bit	10110010	2	1	(b) :=/(b)
ANL C,bit	10000010	2	2	(C) :=(C)^(b)
ANL C,/bit	10110000	2	2	(C) :=(•C)^/(b)
ORL C,bit	01110010	2	2	(C) :=(C)V(b)
ORL C,/bit	10100000	2	2	(C) :=(C)V/(b)
MOV C,bit	10100010	2	1	(C) :=(b)
MOV bit,C	10010010	2	2	(B) :=(C)

Примітка:
символ "/" перед операндом означає його інверсію

9.3 Приклад виконання роботи

Нехай у DSEG, в комірці **ADR**=2D розташований код CODE = 31. Написати в кодах КМ816ЕВ51 програму, яка виконує обчислення заданої бульової функції F= X4/\ /X3\/X0 над цими кодами. Результат обчислень повинен бути записаний за адресою **ADR3**=7F простору BSEG. Програма повинна починатися з комірки **ADR1**= 0F00.

Текст програми:

```
MOV  2 DH,31
MOV  C, 2D.4H
ANL  C, /2D.3H
ORL   C, 2D.0H
MOV  7FH, C
.END
```

9.4 Контрольні запитання та завдання

1. В якій області пам'яті розташований простір BSEG?
2. Як здійснюється бітова адресація в пам'яті бітового процесора?
3. Що є акумулятором в випадку виконання бітових операцій?
4. Які операції над бітами може виконувати мікроконтролер МК 51?
5. Нехай в області пам'яті DSEG в комірці **ADR2** розташований код CODE. Записати на мові Асемблер МК 51 програму, що виконує обчислення заданої бульової функції F над цими кодами. Результат обчислення повинен бути записаний за адресою **ADR3** простору BSEG. Програма повинна починатися з комірки **ADR1**.

Таблиця 9.3. - Варіанти завдань

НОМЕР	ADR1	ADR2	CODE	ADR3
01	333	43	FE	7A
02	A45	54	3F	4B
03	563	62	5D	6C
04	234	71	7C	7D
05	D41	12	8B	1E
06	872	23	7A	2F
07	45F	34	1E	4F
08	32E	45	F2	5A
09	695	56	E3	6B
10	5A2	67	FA	3C
11	4B3	78	A5	4D
12	3C4	23	56	5E
13	2DF	3A	F7	6F
14	1E3	4B	A8	7A
15	7FE	5C	48	5B
16	6E2	6D	29	6C
17	5A2	7E	F3	3D
18	4D4	1F	E2	5E
19	3C3	2E	D3	6F
20	261	3D	A4	5A

21	38F	5D	B5	3D
22	2CC	53	3D	2E
23	3D3	6A	4F	3e
24	7E3	7B	8F	48
25	2F4	5E	6D	5B
26	4E4	45	D9	6C

Вид функції F для обчислення визначається в такий спосіб:
Нехай YZ дві останні цифри номера залікової книжки студента.
Код CODE має такий формат (порозрядно):

X7	X6	X5	X4	X3	X2	X1	X0

F=(X7)OP1(X6)OP2(X5)OP3(X4)OP4(X3)OP5(X2)OP6(X1)OP7(X0),
де OP – бульова операція, знаходиться з табл. 9.4, а кожна змінна X може входити у вираз прямо чи інверсно (визначається з табл. 9.5).

Таблиця 9.4. –Бульові операції

Z	0	1	2	3	4	5	6	7	9	0
OP1	∧	∧	∧	∧	∧	∨	∨	∨	∨	∨
OP2	⊕	⊕	⊕	∧	∧	∧	∨	∨	∨	∨
OP3	∨	∨	∨	⊕	⊕	⊕	⊕	∨	⊕	∧
OP4	⊕	⊕	∨	∧	∨	∧	∧	∧	∨	∨
OP5	∧	∧	∨	∨	∨	⊕	⊕	⊕	⊕	⊕
OP6	⊕	∨	∧	∨	∧	∧	⊕	⊕	∧	∧
OP7	∨	⊕	⊕	⊕	⊕	⊕	∨	∧	⊕	∨

Λ - операція І (кон'юнкція),
V - операція АБО (диз'юнкція),
⊕ - операція " виключне АБО" (XOR)

Таблиця 9.5 - Значення змінних

Y	0	1	2	3	4	5	6	7	8	9
X0	П	І	П	І	П	І	П	І	П	І
X1	П	П	П	П	І	І	І	І	І	І
X2	П	П	І	І	І	П	П	І	І	П
X3	І	І	І	П	П	П	І	І	І	І
X4	І	І	І	П	І	П	П	І	П	П
X5	П	І	І	П	І	П	П	П	І	І
X6	І	П	П	П	І	І	П	П	П	П

П – змінна входить у функцію прямо (без інверсії)
І – змінна у функцію входить інверсно
X7 – входить у вираз для F завжди без інверсії.

Література

1. М.А.Бердяковский, И.С.Кручинкин, В.А.Подолян. Микропроцессоры. – М.: Радио и связь, 1981. – 306 с.
2. Е. П. Балашов, Д. В. Пузанков. Микропроцесори и микропроцессорные системы.- М.: Радио и связь,1981. – 267 с.
3. Микропроцессоры. В 3-х книгах. Кн.1. Архитектура и проектирование микроЭВМ. Организация вычислительных процессов. /Под ред. Л.Н.Преснухина –М.:Высшая школа, 1986. – 456 с.
4. Р. Токхайм. Микропроцессоры. Курс и упражнения. – М.: Энергоатомиздат, 1988. – 336 с.
5. Д. Гивонне, Р. Россер. Микропроцессоры и микрокомпьютеры. – М.: Мир, 1983. – 464 с.
6. Корнеев В.В., Киселев А.В. Современные микропроцессоры.–М.: Нолидж, 2001.– 240 с.
7. Сташин В.В. Проектирование цифровых устройств на однокристальных МК. – М.: Мир, 1990. – 414 с.
8. Бродин В.Б., Калинин А.В. Системы на микроконтроллерах и БИС программируемой логики.- М.: ЭКОМ, 2002.- 400 с.
9. Юров В., Хорошенко С. Ассемблер. Учебный курс.– С. Петербург: "Питер", 2000.– 672 с.

Навчальне видання

Микола Максимович Биков
Геннадій Леонідович Лисенко
Світлана Михайлівна Москвіна
Володимир Андрійович Лужецький
Тетяна Борисівна Мартинюк

Основи мікропроцесорної техніки

Лабораторний практикум

Оригінал-макет підготовлено Биковим М.М.

Редактор С.А. Малішевська

Навчально-методичний відділ ВНТУ
Свідоцтво Держкоміноформу України
серія ДК № 746 від 25.12.2001
21021, м. Вінниця, Хмельницьке шосе, 95, ВНТУ

Підписано до друку 4.12.2003р.
Формат 29,7х42 1/4
Друк різографічний
Тираж 80 прим.
Зам. №

Гарнітура Times New Roman
Папір офсетний
Ум. друк. арк. 4,62

Віддруковано в комп'ютерному інформаційно-видавничому центрі
Вінницького національного технічного університету
Свідоцтво Держкоміноформу України
Серія ДК № 746 від 25.12.2001
21021 м. Вінниця, Хмельницьке шосе, 95, ВНТУ