МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет «Запорізька політехніка»

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторних робіт
з дисципліни

«Архітектура комп'ютера та низькорівневе програмування»

для студентів
спеціальності 122 «Комп'ютерні науки»

2020

Методичні вказівки до лабораторних робіт з дисципліни «Архітектура комп'ютера та низькорівневе програмування» для студентів спеціальності 122 «Комп'ютерні науки» / Укл.: С.К. Корнієнко. – Запоріжжя: НУ «Запорізька політехніка», 2020. – 46 с.

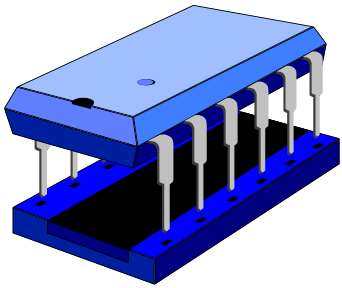
Укладач: С.К. Корнієнко, доцент, к.т.н.,

Рецензент: О.О. Степаненко, доцент, к.т.н.

Відповідальний за випуск: С.О. Субботін, професор, д.т.н.

Затверджено
на засіданні кафедри ПЗ

Протокол № 1 від 18.08.2020



ЗМІСТ

1 Лабораторна робота «Вивчення архітектури 8-розрядного мікропроцесора»

1.1 Мета роботи

Метою роботи є ознайомлення з архітектурою 8-розрядного мікропроцесора (МП) на базі МП I8080, який отримав широке розповсюдження, а його архітектура була визнана класичною, а також отримання навичок практичної роботи з програмою-емулятором, вивчення принципів складання та вводу найпростіших програм до оперативної пам'яті МПС і способу їхньої корекції.

1.2 Завдання до лабораторної роботи

1.2.1 Вивчити архітектуру мікропроцесора i8080 і мікропроцесорної системи на його основі.

1.2.2 Вивчити структуру емулятора мікропроцесорної системи (ЕМПС) та призначення його основних команд.

1.2.3 Ввести програму та перевірити її виконання.

1.3 Методичні вказівки до виконання лабораторної роботи

1.3.1 Вказівки до виконання п. 1.2.1

Теоретичний матеріал за п. 1.2.1 завдання стисло викладений у Додатку А даних методичних вказівок.

1.3.2 Вказівки до виконання п. 1.2.2 завдання

Відомості про призначення, основні функції та склад команд емулятора МПС викладено у Додатку Б.

1.3.3 Вказівки до виконання п. 1.2.3 завдання.

Для виконання п. 1.2.3 завдання необхідно отримати у викладача текст програми, ввести її до емулятора та запустити на виконання.

1.4 Зміст звіту

Звіт повинен містити:
- назва та мета роботи;
- програмістська модель МП;
- текст програми.

1.5 Контрольні питання

1. Що таке регістр?
2. Які регістри МП є програмно доступними?
3. Призначення акумулятора.
4. Яка послідовність налагодження програми та її запуску?
5. Основні функції програми-емулятора МПС.

2 Лабораторна робота «Вивчення принципів реалізації лінійних програм»

2.1 Мета роботи

Метою роботи є вивчення групи команд організації пересилань, арифметичних операцій, вводу-виводу даних, роботи з ОЗП, організації роботи зі стеком та придбання практичних навичок у складанні та налагодженні простих лінійних програм.

2.2 Завдання на лабораторну роботу

2.2.1 Вивчити призначення та склад внутрішніх регістрів мікропроцесора та формат команд.

2.2.2 Вивчити позначення і зміст команд, що входять до групи команд пересилань даних, арифметичних операцій, вводу-виводу даних, звертання до ОЗП та роботи зі стеком.

2.2.3 Скласти та реалізувати програму складання (віднімання) двох чисел з використанням регістрів МП.

2.2.4 Скласти та реалізувати програму обробки даних з використанням ОЗП.

2.2.5 Скласти та реалізувати програму обміну даними між регістрами з використанням стека.

Примітка: при складанні програми використовувати різні методи адресації з метою порівнювального аналізу та вибору оптимального варіанту.

2.3 Методичні вказівки до виконання лабораторної роботи

Для виконання завдань лабораторної роботи необхідно попередньо ознайомитися зі змістом лабораторної роботи №1.

2.3.1 Вказівки до виконання п.2.2.1 завдання

Для виконання п. 2.2.1 лабораторного завдання необхідно ознайомитися з Додатком А методичних указівок. Формати команд мікропроцесора описані в Додатку В.

2.3.2 Вказівки до виконання п. 2.2.2 завдання

Виконання п. 2.2.2 вимагає вивчення вказаних у завданні груп команд згідно табл. В.1 Додатку В.

2.3.3 Вказівки до виконання п. 2.2.3 завдання

Згідно отриманого у викладача завдання скласти, ввести та налагодити програму, користуючись вказівками попередньої лабораторної роботи, а також додатками А-В.

2.3.4 Вказівки до виконання п. 2.2.4 завдання

Виконання п. 2.2.4 завдання здійснюється в послідовності, аналогічній п. 2.2.3.

2.3.5 Вказівки до виконання п. 2.2.5 завдання

Реалізацію обміну між регістрами з використанням стека необхідно здійснювати в такій послідовності.

– сформувати стек, для цього до парного регістру SP завантажити константу, що дорівнює адресі чарунки ОЗП, яка є вершиною стека.

– виконати пересилку змісту РЗП до стека у необхідному порядку, а потім завантажити РЗП інформацією зі стека. В обміні між РЗП та стеком приймають участь пари регістрів. При цьому формування адрес звернення до стека відбувається автоматично, реалізуючи принцип стекової організації «останнім прийшов – першим вийшов».

– виконати перевірку змісту регістрів до та після обміну інформацією.

2.4 Зміст звіту

Звіт повинен містити:

- мету роботи;
- тексти програм по пп.2.3, 2.4, 2.5;

- схеми алгоритмів програм;
- аналіз отриманих результатів та висновки.

Примітка: при захисті лабораторної роботи обов'язково треба мати при собі роздруковану копію системи команд МП.

2.5 Контрольні питання

1. Назвіть склад внутрішніх регістрів ЦПЕ.
2. Що таке формат команди?
3. Яке призначення окремих груп команд?
4. Який зміст основних команд?
5. Як виконується ввід (вивід) інформації у МПС?
6. Як здійснюється адресація чарунок ОЗП?
7. Що таке стек, його організація та використання?

3 Лабораторна робота «Вивчення принципів реалізації розгалужених і циклічних програм»

3.1 Мета роботи

Метою роботи є вивчення групи команд, які використовуються при організації переходів, формуванні лічильників, операціях зсувів та надбання практичних навичок у використанні регістра ознак при складанні й налагодженні розгалужених та циклічних програм.

3.2 Завдання на лабораторну роботу

3.2.1 Вивчити призначення та склад регістра ознак.

3.2.2 Вивчити позначення і зміст команд, що входять до групи зсувів, переходів та інкремент-декремент.

3.2.3 Скласти та реалізувати програму обчислення складної функції.

3.2.4 Скласти та реалізувати програму формування часових інтервалів.

3.3 Методичні вказівки щодо виконання лабораторної роботи

Виконання даної лабораторної роботи базується на знаннях, отриманих під час виконання попередніх робот.

3.3.1 Вказівки до виконання п 3.2.1 завдання

Для виконання п.3.2.1 завдання необхідно вивчити матеріал Додатку Д.

3.3.2 Вказівки до виконання п. 3.2.2 завдання

Для виконання п. 3.2.2 необхідно вивчити вказані у завданні групи команд за таблицею В.1 Додатку В. При цьому потрібно зосередити особливу увагу на графу 3 таблиці В.1, в якій наведено час виконання команд у машинних тактах. Час одного машинного такту **t = 0,5 мкс** (відповідає частоті тактового генератора 2 МГц).

3.3.3 Вказівки до виконання п. 3.2.3 завдання

За отриманим у викладача завданням скласти схему реалізації заданої функції. Текст програми необхідно оформляти у вигляді таблиці, в якій повинні бути передбачені графи для адреси пам'яті, мітки, мнемонічних позначень команд та коментарів.

Результати обчислень у точках, вказаних викладачем, а також кінцевий результат необхідно фіксувати в протоколі виконання лабораторної роботи.

3.3.4 Вказівки до виконання п. 3.2.4 завдання

Для виконання п. 3.2.4 завдання необхідно отримати у викладача завдання, у відповідності з яким визначити варіант реалізації алгоритму (Додаток Ж). Розробити необхідні розрахунки, враховуючи час виконання команд (див. графу 3 табл. В.1) і варіанти реалізації. Скласти детальну схему та текст програми у відповідності зі вказівками п. 3.3.2.

3.4 Зміст звіту

Звіт повинен містити такі матеріали.

– мету роботи.

– завдання, схему алгоритму та текст програми, результати розрахунків за п. 3.2.3 завдання.

– завдання, необхідні розрахунки, схему, текст програми та результати експерименту за п. 3.2.4.

– аналіз отриманих результатів та висновки.

3.5 Контрольні питання

1. Що таке регістр ознак?
2. Як реалізуються команди умовних переходів?
3. Організація лічильника циклів?
4. Яке призначення розгалужених та циклічних програм?

4 Лабораторна робота «Обробка бітів»

4.1 Мета роботи

Метою роботи є вивчення групи команд логічних операцій.

4.2 Завдання на лабораторну роботу

4.2.1 Вивчити позначення та склад команд, що входять до групи логічних операцій.

4.2.2 Отримати завдання у викладача та створити відповідні схеми алгоритмів і робочі програми.

4.3 Методичні вказівки до виконання лабораторної роботи

Для виконання завдання по даній роботі необхідно згадати основні відомості про логічні функції та виконати лабораторну роботу № 3.

4.3.1 Вказівки до виконання п. 4.2.1 завдання

Для виконання завдання за п. 4.2.1 необхідно вивчити відповідний розділ таблиці В.1 Додатку В.

4.3.2 Вказівки до виконання п. 4.2.2 завдання

При виконанні даного пункту необхідно керуватися матеріалом, викладеним у Додатку К.

4.4 Зміст звіту

Звіт повинен містити такі матеріали:

- мету роботи;
- завдання на лабораторну роботу;
- схему алгоритму;
- текст програми;

4.5 Контрольні питання

1. Який зміст команд, що входять до групи логічних операцій?
2. Таблиці істинності логічних операцій.
3. Для чого та яким чином проводиться маскування?
4. Як здійснюється інвертування потрібних розрядів у байті інформації?

5 Лабораторна робота «Вивчення принципів реалізації програм управління зовнішніми пристроями»

5.1 Мета роботи

Метою роботи є вивчення прийому блочної реалізації програм із використанням команд ***CALL*** та ***RET*** на прикладі програми керування портами індикації.

5.2 Завдання на лабораторну роботу

5.2.1 Вивчити позначення та зміст команд ***CALL*** та ***RET*** за різноманітними умовами, що входять до групи команд переходів.

5.2.2 Скласти схему алгоритму, який дозволяє на вихідних портах отримати «рухомі вогні».

5.2.3 Реалізувати програму «рухомі вогні» з керуванням із портів **00** і **01.**

5.2.4 Внести необхідні зміни у програму для сумісного використання індикаторів **00** і **01**.

5.3 Методичні вказівки до виконання лабораторної роботи

5.3.1 Вказівки до виконання п.2.1 завдання

Для виконання п. 5.2.1 лабораторного завдання необхідно вивчити групу команд переходів за таблицею В.1 Додатку В.

Потрібно звернути увагу, що трибайтна команда ***CALL*** є командою звернення до підпрограми, яка починається з деякої адреси. За командою ***CALL*** зміст лічильника команд центрального процесора, тобто адреса наступної команди, заноситься до стекової області ОЗП, а зміст покажчика стека зменшується на 2. Після операцій зі стеком до лічильника команд заноситься початкова адреса підпрограми, вказана другим та третім байтами команди ***CALL***, після чого починається її виконання.

У кінці підпрограми обов’язково повинна стояти команда ***RET***. Команда ***RET*** відновлює той стан процесора й стека, який був безпосередньо перед виконанням команди ***CALL***. Зміст покажчика

стека збільшується на 2, а до лічильника центрального процесора заноситься адреса, що була передана для зберігання до стекової пам'яті за командою ***CALL***.

5.3.2 Вказівки до виконання п. 5.2.2 завдання

Працююча програма повинна реагувати на такі керуючі впливи, що задаються шляхом установлення відповідних розрядів **Bi** порту **01** в одиницю (рисунок 5.1):

«***ПУСК-СТОП»***, тобто при $\mathbf{B_7}$=1 – переміщення інформації на порту індикації (вогні «рухаються»), а при $\mathbf{B_7}$=0 – немає переміщення;

«***ВЛІВО-ВПРАВО»***, тобто при $\mathbf{B_6}$=1 – переміщення інформації вліво, а при $\mathbf{B_6}$=0 – вправо;

«***ШВИДКІСТЬ»*** – розряди $\mathbf{B_0}$-$\mathbf{B_5}$ повинні дозволяти в широких межах змінювати швидкість переміщення інформації (час переносу інформації на один розряд від 0,05с до 3-4с);

«***ВВІД ІНФОРМАЦІЇ»*** – в положенні «***СТОП»*** інформація, яка вводиться повинна задаватися з порту **00**.

5.3.3 Вказівки до виконання п. 5.2.3 завдання

Оскільки режим роботи програми визначають старші розряди порту **B** ($\mathbf{B_7}$ і $\mathbf{B_6}$), їхній стан доцільно аналізувати за прапорцем перенесення, здійснив попередньо зсув байта, що аналізується, вліво.

Константа, що визначає час затримки між зсувами ("**ШВИДКІСТЬ**"), легко отримується в акумуляторі з використанням відповідної маски (див. лабораторну роботу №4).

«Рухомі вогні» отримуються регулярним оновленням інформації на порту індикації. Час затримки визначається розрядами $\mathbf{B_0}$-$\mathbf{B_5}$ порту **B** (див. вище). Оновлення інформації полягає в її зсуві в акумуляторі на один розряд уліво або вправо. Після кожного зсуву інформацію необхідно запам'ятовувати в одному із РЗП і для наступного зсуву знов використовувати акумулятор. Для спрощення процесу налагодження програми керування індикатором доцільно виділити дві підпрограми:

- перша – підпрограма мінімальної затримки t_{min}=10мкс;
- друга – підпрограма змінної затримки з кроком t_{min};

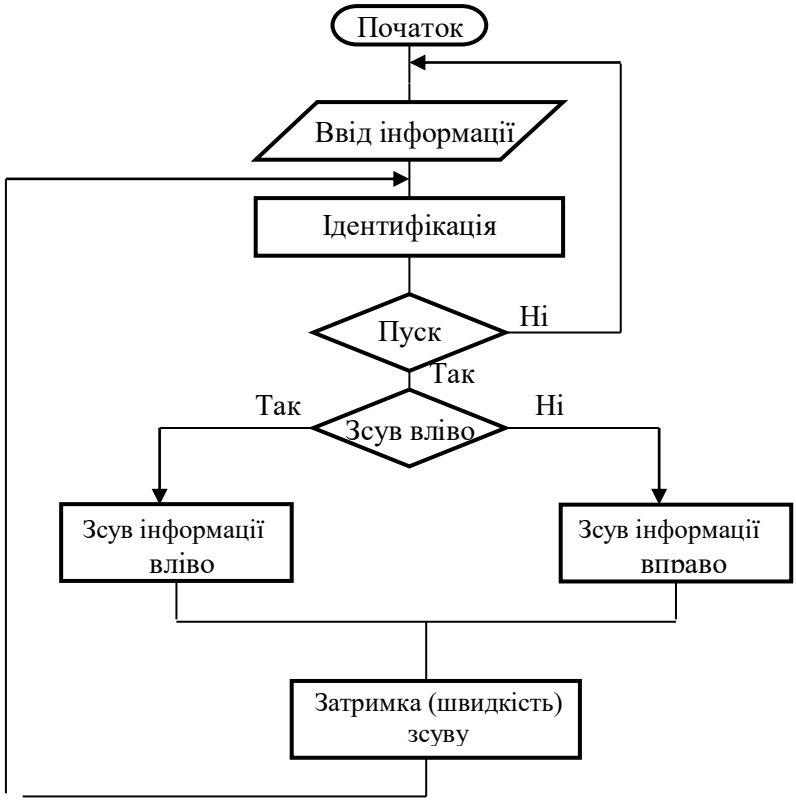


Рисунок 5.1 - Схема алгоритму керування портом виводу

5.3.4 Вказівки до виконання п.2.4 завдання

Зміни в програмі насамперед стосуються вводу інформації та її виводу на індикатори. При цьому треба пам'ятати, що операції ***IN*** та ***OUT*** використовують однобайтовий регістр-акумулятор. Зсув інформації в двобайтовому регістрі вліво доцільно здійснювати за допомогою команди ***DAD***, а вправо – комбінуючи команди зсуву через прапорець перенесення та пересилання між регістрами.

5.4 Зміст звіту

Звіт повинен містити наступні матеріал:

- мету роботи;
- позначення та зміст команд звернення до підпрограми і повернення з неї, команд циклічного зсуву;
- схему алгоритму програми керування;
- текст програми керування індикатором;
- текст змін програми для одночасного використання двох портів R та L ;
- аналіз отриманих результатів та висновки.

5.5 Контрольні питання

1. Який зміст команд ***CALL*** і ***RET***?
2. Який зміст команд циклічного зсуву?
3. Що таке адреса повернення? Де вона зберігається і як використовується?
4. Як може виконуватися аналіз окремих бітів слова?
5. У чому полягає різниця та схожість прийомів зсуву однобайтової та двобайтової інформації?
6. Як реалізується зміна величини затримки операцією з порту вводу?

6 Лабораторна робота «Розробка програми управління технологічними процесами за допомогою графічного інтерпретатора портів виводу»

6.1 Мета роботи

Метою даної роботи є одержання навичок складання програм керування технологічними процесами за допомогою графічного інтерпретатора портів виводу (ГІПВ).

6.2 Завдання до лабораторної роботи

6.2.1 Вивчити основні функції ГІПВ (таблиця Л.1).
6.2.2 Вивчити константи для завдання кольорів (таблиця Л.2).
6.2.3 Одержати завдання у викладача.
6.2.4 Скласти та запустити програму на виконання.

6.3 Методичні вказівки до виконання лабораторної роботи

6.3.1 Вказівки до виконання п. 6.2.1 завдання

Графічний інтерпретатор портів виводу описаний у Додатку Л. Основні функції ГІПВ наведені у таблиці Л.1.

6.3.2 Вказівки до виконання п. 6.2.2 завдання

Робота з кольором (константи, що завдають колір) наведені у таблиці Л.2.

6.3.3 Вказівки до виконання п.п. 6.2.3 та 6.2.4 завдання

Одержавши від викладача завдання на лабораторну роботу, треба спочатку структурувати майбутнє графічне зображення на окремі графічні об’єкти, а вже потім приступати до реалізації програми, керуючись Додатком Л.

Робота з графічним інтерпретатором здійснюється через порт 05.

Значення, що надходять на цей порт, сприймаються (інтерпретуються) ГІПВ як деякі керуючі команди.

Кожна така команда займає 2 байти, тому може бути передана на виконання тільки за 2 кроки: спочатку на порт 05h посилається перший байт команди, а потім другий байт. Перший байт містить код команди, а другий байт - її параметри.

6.4 Зміст звіту

- ціль роботи.
- текст програми.
- результат роботи програми.
- висновки.

6.5 Контрольні питання

1. Для чого призначений ГІПВ?
2. Чому інформація на ГІПВ передається в 2 прийоми?
3. Які основні функції виконує ГІПВ?

Додаток А Програмістська модель мікропроцесорної системи

Для кращого розуміння особливостей системи команд мікропроцесора використовується програмістська модель мікропроцесорної системи (МПС), що відповідає її спрощеній структурі (рис. А.1). Модель містить тільки вузли, що є програмно доступними. До них відносяться блок регістрів мікропроцесора, основна пам'ять розміром 64К, 256 портів вводу та 256 портів виводу.

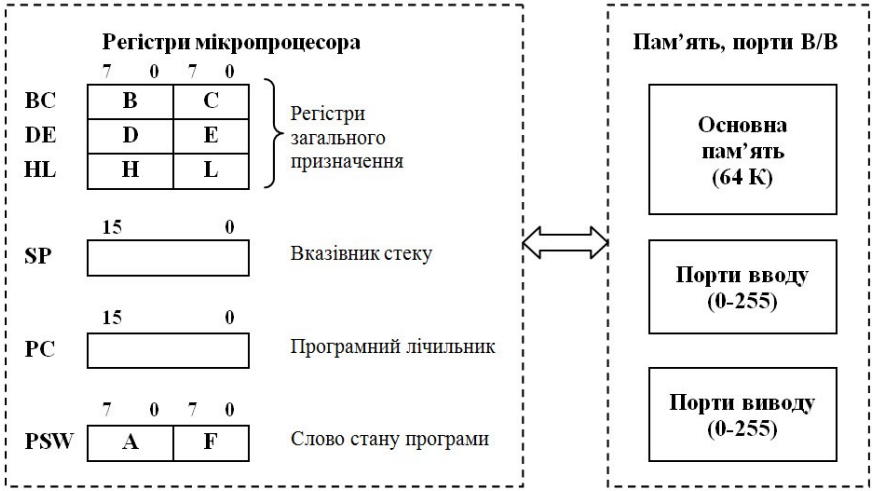


Рисунок А.1 – Програмістська модель МПС

Додаток Б Емулятор мікропроцесорної системи

Емулятор мікропроцесорної системи (ЕМПС) – це програмний комплекс, призначений для моделювання роботи мікропроцесорних систем (МПС) на ПЕОМ старших поколінь та навчання студентів основам програмування на мові Асемблера мікропроцесора I8080. МПС, яка моделюється, складається з центрального процесорного елементу, ОЗП та портів вводу-виводу.

Крім того, програма містить графічний інтерпретатор портів виводу (ГІПВ), що імітує керування зовнішнім приладом за допомогою порту виводу 05. Програма містить детальну довідку як по роботі з емулятором, так і по командам мікропроцесора I8080, яка при запущеному застосунку «Емулятор CPU I8080» викликається натисканням кнопки F1.

Функції ЕМПС:

– створення програм у текстовому редакторі та збереження їх до файлу з виконанням усіх функцій будь-якого текстового редактора: друк файлу, робота з блоками, пошук та заміна, тощо;

– компіляція та запуск програми користувача;

– покрокове виконання програми та встановлення точок зупинки у вказаних пунктах програми;

– можливість контролю та зміни стану процесора (РЗП, РС та SP), вмісту ОЗП та портів вводу-виводу за допомогою повно-функціонального налагоджувача;

– перевірка синтаксису програми;

– робота з ГІПВ.

Редактор ЕМПС

На рисунку Б.1 наведено зовнішній вигляд редактора ЕМПС.

Текстовий редактор програми «Емулятор CPU I8080» містить в собі всі стандартні механізми, що потрібні для редагування тексту: копіювання, вставка та вирізання тексту. Заголовок вікна містить ім’я відкритого файлу. Символ * показує, що вміст поточного файла був змінений. В текст програми можна вставляти коментарі, що починаються з символу «**;**» та продовжуються до кінця рядку.

При виборі пункту меню ***Сервис → Опции редактора...,*** налаштуванн редактора можна змінити. У нижній частині редактора міститься вікно повідомлень компілятора. Користувач може змінювати розміри цього вікна, закривати його, а також очищувати список повідомлень вибором відповідного пункту спливаючого меню.

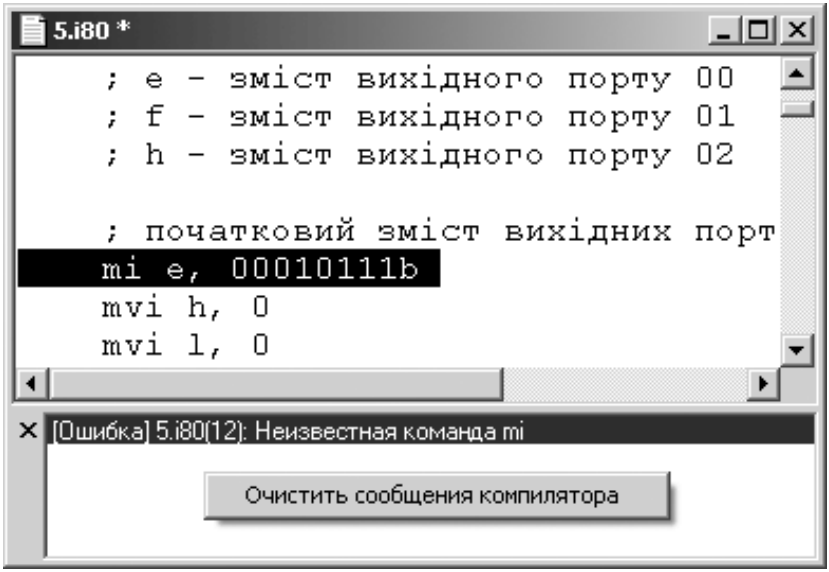


Рисунок Б.1 – Вікно Редактора

Команди редактора:

Ctrl+X – вирізання фрагменту до буфера обміну;
Ctrl+C – копіювання виділеного фрагменту до буфера обміну;
Ctrl+V – вставка вмісту буфера обміну;
Ctrl+Del – вилучання фрагменту програми;
Ctrl+A – виділення усієї програми;
Ctrl+Z – відміна останньої операції.
Ctrl+F – пошук тексту у програмі;
Ctrl+R – заміна попередньо знайденого тексту.

Директиви компілятора

ЕМПС має ряд додаткових директив, необхідних для вірної організації коду. Розглянемо ці директиви:

org адреса – встановлення адреси формування коду програми. Директив ORG в тілі програми може бути декілька.

мітка: equ значення – визначення констант. Мітці надається значення, вказане після директиви. При завданні значень констант можна використовувати як число, так і внутрішню змінну ЕМПС $, яка має значення поточної адреси. При використанні $, можна до $ додавати або віднімати деяке значення.

.startup – визначає точку входження до програми. У тілі програми може зустрічатися лише один раз.

include «ім'я файла» – підключення до тіла програми зовнішнього файла з текстом процедур або описів констант. Файл буде вставлений після описаної директиви. Мітки в файлах не повинні повторюватися, а також у допоміжних файлах не повинна зустрічатися директива ***.startup.***

[мітка] db значення – розміщує у пам'яті дані, що перелічені після директиви. Значення необхідно записувати через кому. Дані будуть розміщені послідовно, розмірність – байт.

[мітка] db N dup (значення) – заповнює ***N*** чарунок пам'яті заданим значенням. Якщо замість значення вказати «?», то ЕМПС зарезервує місце завдовжки ***N***.

Директива ***dw*** має такий синтаксис, як і ***db***, але розмірність – слово.

Значення внутрішньої змінної $ можна використовувати як константу при визначенні команд пересилки даних, а також у командах передачі управління. До $ можна як додавати, так і віднімати деяке значення, що не повинне перевищувати 65535.

Налагоджувач ЕМПС

Налагоджувач використовується для емуляції МПС I8080 (рис. Б.2). Він складається з 8 основних частин:

1. **Список команд.** Цей список поділений на 3 стовпця: адреса першого байта команди, код команди, а також сама команда.

Зелена стрілка вказує на чергову команду, що буде виконуватися. При виборі пункту меню ***Выполнить→Добавить точку зупину,*** натисканні клавіші ***F5*** або при натисканні мишею на першу чарунку потрібного рядку можна додавати або вилучати точку зупинки. Переміщуватися за списком можна за допомогою клавіш «↑» та «↓», а також за допомогою скролера.

2. **Зміст ОП.** Показує значення кожного байта ОП. Переміщуватися за списком можна тими ж способами, що й в попередньому списку.
3. **Такти та дозвіл переривань.** Відображає кількість тактів, на протязі яких виконувалася програма, а також значення сигналу дозволу переривань. Спливаюче меню в даній панелі містить пункт «Изменить INTE», що служить для зміни значення INTE.
4. **РЗП.** Панель відображає поточний стан регістрів. Вибираючи відповідний пункт спливаючого меню, можна змінювати формат відображення, збільшувати, зменшувати на одиницю, очищати, а також заносити власне значення до кожного регістра. Змінювати значення регістра можна також подвійним натисканням на відповідний регістр.
5. **Прапорці.** Панель відображає поточний стан прапорців. Вибираючи відповідний пункт спливаючого меню, можна змінювати значення кожного прапорця. Це також можна робити подвійним натисканням на відповідний регістр.
6. **Порти вводу.** Панель відображає значення останніх даних, що поступали на порти вводу. Спливаюче меню містить пункти для зміни даних, а також формату їх відображення на панелі. Переміщуватися за списком можна за допомогою клавіш «↑» та «↓», а також за допомогою скролера.
7. **Порти виводу.** Панель відображає значення останніх даних, що поступали на порти виводу. Спливаюче меню містить пункти для зміни формату відображення даних. Переміщуватися за списком можна тими ж способами, що й в попередньому пункті.

При виборі пункту меню ***Сервис→Опции отладчика...***, опції налагоджувача можна змінювати.

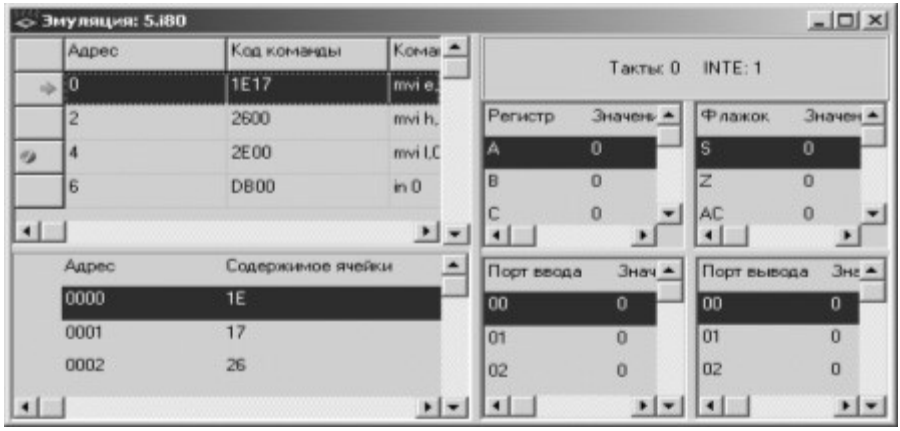


Рисунок Б.2 – Вікно Налагоджувача

Команди налагоджувача:

F4 – запуск програми до позиції курсору;
F5 – встановлення контрольної точки;
F7 – трасування за однією командою;
F8 – трасування за командою без входу до підпрограми;
F9 – запуск програми без перекомпіляції;
Ctrl+F2 – зупин програми (встановлення PC на початок програми).

Графічний інтерпретатор портів виводу описаний у Додатку Л.

Меню ЕМПС

Файл:
Новый – (елемент по умовчанню; виконується при подвійному натисканні напису «Файл») – створює новий файл;
Открыть – відкриває файл для перегляду та редагування;
Сохранить – збереження поточного файлу;
Сохранить как... – збереження файлу під іншим ім'ям;
Закрыть – закриває активне вікно;
Закрыть всё – закриває усі вікна додатку, крім головного;
Предварительный просмотр – перегляд та друк змісту активного вікна;

Печать – безпосередній друк змісту активного вікна;
Выход – завершення роботи програми.

Правка:

Отменить – відміна останньої операції у текстовому редакторі;
Вырезать – перенос виділеного фрагменту до буфера обміну;
Копировать – копіювання фрагменту до буфера обміну;
Вставить – вставка з буфера обміну до позиції курсору;
Удалить – вилучання фрагменту;
Выделить всё – виділення всього тексту програми.

Поиск:

Найти... – пошук фрагменту тексту;
Заменить... – пошук фрагменту та його заміна.

Вид:

Панель инструментов – налагодження панелі інструментів:
Стандартная – стандартний вид панелі інструментів;
Скрыть(Отобразить) – керування видимістю панелі;
Настроить... – налагодження панелі інструментів;
Строка статуса – керування видимістю рядку статусу.

Выполнить:

Компилировать... – компіляція програми;
Проверить синтаксис... – перевірка синтаксису програми;
Выполнить – компіляція та запуск програми;
Пауза – пауза при виконанні програми;
Остановить – завершити виконання програми;
Шаг вне – покрокове виконання програми без заходу допроцедури;
Шаг внутрь – покрокове виконання програми із заходом до процедури;
Выполнить до курсора – виконання програми до курсору;
Добавить точку останова – додавання в програму точки зупинки.

Вставка:

Заголовок – вставка до програми стандартного шаблону:

```
; Лабораторная работа №
; ()

org 0
.startup   .
```

Сервис:

Опции редактора... – зміна опцій редактора;
Опции отладчика... – зміна опцій налагоджувача.

Окно:

Каскад – розташування вікон каскадом;
Упорядочить – упорядкування вікон:
По горизонтали;
По вертикали;
Упорядочить значки – упорядкування значків згорнутих вікон;
Свернуть всё – згорнути усі вікна, крім головного;
Восстановить всё – відновлення раніше згорнутих вікон.

Помощь:

Справка – запуск довідки про програму «Эмулятор CPU I8080», команди 8-ми розрядного Асемблера та директиви компілятора;
О программе... – інформація про версію та авторів програми.

Додаток В Система команд мікропроцесора I8080

Система команд однокриштального мікропроцесора I8080 має команди трьох форматів. Перший байт команди містить інформацію про формат команди, код операції, вид адресації та про регістри або регістрові пари, якщо вони приймають участь у виконанні операцій.

У двобайтових командах другий байт (***B2***) містить 8-розрядний операнд або 8-розрядну адресу пристрою вводу чи виводу.

У трибайтових командах другий та третій байти (***B2***, ***B3***) містять 16-розрядні адреси (у командах із прямою адресацією пам'яті) або 16-розрядні операнди (у командах завантаження регістрових пар або покажчика стека). Другий байт трибайтової команди містить молодший байт числа, третій – старший.

У таблиці В.1 наведені мнемонічні позначення та опис команд мікропроцесора i8080. При цьому використовуються наступні умовні позначення :

r - регістр загального призначення;
rp - пара регістрів загального призначення;
PC - програмний лічильник (лічильник команд);
SP - покажчик стека;
M - чарунка пам'яті;
B2,B3 - другий та третій байти команд

Таблиця В.1 - Система команд мікропроцесора I8080

Команда	Довжина команди, байт	Число тактів	Опис команди	Ознаки
1	2	3	4	5
1 Команди пересилання даних				
MOV r1,r2	1	5	Пересилання даних із регістра **r**2 до регістра **r**1	-
MOV M,r (MOV r,M)	1	7	Пересилання даних із регістра **r** до пам'яті за адресою, що зберігається у регістровій парі **H-L** (із пам'яті до регістру **r**)	-
XCHG	1	4	Обмін даними між парами регістрів **H-L** і **D-E**	-

Продовження таблиці В.1

1	2	3	4	5
MVI r,<B2> (MVI M,<B2>)	2	7 (10)	Занесення байта даних до регістру **r** (до пам'яті)	-
LXI rp <2байта>	3	10	Занесення двох байтів даних у пару регістрів **(B-C, D-E, H-L, SP**). Третій байт команди заноситься в старший регістр, а другий - у молодший	-
LDAX rp	1	7	Завантаження в накопичувач вмісту чарунки, яка опосередковано адресується парою регістрів **rp (B-C, D-E)**	-
LDA <адреса>	3	13	Завантаження накопичувача вмістом чарунки за вказаною адресою. 2-й байт команди - молодший байт адреси, 3-й байт - старший	-
STAX rp	1	7	Занесення вмісту накопичувача до чарунки, яка опосередковано адресується парою **rp**	-
STA, <адреса>	3	13	Занесення вмісту накопичувача до чарунки за вказаною адресою	-
LHLD <адреса>	3	16	Завантаження регістра **L** вмістом чарунки за вказаною адресою, а регістр **H** - чарунки з адресою на одиницю більше	-
SHLD <адреса>	3	16	Занесення вмісту регістрів **H** і **L** до пам'яті (аналогічно команді **LHLD**)	-
2 Арифметичні команди				
ADD r (ADD M)	1	4 (7)	Складання змісту регістра **r** (чарунки пам'яті) і накопичувача	Z,S,P,C
ADC r (ADC M)	1	4 (7)	Складання змісту регістра **r** (чарунки пам'яті) і накопичувача з бітом перенесення	Z,S,P, C,AC

Продовження таблиці В.1

1	2	3	4	5
SUB r (SUB M)	1	4 (7)	Віднімання змісту регістра **r** (чарунки пам'яті) від змісту накопичувача	Z,S,P, C[1],AC[2]
SBB r (SBB M)	1	4 (7)	Віднімання змісту регістра **r** (чарунки пам'яті) та біта перенесення від змісту накопичувача	Z,S,P, C[1],AC[2]
ADI ,<байт>	2	7	Складання байта зі змістом накопичувача	Z,S,P,C, AC
ACI ,<байт>	2	7	Складання байта зі змістом накопичувача та бітом перенесення	Z,S,P,C, AC
SUI,<байт>	2	7	Віднімання байта із змісту накопичувача	Z,S,P, C[1],AC[2]
SBI,<байт>	2	7	Віднімання байта команди і біта перенесення від змісту накопичувача	Z,S,P, C[1],AC[2]
DAD rp	1	10	Складання змісту пари регістрів **rp** (**B-C,D-E,H-L,SP**) зі змістом пари регістрів **H-L**	C
INR r (INR M)	1	5 (10)	Збільшення змісту регістра **r** (чарунки пам'яті) на одиницю	Z,S,P, AC
DCR r (DCR M)	1	5 (10)	Зменішення змісту регістра **r** (чарунки пам'яті) на одиницю	Z,S,P, AC[2]
INX rp (DCX rp)	1	5	Збільшення (зменьшення) змісту пари регістрів **rp** (**B-C,D-E,H-L,SP**) на одиницю	-
DAA	1	4	Перетворення змісту накопичувача в двійково-десятичний код	Z,S,P,C, AC
3 Логічні команди				
ANA r (ANA M)	1	4 (7)	Порозрядне 'І' над змістом регістра **r** (чарунки пам'яті) і накопичувача	Z,S,P, C=0, AC=0
XRA r (XRA M)	1	4 (7)	Порозрядне виключаюче 'АБО' над змістом регістра **r** (чарунки пам'яті) і накопичувача	Z,S,P, C=0, AC=0

Продовження таблиці В.1

1	2	3	4	5
ORA r (ORA M)	1	4 (7)	Порозрядне ‘АБО’ над змістом регістра **r** (чарунки пам’яті) і накопичувача	Z,S,P, C=0, AC=0
CMP r (CMP M)	1	4 (7)	Порівняння змісту регістра **r** (чарунки пам’яті) і накопичувача	(Z,S,P, C,AC)[3]
ANI,<байт>	2	7	Порозрядне ‘І’ над змістом накопичувача і байтом	Z,S,P, C=0, AC=0
XRI,<байт>	2	7	Порозрядне виключаюче ‘АБО’ над змістом накопичувача і байтом	Z,S,P, C=0, AC=0
ORI,<байт>	2	7	Порозрядне ‘АБО’ над змістом накопичувача і байтом	Z,S,P, C=0, AC=0
CPI,<байт>	2	7	Порівняння байта зі змістом накопичувача	(Z,S,P, C,AC)[3]
RLC (RRC)	1	4	Циклічний зсув змісту накопичувача вліво (вправо)	C[4]
RAL (RAR)	1	4	Циклічний зсув змісту накопичувача вліво (вправо) через перенесення	C[4]
CMA	1	4	Порозрядне інвертування накопичувача	-
STC	1	4	Встановлення ознаки перенесення в одиницю	C=1
CMC	1	4	Інвертування ознаки перенесення C	$C=\overline{C}$
4 Команди переходів				
PCHL	1	5	Занесення змісту регістрів **H**, **L** в лічильник команд (зміст **H** – в старший байт, **L** – в молодший)	-
JMP,<адреса>	3	10	Безумовний перехід по вказаній адресі	-
JC/(JNC), <адреса>	3	10	Перехід при наявності (відсутності) перенесення	-
JZ/(JNZ), <адреса>	3	10	Перехід при наявності (відсутності) нуля	-
JP/(JM), <адреса>	3	10	Перехід при плюсі (мінусі)	-

Продовження таблиці В.1

1	2	3	4	5
JPE/(JPO), <адреса>	3	10	Перехід при парності (непарності)	-
CALL,<адреса>	3	17	Виклик підпрограми	-
CC/(CNC), <адреса>	3	11 (17)	Виклик підпрограми при наявності (відсутності) перенесення	-
CZ/(CNZ), <адреса>	3	11 (17)	Виклик підпрограми при наявності (відсутності) нуля	-
CP/(CM), <адреса>	3	11 (17)	Виклик підпрограми при плюсі (мінусі)	-
CPE/(CPO), <адреса>	3	11 (17)	Виклик підпрограми при парності (непарності)	-
RET	1	10	Повернення з підпрограми	-
RC/(RNC)	1	5 (11)	Повернення при наявності (відсутності) перенесення	-
RZ/(RNZ)	1	5 (11)	Повернення при наявності (відсутності) нуля	-
RP/(RM)	1	5 (11)	Повернення при плюсі (мінусі)	-
RPE/(RPO)	1	5 (11)	Повернення при парності (непарності)	-
RST,<номер>	1	11	Повторне запущення з адреси 8 х номер (0,8,…,56)	-
5 Команди вводу-виводу та управління				
IN,<порт>	2	10	Ввод даних із вказаного порту до накопичувача	-
OUT,<порт>	2	10	Вивід даних із накопичувача до вказаного порту	-
PUSH rp	1	11	Занесення змісту пари регістрів **rp** (**B-C,D-E,H-L, PSW**) до стека	-
POP rp	1	10	Видання даних зі стека в пару регістрів **rp** (**B-C,D-E, H-L,PSW**)	(Z,S,P,C AC)[6]
XTHL	1	18	Обмін даними між верхівкою стека та парою регістрів **H-L**	-
SPHL	1	5	Занесення в покажчик стека змісту регістрів **H-L**	-
DI/EI	1	5	Заборонити/дозволити пере-ривання	-
NOP	1	4	Порожня операція	-
HLT	1	7	Зупиню	-

Примітки:

1 встановлюється при наявності займу до старшого розряду, у протилежному випадку скидається;

2 встановлюється при наявності займу зі старших чотирьох розрядів в молодші, у протилежному випадку скидається;

3

- **Z** встановлюється, якщо зміст регістра та байта даних дорівнює змісту накопичувача ;
- **S,C,** якщо зміст регістра або байта даних більше змісту накопичувача;
- **AC,** якщо зміст молодших чотирьох розрядів регістра й байта даних більше змісту молодших чотирьох розрядів накопичувача;
- **P,** якщо байт різниці між змістом накопичувала та змістом регістра або байта даних містить парне число одиниць;

4 стан ознаки дорівнює значенню висунутого з накопичувача двійкового розряду;

5 у знаменнику дробу вказано кількість тактів при виконанні умов, в чисельнику – при невиконанні;

6 за командою ***POP PSW*** ознаки установлюються відповідно до значення розрядів слова, яке занесено до стека, при інших значеннях гр ознаки не змінюються

Додаток Д Призначення та склад регістра ознак

Регістр ознак використовується для зберігання ознак результатів операцій. В МП І8080 з цією метою використовуються тригери умов (прапорці), які для зручності об'єднано у спеціальний регістр (рис. Д.1).

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

Рисунок Д.1 – Склад регістра прапорців

Прапорці мають такий сенс:

S – ознака знаку: якщо результат < 0, то S=1;

Z – ознака нуля: якщо результат = 0, то Z=1;

AC – ознака допоміжного переносу: AC=1, якщо є перенесення з розряду D3 до розряду D4;

P – ознака парності: якщо у двійковому коді результату кількість одиниць є парне число, то P=1;

C – ознака перенесення зі старшого розряду: у випадку перенесення C=1.

Кожен із перелічених п'яти розрядів є ознакою (прапорцем), а інші розряди мають фіксовані значення.

За допомогою регістра ознак реалізуються розгалужені алгоритми.

Додаток Ж Способи організації часових затримок

При організації часової затримки використовують дані табл.В.1 про час виконання тієї чи іншої команди. В залежності від потрібної тривалості затримки доцільно розділити всі способи організації затримки на дві групи:

– способи організації затримки малої тривалості;

– способи організації затримки великої тривалості;

Для обох груп затримка кратна тривалості одного машинного такту $t_{зад}$=0,5 мкс.

Для організації затримок малої тривалості використовуються команди, що не змінюють інформацію та стан ЦПЕ. Рекомендується використовувати команди NOP; MOV A,A; ADI 0 і пару команд PUSH-POP, які складають у ланцюжок.

Для організації затримок великої тривалості використовують однокаскадний або багатокаскадний лічильник. Типова схема однокаскадного лічильника показана на рисунку Ж.1.

Час затримки в машинних тактах розраховується за формулою:

$$N_{зад}=T_{вст}+N_{в}(T_{рах}+T_{дод.\,в})+T_{дод.\,к} \qquad \text{(В.1)}$$

де $N_{в}$ - кількість циклів лічильника;

$T_{вст}$ - кількість тактів виконання команди встановлення початкового стану лічильника;

$T_{рах}$ - мінімальна затримка у циклі рахування;

$T_{дод.\,в}$ - додаткова затримка у циклі рахування;

$T_{дод.\,к}$ - додаткова затримка після закінчення циклу;

Вибір величин, що входять до цього виразу, здійснюється методом підбору за даними (кількість тактів) табл. В.1.

Текст програми, що реалізує схему однокаскадного лічильника, наведено у таблиці Ж.1.

Таблиця Ж.1 - Текст програми однокаскадного лічильника

Адреса ОЗП	Код команди	Мітка	Мнемоніка	Кільк. тактів	Формат	Текст коментарю
04A0 04A1	06 C8H		**MVI B,N_B**	7	2	Встановлення пачатк. стану
04A2	00	**M1:**	**NOP**	4	1	Додаткова затримка у циклі
04A3	05		**DCR B**	5	1	Мінімальна затримка лічильника
04A4 04A5 04A6	C2 A2 04		**JNZ M1**	10	3	
04A7	00		**NOP**	4	1	Додаткова затримка після закінчення рахування
04A8	7F		**MOV A,A**	5	1	

При $N_в$=200 дана програма реалізує затримку в машинних тактах, яка дорівнює

$$N_{зад}=7+200*(15+4)+9=3816$$

Звідси час затримки:

$$T_{зад}=t_{зад}*N_{зад}=0,5*3816=1908 мкс.$$

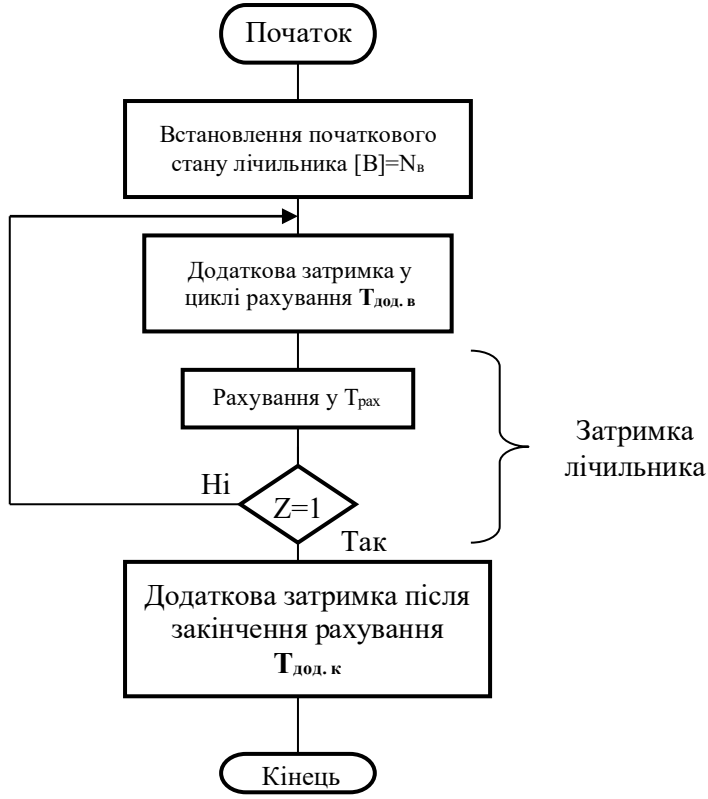


Рисунок Ж.1 - Типова схема однокаскадного лічильника.

Використовуючи багатокаскадні лічильники, можливо отримати практично будь-яку затримку.

Додаток К Обробка окремих бітів

Як відомо, байт є одиницею інформації, що мінімально адресується, тобто мікропроцесор за одне звернення до пам'яті чи зовнішнього пристрою може прийняти або видати один байт інформації. Але часто виникають ситуації, коли потрібно обробляти окремі біти (розряди) чисел, що передаються.

Наприклад, в системі використовуються 8 датчиків, які можуть бути або включені (стан 1), або вимкнуті (стан 0). Для збереження стану цих датчиків доцільно для кожного датчику виділяти не один байт, а лише один біт. Таким чином, інформація про стан 8 датчиків буде упакована в 8 бітів, тобто один байт.

Для виділення окремих розрядів числа використовують спеціальні «слова-маски» – двійкові числа та відповідні логічні команди (див. табл. В.1 Додатку В).

Якщо треба протестувати значення конкретного біта в числі, маска повинна складатися з нулів у всіх розрядах, крім того, що аналізується. Потім виконується команда порозрядного «І» над вмістом акумулятора та маскою.

В результаті в акумуляторі всі розряди, крім того, що аналізується, будуть містити нулі. Якщо він нульовий, то і все число дорівнює нулю, і прапорець Z (ознака нульового результату) буде встановлений в 1. Якщо Z=0 (ненульовий результат), розряд, що аналізується, дорівнює 1.

Наприклад, потрібно проаналізувати третій розряд числа. Тоді в якості маски буде число $0000\mathbf{1}000_{В}=08_{Н}$ (не забувайте, що відлік йде від 0-го розряду), а команда – ***ani 08h***.

Якщо треба примусово встановити певний розряд в 1, не змінюючи інших, використовується аналогічна маска, але команда буде «АБО». Наприклад, ***ori 08h.***

В разі необхідності можна виділяти не лише одиночні розряди, а й групи розрядів, використовуючи відповідну маску.

Додаток Л Графічний інтерпретатор портів виводу

Графічний інтерпретатор портів виводу (ГІПВ) призначений для складання програм імітації управління технологічними процесами (рис. Л.1).

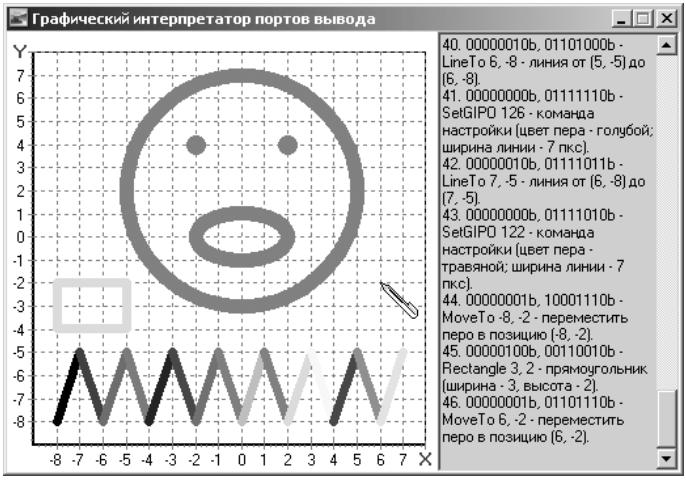


Рисунок Л.1 – Вікно ГІПВ

Для відображення ГІПВ при виконанні програми необхідно в опціях налагоджувача задати опцію «***Отображать графический интерпретатор портов вывода***». По умовчанню ГІПВ не відображається.

У лівій частині вікна знаходиться графічна область для рисування всіляких графічних примітивів, у правій – список команд, що поступили на ГІПВ, з їх описом.

Робота з ГІПВ здійснюється через порт **05**. Значення, що поступають на цей порт, сприймаються (інтерпретуються) ГІПВ як деякі керуючі команди. Кожна така команда займає 2 байти, тому може бути передана на виконання тільки за 2 кроки: спочатку на порт 05h посилається перший байт команди, а потім другий байт. Перший байт містить код команди (дана версія ГІПВ підтримує 7 команд), а другий – її параметри.

Система команд ГІПВ наведена в таблиці Л.1.

Таблиця Л.1 – Система команд ГІПВ

Команда	Виклик	Коментар
1	2	3

SetGIPO W, C	`mvi a,00h`
	`out 05h`
	`mov a,B2`
	`out 05h`
		Команда налагодження ГІПВ. Молодший напівбайт числа ***B2*** задає колір пера, старший напівбайт – ширину лінії. Для завдання кольору є 16 констант (див. нижче), ширина лінії відповідно може бути від 1 пкс до 15 пкс. Якщо старший напівбайт числа ***B2*** буде дорівнювати нулю, то ширина лінії пера встановиться в 1 пкс. Наприклад, наступний фрагмент програми
		`mvi a,00h`
		`out 05h`
		`mvi a,01111110b`
		`out 05h`
		встановлює ширину лінії в 7 пкс (0111) та голубий колір пера (1110).

MoveTo X, Y	`mvi a,01h`
	`out 05h`
	`mov a,B2`
	`out 05h`
		Переміщення пера до зазначеної позиції. Молодший напівбайт числа ***B2*** задає координату по осі **X** (від -8 до 7), старший напівбайт – координату по осі **Y** (від -8 до 7).
		Наприклад,
		`mvi a,01h`
		`out 05h`
		`mvi a,01101110b`
		`out 05h`
		переміщує перо в точку X = 6 (0110), Y = -2 (1110).

Продовження таблиці Л.1

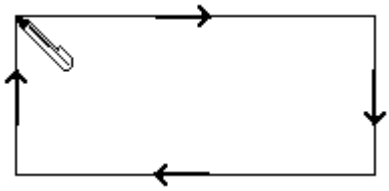
1	2	3

LineTo X, Y	`mvi a,02h`
	`out 05h`
	`mov a,B2`
	`out 05h`
		Рисування лінії з поточної позиції пера в точку, що вказана другим байтом команди. Як і для команди MoveTo, молодший напівбайт числа ***B2*** задає координату по осі **X**, старший напівбайт – координату по осі **Y**. Наприклад,
		
		`mvi a,02h`
		`out 05h`
		`mvi a,01111011b`
		`out 05h`
		
		рисує лінію від поточної точки в точку X = 7 (0111), Y = -5 (1011).

PutPixel	`mvi a,03h`
	`out 05h`
	`mov a,B2`
	`out 05h`
		Використовуючи поточні значення для ширини лінії та кольору пера, в поточній позиції ставиться точка. Значення другого байту команди ***B2*** не має значення. Але його слід посилати на порт 05, інакше команда не буде виконана. Наприклад,
		
		`mvi a,03h`
		`out 05h`
		`mvi a,0`
		`out 05h`
		
		ставить точку в поточну позицію пера.

Продовження таблиці Л.1

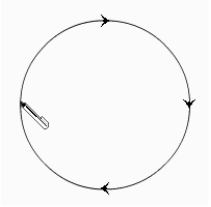
1	2	3

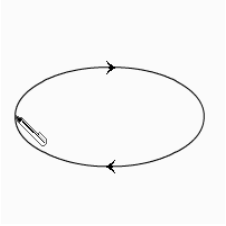
Rectangle W, H	`mvi a,04h`
	`out 05h`
	`mov a,B2`
	`out 05h`
		Рисування прямокутника з шириною, що міститься в старшому напівбайті числа ***B2*** (від 0 до 15), та з висотою, що міститься в молодшому напівбайті числа ***B2*** (від 0 до 15). Прямокутник рисується з поточної позиції пера в напрямку, вказаному на рисунку. Таким чином, після виконання команди, перо повертається в початкове положення.
		 
		Наприклад,
		`mvi a,04h`
		`out 05h`
		`mvi a,00110010b`
		`out 05h`
		рисує прямокутник з шириною - 3 та висотою - 2.

Circle R	`mvi a,05h`
	`out 05h`
	`mov a,B2`
	`out 05h`
		Рисування кола з радіусом, що міститься у молодшому напівбайті числа ***B2*** (від 0 до 15). Значення старшого напівбайта числа ***B2*** не має значення. Коло рисується з поточної позиції пера у напрямку, що вказаний на рисунку. Тому, як у випадку виконання команди **Rectangle,** перо повертається у початкове положення.

Продовження таблиці Л.1

1	2	3

		
		Наприклад, програма
		`mvi a,05h`
		`out 05h`
		`mvi a,00000101b`
		`out 05h`
		рисує коло з радіусом - 5.

Ellipse a, b	`mvi a,06h`
	`out 05h`
	`mov a,B2`
	`out 05h`
		Рисування еліпса з радіусами, що містяться у напівбайтах числа ***B2*** (від 0 до 15). Еліпс рисується з поточної позиції пера у напрямку, що вказаний на рисунку. Тому, як і у випадку виконання команд ***Rectangle*** та ***Circle,*** перо повертається у початкове положення.
		 
		Наприклад, програма
		`mvi a,06h`
		`out 05h`
		`mvi a,00110101b`
		`out 05h`
		рисує еліпс з радіусами - 3 та 5.

Колір задається константами, що наведені у таблиці Л.12.

Таблиця Л.2 – Список констант для завдання кольорів

Константа	Колір	Константа	Колір
0	чорний	8	сріблистий
1	малиновий	9	червоний
2	зелений	10	трав'яний
3	маслиновий	11	жовтий
4	темно-синій	12	синій
5	бузковий	13	рожевий
6	бірюзовий	14	блакитний
7	сірий	15	білий

Розглянемо фрагмент програми, результат роботи якої можна побачити на рисунку Л.2.

```
org 0
circle:
mvi a,05h
out 05h
mvi a,1b
out 05h
ret

org 10
.startup
mvi a,00h
out 05h
mvi a,11000000b
out 05h
mvi a,1h
out 5h
mvi a,10000111b
out 5h
mvi a,02h
out 05h
mvi a,10010110b
out 05h
mvi a,01h

out 05h
mvi a,10010101b
out 05h
call circle
mvi a,1h
out 05h
mvi a,10110111b
out 05h
call circle
mvi a,1h
out 05h
mvi a,11010101b
out 05h
call circle
mvi a,1h
out 05h
```

```
mvi a,11110110b
out 05h
mvi a,2h
out 05h
mvi a,00000111b
out 05h
mvi a,2h
out 05h
mvi a,01110111b
out 05h
mvi a,01h
out 05h
mvi a,11000110b
out 05h
mvi a,2h
out 05h
mvi a,11000001b
out 05h
mvi a,01h
out 05h
mvi a,10110000b
out 05h
call circle
mvi a,01h
out 05h
mvi a,11110000b
out 05h
call circle
mvi a,01h
out 05h
mvi a,00110000b
out 05h
call circle
mvi a,01h
out 05h
mvi a,10111011b
out 05h
call circle
mvi a,01h
out 05h
mvi a,11111011b
out 05h
call circle
```

```
mvi a,01h
out 05h
mvi a,00111011b
out 05h
call circle
mvi a,01h
out 05h
mvi a,00000001b
out 05h
mvi a,02h
out 05h
mvi a,00000100b
out 05h
mvi a,02h
out 05h
mvi a,00010101b
out 05h
mvi a,02h
out 05h
mvi a,01110101b
out 05h
mvi a,01h
out 05h
mvi a,01000001b
out 05h
mvi a,02h
out 05h
mvi a,01000010b
out 05h
mvi a,02h
out 05h
mvi a,01010011b
out 05h
mvi a,02h
out 05h
mvi a,01110011b
out 05h
mvi a,01h
out 05h
mvi a,00111100b
out 05h
mvi a,02h
out 05h
```

```
mvi a,00101101b
out 05h
mvi a,02h
out 05h
mvi a,10001101b
out 05h
mvi a,01h
out 05h
```

```
mvi a,11001010b
out 05h
mvi a,02h
out 05h
mvi a,11001000b
out 05h

hlt
```

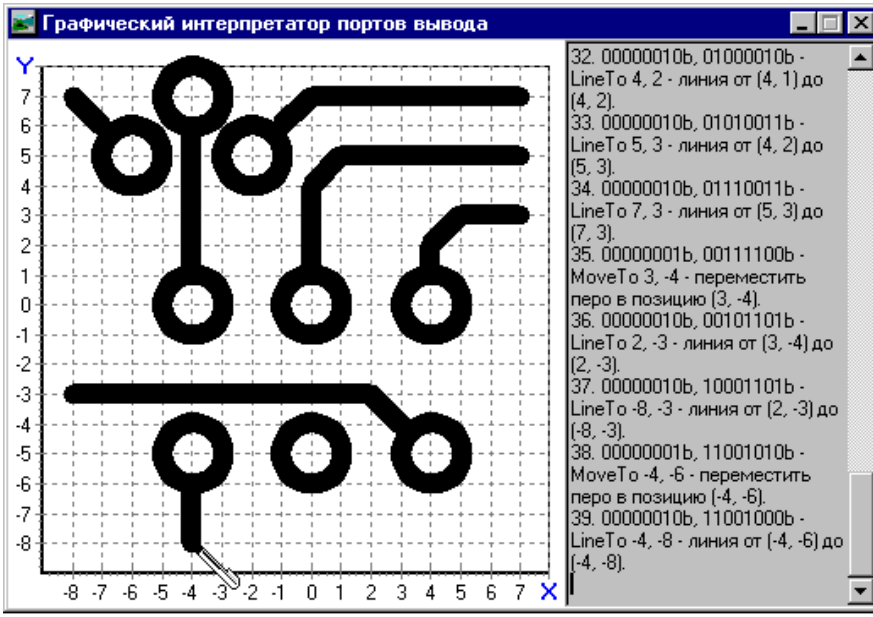


Рисунок Л.2 – Результат роботи програми

Спливаюче меню інтерпретатора містить один пункт – «Очистить графический інтерпретатор», що дозволяє очистити графічну область.

Додаток М Варіанти завдань для самостійної роботи

На мові Асемблера мікропроцесора I8080 скласти програми, які реалізують такі завдання :

1. У масиві чисел, розташованому з адреси 0150 Н до 0200 Н, встановити в «1» старші і в «0» молодші розряди чисел.
2. Підрахувати, скільки разів у масиві чисел з адресами 0200-02FF зустрічаються числа, менші, ніж перший елемент масиву.
3. У масиві чисел, які зберігаються в ОЗП у чарунках з адресами з 0500 до 05DE, замінити від'ємні числа нулем.
4. Підрахувати число додатних елементів у масиві з адресами 0600-06AE.
5. Здійснити обмін інформації між масивами, розташованими з адрес 0400 і 0500, які вміщають по 21 елементу.
6. Програма формування тимчасової затримки тривалістю 0,3 с (f = 1,2 МГц).
7. Зсунути вміст чарунки пам'яті з адресою 040F на число бітів, яке визначається вмістом регістру **D**.
8. Скласти два числа, які знаходяться у чарунках з адресами 0100Н та 0200Н, і вивести молодший напівбайт у порт 01, а старший напівбайт – порт 02.
9. Масив чисел у чарунках ОЗП з адресами з 0500 до 05CD переписати у ті ж самі чарунки у зворотному порядку.
10. З масиву чисел із адресами з 0400 до 04CD видавати вміст чарунок із непарними адресами у вихідний порт, помножуючи їх на 2.
11. Переписати 10 послідових байтів із одної області пам'яті (починаючи з адреси 2200 Н) у другу (з адреси 2300 Н).
12. Програма пересилки у старший розряд вихідного порту логічної одиниці з частотою 100 Гц (f = 2 МГц).
13. Підрахувати в масиві чисел кількість одиничних розрядів.
14. У масиві чисел упаковані показання 8 датчиків, зняті у різні моменти часу. Підрахувати, скільки разів спрацьовував датчик N.
15. Замінити у масиві з адресами 0500Н-05ССН числа, більші 9, числом 10.
16. В масиві чисел із адресами з 0500Н до 05FFH числа, які зберігаються у чарунках із парними адресами, замінити на 0.