НАВЧАЛЬНО-МЕТОДИЧНА ЛІТЕРАТУРА

Тернопільський національний технічний університет імені Івана Пулюя

Кафедра автоматизації технологічних процесів і виробництв

Лабораторна робота № 3

з курсу

Проектування мікропроцесорних систем керування технологічними процесами

Програмування мікроконтролера MCS51 з використанням програмної моделі EdSim51. Арифметичні та логічні команди MCS51

Тернопіль 2023

Методичні вказівки для виконання лабораторної роботи № 3 «Програмування мікроконтролера MCS51 з використанням програмної моделі EdSim51. Арифметичні та логічні команди MCS51» з курсу «Проектування мікропроцесорних систем керування технологічними процесами»/Укл.: Медвідь В.Р., Пісьціо В.П. - Тернопіль ТНТУ, 2023 - 12 с.

Розглянуто і затверджено на засіданні кафедри автоматизації технологічних процесів і виробництв (протокол № 1 від 30.08.2023 року)

Лабораторна робота № 3

Програмування мікроконтролера MCS51 з використанням програмної моделі EdSim51. Арифметичні та логічні команди MCS51

1. Команди MCS51

Система команд мікроконтролера MCS51 містить 111 базових команд, які зручно розділити **за функціональною ознакою на п'ять груп: команди передачі даних, арифметичних операцій, логічних операцій, передачі управління і операцій з бітами.**

Більшість команд мають формат один або два байти і виконуються за один або два машинних циклу. При тактовій частоті 12 МГц тривалість машинного циклу складає 1 мкс.

Склад операндів MCS51 включає в себе **операнди чотирьох типів: біти, 4-бітні цифри, байти і 16-бітні слова.** Є також можливість адресації окремих бітів блоку регістрів спеціальних функцій (РСФ) і портів. **Для адресації бітів використовується пряма 8-бітна адреса (bit)**.

Чотирибітні операнди використовуються тільки під час операції обміну (команди SWAP і XCHD).

Восьмибітним операндом може бути комірка пам'яті програм або даних (резидентної або зовнішньої), константа (безпосередній операнд), регістри спеціальних функцій (РСФ), а також порти вводу/виводу.

Порти і РСФ адресуються тільки прямим способом. Байти пам'яті можуть адресуватися також і непрямим чином через **адресні регістри (R0, R1, DPTR і PC).**

Двобайтні операнди - це константи і прямі адреси, для подання яких використовуються другий і третій байти команди.

2. Правила запису програм на мові асемблера

Оригінальний текст програми на мові асемблера має певний формат. Кожна команда і директива є рядок:

МІТКА: ОПЕРАЦІЯ ОПЕРАНД (ОПЕРАНДИ) ; КОМЕНТАР

Поля можуть відділятися один від одного довільним числом прогалин і табуляцією.

Мітка

В поле мітки розміщується символічне ім'я комірки пам'яті, в якій зберігається зазначена команда або операнд. Мітка є буквено-цифровою комбінацію, що починається з літери. Використовуються тільки літери латинського алфавіту. Асемблер A51 допускає використання в мітках символу підкреслення **(_)**. **Мітка завжди завершується двокрапкою (:).**

Директиви асемблера перетворюються на двійкові коди, а тому не можуть мати міток. Виняток становлять директиви резервування пам'яті і визначення даних (DS, DB, DW). У директивах, що визначають символічні імена, в поле мітки записується визначене символічне ім'я, після якого двокрапка не ставиться.

В якості символічних імен та міток не можуть бути використані мнемокоди команд, директив та операторів асемблера, зарезервовані імена, а також мнемонічні позначення регістрів і інших внутрішніх блоків мікроконтролера.

Операція

В поле операції записується мнемонічне позначення команди або директиви асемблера, яке є скороченням (абревіатурою) повного англійського найменування виконуваної дії. Наприклад: MOV - move - перемістити, JMP - jump - перейти, DB - define byte - визначити байт.

Для мікроконтролера Intel 8051 використовується певний і обмежений набір мнемонічних кодів. Будь-який інший набір символів, розміщений в поле операції, сприймається асемблером як помилковий.

Операнди

У цьому полі визначаються операнди (або операнд), які беруть участь в операції. Команди асемблера можуть бути без-, одно- або двоопераднними. **Операнди розділяються комою (,)**.

Операнд може бути заданий безпосередньо або у вигляді його адреси (прямої або непрямої).

Безпосередній операнд представляється числом (MOV A, # 15) або символічним ім'ям (ADDC A, # OPER2) з обов'язковим покажчиком префіксу безпосереднього операнду (#).

Прямий доступ операнду може бути заданий мнемонічним позначенням (IN A, P1), числом (INC 40), символічним ім'ям (MOV A, MEMORY).

Визначенням непрямої адресації служить префікс @. У командах передачі управління операндом може бути число (LCALL 0135H), мітка (JMP LABEL), непряма адреса (JMP @A) або вираз (JMP $ - 2, де **$ - поточний вміст лічильника команд**).

Використовувані в якості операндів символічні імена і мітки повинні бути визначені, а числа представлені із зазначенням системи числення, для чого використовується суфікс (літера, що стоїть після числа): **B - для двійкової, Q - для вісімкової, D - для десяткової і H - для шістнадцяткової**. **Число без суфікса за замовчуванням вважається десятковим.**

Коментар

Це поле може бути використане для текстового або символьного пояснення логічної організації прикладної програми. Поле коментаря повністю ігнорується асемблером, а тому в ньому допустимо використовувати будь-які символи. За правилами мови асемблера **поле коментаря починається з крапки з комою (;)**.

3. Група команд арифметичних операцій

Дану групу утворюють 24 команди (табл.1), що виконують операції додавання, віднімання, множення і ділення байтів, десяткові корекції, инкременту / декременту байтів.

Команди ADD і ADDC допускають додавання акумулятора з рядом операндів. Аналогічно командам ADDC існують чотири команди SUBB, що дозволяє просто виконувати віднімання байтів і багатобайтових двійкових чисел. У МК51 реалізується розширений список команд инкременту / декременту байтів, введена команда инкременту 16-бітного регістру-покажчика даних.

Таблиця 1.Арифметичні операції

Назва команди	Мнемокод	КОП	Т	Б	Ц	Операція
Додавання акумулятора і регістру (n=0÷7)	ADD A, Rn	00l01rrr	1	1	1	(A) ← (A) + (Rn)
Додавання акумулятора і прямоадресованого байту	ADD A, ad	00100101	3	2	1	(A) ← (A) + (ad)
Додавання акумулятора і байту з РПД (i = 0,1)	ADD A, @Ri	0010011i	1	1	1	(A) ← (A) + ((Ri))
Додавання акумулятора і константи	ADD A, #d	00100100	2	2	1	(A) ← (A) + #d
Додавання акумулятора, регістру і перенесення	ADDC A, Rn	00111rrr	1	1	1	(A) ← (A) + (Rn) + (C)
Додавання аккумулятора, прямоадресованого байту і перенесення	ADDC A, ad	00110101	3	2	1	(A) ← (A) + (ad) + (C)
Додавання акумулятора, байту з РПД і перенесення	ADDC A, @Ri	0011011i	1	1	1	(A) ← (A) + ((Ri)) + (C)
Додавання акумулятора, константи і перенесення	ADDC A, #d	00110100	2	2	1	(A) ← (A) + # d + (C)
Десяткова корекція акумулятора	DA A	11010100	1	1	1	Якщо ($A_{0...3}$)>9 або ((AC)=1), то ($A_{0...3}$) ← ($A_{0...3}$) + 6, далі якщо ($A_{4...7}$) >9 або ((C)=1), то ($A_{4...7}$) ← ($A_{4...7}$) + 6
Віднімання від акумулятора регистру і займання	SUBB A, Rn	10011rrr	1	1	1	(A) ← (A) - (C) - (Rn)
Віднімання від акумулятора прямоадресованого байту і займання	SUBB A, ad	10010101	3	2	1	(A) ← (A) - (C) - ((ad))

Назва команди	Мнемокод	КОП	Т	Б	Ц	Операція
Віднімання від акумулятора байту з РПД і займання	SUBB A, @Ri	1001011i	1	1	1	(A) ← (A) - (C) - ((Ri))
Віднімання від акумулятора константи і займання	SUBB A, d	10010100	2	2	1	(A) ← (A) - (C) - #d
Інкремент акумулятора	INC A	00000100	1	1	1	(A) ← (A) + 1
Інкремент регистру	INC Rn	00001rrr	1	1	1	(Rn) ← (Rn) + 1
Інкремент прямоадресованого байту	INC ad	00000101	3	2	1	(ad) ← (ad) + 1
Инкремент байту в РПД	INC @Ri	0000011i	1	1	1	((Ri)) ← ((Ri)) + 1
Інкремент покажчика даних	INC DPTR	10100011	1	1	2	(DPTR) ← (DPTR) + 1
Декремент акумулятора	DEC A	00010100	1	1	1	(A) ← (A) – 1
Декремент регістру	DEC Rn	00011rrr	1	1	1	(Rn) ← (Rn) – 1
Декремент прямоадресованого байту	DEC ad	00010101	3	2	1	(ad) ← (ad) – 1
Декремент байту в РПД	DEC @Ri	0001011i	1	1	1	((Ri)) ← ((Ri)) - 1
Множення акумулятора на регістр В	MUL AB	10100100	1	1	4	(B)(A) ← (A)*(B)
Ділення акумулятора на регістр В	DIV AB	10000100	1	1	4	(B).(A) ← (A)/(B)

4. Група команд логічних операцій

Дану групу утворюють 25 команд, що реалізують логічні операції над байтами (табл. 2). Є можливість виконувати операцію "виключаюче АБО" з вмістом портів. Команда XRL ("виключаюче АБО") може бути ефективно використана для інверсії окремих бітів портів.

Табл. 2. Логічні операції

Назва команди	Мнемокод	КОП	Т	Б	МЦ	Операція
Логічне І акумулятора і регістру	ANL A, Rn	01011rrr	1	1	1	(A) ← (A) AND (Rn)
Логічне І акумулятора і прямоадресованого байту	ANL A, ad	01010101	3	2	1	(A) ← (A) AND (ad)
Логічне І акумулятора і байту з РПД	ANL A, @Ri	0101011i	1	1	1	(A)←(A) AND ((Ri))
Логічне І акумулятора і константи	ANL A, #d	01010100	2	2	1	(A) ← (A) AND #d
Логічне І прямоадресованого байту і акумулятора	ANL ad, A	01010010	3	2	1	(ad) ← (ad) AND (A)
Логічне І прямоадресованого байту і константи	ANL ad, #d	01010011	7	3	2	(ad) ← (ad) AND #d
Логічне АБО акумулятора і регістру	ORL A, Rn	01001rrr	1	1	1	(A) ← (A) OR (Rn)
Логічне АБО акумулятора і прямоадресованого байту	ORL A, ad	01000101	3	2	1	(A) ← (A) OR (ad)
Логічне АБО акумулятора і байту з РПД	ORL A, @Ri	0100011i	1	1	1	(A) ←(A) OR ((Ri))
Логічне АБО акумулятора і константи	ORL A, #d	01000100	2	2	1	(A) ← (A) OR #d
Логічне АБО прямоадресованого байту і акумулятора	ORL ad, A	01000010	3	2	1	(ad) ← (ad) OR (A)
Логічне АБО прямоадресованого байту і константи	ORL ad, #d	01000011	7	3	2	(ad) ← (ad) OR #d
Виключаюче АБО акумулятора і регістру	XRL A, Rn	01101rrr	1	1	1	(A) ←(A) XOR (Rn)
Виключаюче АБО акумулятора і прямоадресованого байту	XRL A, ad	01100101	3	2	1	(A) ← (A) XOR (ad)
Виключаюче АБО акумулятора і байту з РПД	XRL A, @Ri	0110011i	1	1	1	(A) ←(A) XOR ((Ri))
Виключаюче АБО акумулятора і константи	XRL A, #d	01100100	2	2	1	(A) ← (A) XOR #d
Виключаюче АБО прямоадресованого байту і акумулятора	XRL ad, A	01100010	3	2	1	(ad) ← (ad) XOR (A)
Виключаюче АБО прямоадресованого байту і константи	XRL ad, #d	01100011	7	3	2	(ad) ← (ad) XOR #d
Скидання акумулятора	CLR A	11100100	1	1	1	(A) ← 0
Інверсія акумулятора	CPL A	11110100	1	1	1	(A) ← NOT(A)
Зсув акумулятора вліво циклічний	RL A	00100011	1	1	1	$(A_{n+1}) \leftarrow (A_n)$,n=0÷6, $(A_0) \leftarrow (A_7)$
Зсув акумулятора вліво через перенесення	RLC A	00110011	1	1	1	$(A_{n+1}) \leftarrow (A_n)$,n=0÷6 $(A_0) \leftarrow (C)$,$(C) \leftarrow (A_7)$
Зсув акумулятора вправо циклічний	RR A	00000011	1	1	1	$(A_n) \leftarrow (A_{n+1})$,n=0÷6, $(A_7) \leftarrow (A_0)$
Зсув акумулятора вправо через перенесення	RRC A	00010011	1	1	1	$(A_n) \leftarrow (A_{n+1})$, n=0÷6 $(A_7) \leftarrow (C)$,$(C) \leftarrow (A_0)$
Обмін місцями тетрад в акумуляторі	SWAP A	11000100	1	1	1	$(A_{0...3}) \leftrightarrow (A_{4...7})$

5. Група команд операцій з бітами

Відмінною особливістю даної групи команд є те, що вони оперують з однобітними операндами. В якості таких операндів можуть виступати окремі біти деяких регістрів спеціальних функцій (РСФ) і портів, а також 128 програмних флагів користувача.

Існують команди скидання в нуль (CLR), встановлення в одиницю (SETB) і інверсії (CPL) бітів, а також кон'юнкції і диз'юнкції біта і флажка перенесення. Для адресації бітів використовується пряма восьмирозрядна адреса (bit).

6. Реалізація часової затримки

Процедура реалізації часової затримки використовує метод програмних циклів. При цьому в деякий робочий регістр завантажується число, яке потім в кожному проході циклу зменшується на 1. Так триває до тих пір, поки вміст робочого регістру не стане рівним нулю, що інтерпретується програмою як момент виходу з циклу. Час затримки при цьому визначається числом, завантаженим в робочий регістр, і часом виконання команд, що утворюють програмний цикл.

Нижче приведена підпрограма формування часової затримки, що має ім'я DELAY.

Нехай в керуючій програмі необхідно реалізувати тимчасову затримку 25 мкс. Фрагмент програми, що реалізує часову затримку, потрібно оформити у вигляді підпрограми, так як передбачається, що основна керуюча програма буде виробляти до неї багаторазові звернення для формування вихідних імпульсних сигналів, тривалість яких кратна 25 мкс:

```
        ACALL  DELAY
……..
DELAY:
        MOV    R2, #X          ; завантаження кількості циклів
        COUNT:
        PJNZ   R2,COUNT   ; декремент R2 і цикл, якщо не нуль
        RET                    ; повернення з підпрограми.
```

Для отримання необхідної часової затримки необхідно визначити **кількість циклів X**, що завантажується в робочий регістр. Визначення числа X виконується на основі розрахунку часу виконання команд, що утворюють дану підпрограму. При цьому, необхідно враховувати, що команди MOV і RET виконуються одноразово, а число повторень команди DJNZ дорівнює числу X. Крім того, звернення до підпрограми часової затримки здійснюється за командою ACALL DELAY, час виконання якої також необхідно враховувати при підрахунку невеликої затримки.

В таблицях з командасм мікроконтролера вказується, за скільки машинних циклів (МЦ) виконується кожна команда. На підставі цих даних **визначається сумарна кількість машинних циклів в програмі:**

ACALL - 2 МЦ, MOV - 1 МЦ, DJNZ - 2 МЦ, RET - 2 МЦ.

При тактовій частоті 12 МГц кожен машинний цикл виконується за 1 мкс. Таким чином, підпрограма виконується за час $2 + 1 + 2 X + 2 = 5 + 2 X$ (мкс).

Для реалізації часової затримки 25 мкс число **X = (25-5) / 2 = 10.**

В даному випадку, при завантаженні в регістр R2 числа 10 необхідна часова затримка (25 мкс) реалізується точно. Якщо число X виходить дробовим, то часову затримку можна реалізувати лише приблизно. Для більш точного підстроювання в підпрограму можуть бути включені команди NOP, час кожної з яких дорівнює 1 мкс.

Максимальна тривалість затримки, що реалізується підпрограмою DELAY, відповідає X = 255.

Для реалізації затримки більшої тривалості можна рекомендувати збільшити тіло циклу включенням додаткових команд або використовувати метод вкладених циклів.

Приклад програми часової затримки:

Приклад 1:

```
DELAY:
        MOV     Rl,#84              ; завантаження X
LOOPEX:
        MOV     R2,#236             ; завантаження Y
LOOPIN:
        DJNZ    R2, LOOPIN          ; декремент R2 і внутрішній цикл, якщо R2 не рівне нулю
        DJNZ    Rl, LOOPEX          ; декремент R1 і внутрішній цикл, якщо R1 не рівне нулю
LOOPAD:
        MOV     R3,#4               ; точне налаштування часової затримки
        DJNZ    R3, LOOPAD
        RET                         ; повернення з підпрограми.
```

Програма містить два вкладених цикли, а додатковий цикл LOOPAD забезпечує точне налаштування часової затримки.

7. Завдання

1. Вивчити та дослідити арифметичні команди.
2. Вивчити та дослідити логічні команди.
3. Вивчити та дослідити команди зсуву.
4. Написати і дослідити роботу команд з програм (**Завдання 1….Завдання 4**) з використанням програмної моделі відповідно до «**Варіантів індивідуальних завдань**»:

Завдання 1. Додати два двійкові багатобайтні (n-байтні) числа. Доданки розташовуються в резидентній пам'яті даних, починаючи з молодшого байту. Початкові адреси доданків $adr1_{min}$ та $adr2_{min}$ задані в R0 і R1, кількість доданків в байтах (n) - в R2:

```
 adr1 EQU  adr1min     ;присвоєння позначенню adr1 адреси молодшого байту першого доданка
 adr2 EQU adr2min     ;присвоєння позначенню adr1 адреси молодшого байту другого доданка
    MOV R0,#adr1    ;завантаження початкової адреси молодшого байту першого числа в R0
    MOV @R0,#d1     ;завантаження молодшого байту першого доданка в пам'ять
    INC R0          ;інкремент адреси
    MOV @R0,#d2     ;завантаження наступного байту першого доданка в пам'ять
    ……

    INC R0          ;інкремент адреси
    MOV @R0, #dn    ;завантаження останнього байту першого доданка в пам'ять


    MOV R0,#adr1    ;завантаження початкової адреси молодшого байту другого числа в R0
    MOV @R1,#c1     ;завантаження молодшого байту другого доданка в пам'ять
    INC R1          ;інкремент адреси
    MOV @R1,#c2     ;завантаження наступного байту другого доданка в пам'ять
  ……
    INC R1          ;інкремент адреси
    MOV @R1, #cn    ;завантаження останнього байту другого доданка в пам'ять

    MOV R0,#adr1min  ;завантаження адреси молодшого байту першого доданка
    MOV R1,#adr2min  ; завантаження адреси молодшого байту другого доданка
    MOV R2,#n        ;завантаження формату доданків в байтах
    CLR C            ;скидання ознаки перенесення
LOOP:
    MOV A, @R0       ;завантаження в А поточного байту  першого доданка
    ADDC A, @R1        ;додавання байтів з врахуванням флажка перенесення
```

```
    MOV @R0, A          ;розміщення байту результата
     INC R0             ;просування (інкремент) покажчиків
     INC R1
     DJNZ R2, LOOP      ;цикл, якщо не всі байти просумовані
```

Завдання 2. Помножити ціле шістнадцяткове число довільного формату (n-байтів) на константу #d. Початкове число розташовується в резидентній пам'яті даних (адреса $adr1_{min}$), адреса молодшого байту числа знаходиться в регістрі R0. Формат числа в байтах (n) зберігається в R2:

```
 adr1 EQU  adr1min  ;присвоєння позначенню adr1 початкової адреси молодшого байту числа
      MOV R0,#adr1  ;завантаження початкової адреси молодшого байту числа в R0
      MOV @R0,#d1   ;завантаження молодшого байту числа в пам'ять
      INC R0        ;інкремент адреси
      MOV @R0,#d2   ;завантаження наступного байту числа в пам'ять
      ……
      INC R0        ;інкремент адреси
      MOV @R0, #dn  ;завантаження останнього байту числа в пам'ять
      MOV R0,adr1min ;завантаження початкової адреси множеного
      MOV R2,#n     ;завантаження формату числа в байтах
      MOV A,#0      ;скидання акумулятора
LOOP:
      ADD A,@R0     ;завантаження множеного
      MOV B,#d      ;завантаження множника
      MUL AB        ;множення
      MOV @R0,A     ;пересилання молодшого байту добутку в пам'ять
      INC R0        ;інкремент  адреси
      MOV A,B       ;пересилання старшого байту добутку в акумулятор
      XCH A,@R0     ;попереднє формування наступного байту добутку
      DJNZ R2,LOOP  ;цикл, якщо не всі байти початкового числа помножені на константу
```

Завдання 3. Використання команд логічних операцій маскування при вводі даних. Завантажити в регістр Rn (n=0….7) дані з порту Pm і виділити необхідні біти за допомогою маски d, що знаходиться в регістрі Rm:

```
      MOV Rm,#d     ;завантаження в Rm значення маски
      MOV A, Pm     ;ввід байту з ліній порту Pm
      ANL  A,Rm     ;маскування
      MOV Rn,A      ;пересилання вмісту акумулятора в регістр
```

Завдання 4. Перетворення системи числення. Перетворити ціле **двійкове** 8-розрядне число без знаку, що міститься в акумуляторі A (значення в інтервалі 0-255), в тризначне **двійково-десяткове** число, яке займає два байти і розташовується в регістрах R1 (сотні) та R0 (десятки і одиниці) (в регістрах R0 та R1симулятора відображається як десяткове число).

```
      MOV A,#d7d6d5d4d3d2d1d0b   ;завантаження в акумулятор двійкового числа
      MOV B,#100                 ;завантажуємо в регістр B значення дільника
      DIV AB                     ;ділимо на 100 для визначення  числа сотень
      MOV R1,A                   ;зберігаємо число сотень в R1
      MOV A,#10
      XCH A,B                    ;розташовуємо залишок в A
      DIV AB                     ;ділимо на 10 для визначення числа десятків
      SWAP A                     ;число десятків в старшій тетраді A
      ADD A,B                    ;число одиниць в молодшій тетраді A
      MOV R0,A                   ;зберігаємо число десятків і одиниць в R0
      RET
```

Варіанти індивідуальних завдань

№	Зміст індивідуального завдання
1	1. Реалізувати часову затримку, використовуючи програму з Прикладу 1, що дорівнює 25 мкс; 2. Завантажити в симулятор і виконати програму з Завдання 1 для наступних даних: $adr1_{min}$=15H, $adr2_{min}$=25H, n=3, d1=21H, d2=09H, d3=12H, c1=05H, c2=11H, c3=33H.
2	1.Реалізувати часову затримку, використовуючи програму з Прикладу 1, що дорівнює 10 мкс; 2. Завантажити в симулятор і виконати програму з Завдання 2 для наступних даних: $adr1_{min}$ =15H, n=2, d1=18H, d2=12H.
3	1. Реалізувати часову затримку, використовуючи програму з Прикладу 1, що дорівнює 30 мкс; 2. Завантажити в симулятор і виконати програму з Завдання 2 для наступних даних: $adr1_{min}$ =20H, n=2, d1=09H, d2=07H,
4	1.Завантажити в симулятор і виконати програму з Завдання 3 для наступних даних: Rn=R3, Rm=R2, Pm=P2, d=01100101b. 2. Завантажити в симулятор і виконати програму з Завдання 2 для наступних даних: $adr1_{min}$ =21H, n=2, d1=0A1H, d2=1CH.
5	1.Завантажити в симулятор і виконати програму з Завдання 3 для наступних даних: Rn=R4, Rm=R5, Pm=P1, d=10010011b. 2.Завантажити в симулятор і виконати програму з Завдання 1 для наступних даних: $adr1_{min}$=15H, $adr2_{min}$=25H, n=3, d1=21H, d2=09H, d3=12H, c1=05H, c2=11H, c3=33H.
6	1. Завантажити в симулятор і виконати програму з Завдання 4 для наступних даних: числа для завантаження в акумулятор – 22H; 0F15H. 2.Завантажити в симулятор і виконати програму з Завдання 2 для наступних даних: $adr1_{min}$ =17H, n=2, d1=0A3H, d2=36H.
7	1. Завантажити в симулятор і виконати програму з Завдання 4 для наступних даних: числа для завантаження в акумулятор - 01011001b; 0A5H. 2.Завантажити в симулятор і виконати програму з Завдання 1 для наступних даних: $adr1_{min}$=18H, $adr2_{min}$=29H, n=3, d1=12H, d2=19H, d3=31H, c1=15H, c2=16H, c3=21H.
8	1.Реалізувати часову затримку, використовуючи програму з Прикладу 1, що дорівнює 15 мкс; Завантажити в симулятор і виконати програму з Завдання 1 для наступних даних: $adr1_{min}$=20H, $adr2_{min}$=31H, n=3, d1=19H, d2=35H, d3=0FH, c1=1AH, c2=0A5H, c3=0B3H.
9	1. Завантажити в симулятор і виконати програму з Завдання 4 для наступних даних: числа для завантаження в акумулятор – 22H; 15H. 2. Завантажити в симулятор і виконати програму з Завдання 1 для наступних даних: $adr1_{min}$=17H, $adr2_{min}$=28H, n=2, d1=0A2H, d2=C9H, c1=0B5H, c2=1DH.
10	1.Завантажити в симулятор і виконати програму з Завдання 3 для наступних даних: Rn=R3, Rm=R7, Pm=P2, d=01110101b. 2. Завантажити в симулятор і виконати програму з Завдання 1 для наступних даних: $adr1_{min}$=16H, $adr2_{min}$=29H, n=3, d1=0A1H, d2=19H, d3=2CH, c1=0B5H, c2=31H, c3=0C6H.
11	1. Завантажити в симулятор і виконати програму з Завдання 4 для наступних даних: числа для завантаження в акумулятор – 45H; 31H. 2. Завантажити в симулятор і виконати програму з Завдання 1 для наступних даних: $adr1_{min}$=17H, $adr2_{min}$=28H, n=2, d1=0A2H, d2=C9H, c1=0B5H, c2=1DH.
12	1. Завантажити в симулятор і виконати програму з Завдання 4 для наступних даних: числа для завантаження в акумулятор - 11011011b; 65H. 2. Завантажити в симулятор і виконати програму з Завдання 1 для наступних даних: $adr1_{min}$=23H, $adr2_{min}$=36H, n=3, d1=0A2H, d2=14H, d3=11H, c1=0B5H, c2=1FH, c3=41H.

8. Записати в звіт зміни в вікнах регістрів мікроконтролера при виконанні програми завдання вибраного варіанту **для десяти команд** відповідно до табл. 2.

Таблиця 2 - Результати виконання команд

№	Команда	Виконувана операція	Вміст використовуваних регістрів та комірок пам'яті до і після виконання		Пояснення
			До	**Після**	
1	MOV A,R0	Пересилання байту даних з регістру R0 в акумулятор A	A/00 PC/00 PC/01	A/F2 PSW/00 PSW/01	
2	…	…	…	…	…
…	…	…	…	…	…
5	…	…	…	…	…

9. Додати у звіт копію екрану з виконаною програмою на програмному симуляторі відповідно до обраного варіанту.

***Примітка**

1. Якщо ви хочете виконати якусь з команд пересилання, наприклад, з регістра в регістр, необхідно в регістр, з якого буде здійснене пересилання, командою MOV попередньо записати якесь значення операнду (адресу чи константу).

2. Програма, що виконується, буде записана в пам'ять програм, вміст якої можна побачити, натиснувши на кнопку ***"Data memory"*** в нижній частині **"Панелі пам'яті даних та програмної пам'яті"**, що знаходиться зліва від **"Панелі коду Асемблера"**. Після натискання кнопка ***"Data memory"*** зміниться на кнопку ***"Code memory"***, тобто буде висвічуватися в полі пам'яті вміст пам'яті програм.

10. Послідовність виконання роботи

10.1. Вивчити команди пересилання. Вивчення кожної команди проводити наступним чином:

10.1.1. Відкрити інтерфейс симулятора, двічі клацнувши клавішею миші на архівованому файлі «EdSim51.jar". Відкриється інтерфейс програмного симулятора, зображений на рис.1.

Середнє поле емулятора, що називається **"Панель коду Асемблера"**, в верхній частині містить кнопки **"*Reset*", "*Assm*", "*Run*", "*Load*", "*Save*", "*Copy*", "*Past*"**.

Панель коду використовується для:

- **набору команд** програми з клавіатури. Для цього курсор встановлюється в верхній частині панелі і вводиться програма по одній команді в рядку (при потребі, з міткою та коментарем)(див. рис.1);
- **завантаження** вже існуючої програми. Для цього необхідно на панелі вгорі натиснути кнопку **"*Load*"** і вказати шлях до потрібного файлу;
- **запису** набраного файлу. Для цього потрібно натиснути кнопку **"*Save*"** і вказати шлях для збереження файлу.

10.1.2. Перед виконанням програми необхідно натиснути кнопку **"*Assm*"** панелі для асемблювання програми. Після цього, якщо команда записана невірно, в рядку під верхнім рядом кнопок панелі (на рис.1 виділений сірим кольором) з'явиться повідомлення про помилку, а **колір рядка зміниться на червоний**. Червоним кольором буде виділена також невірно написана команда.

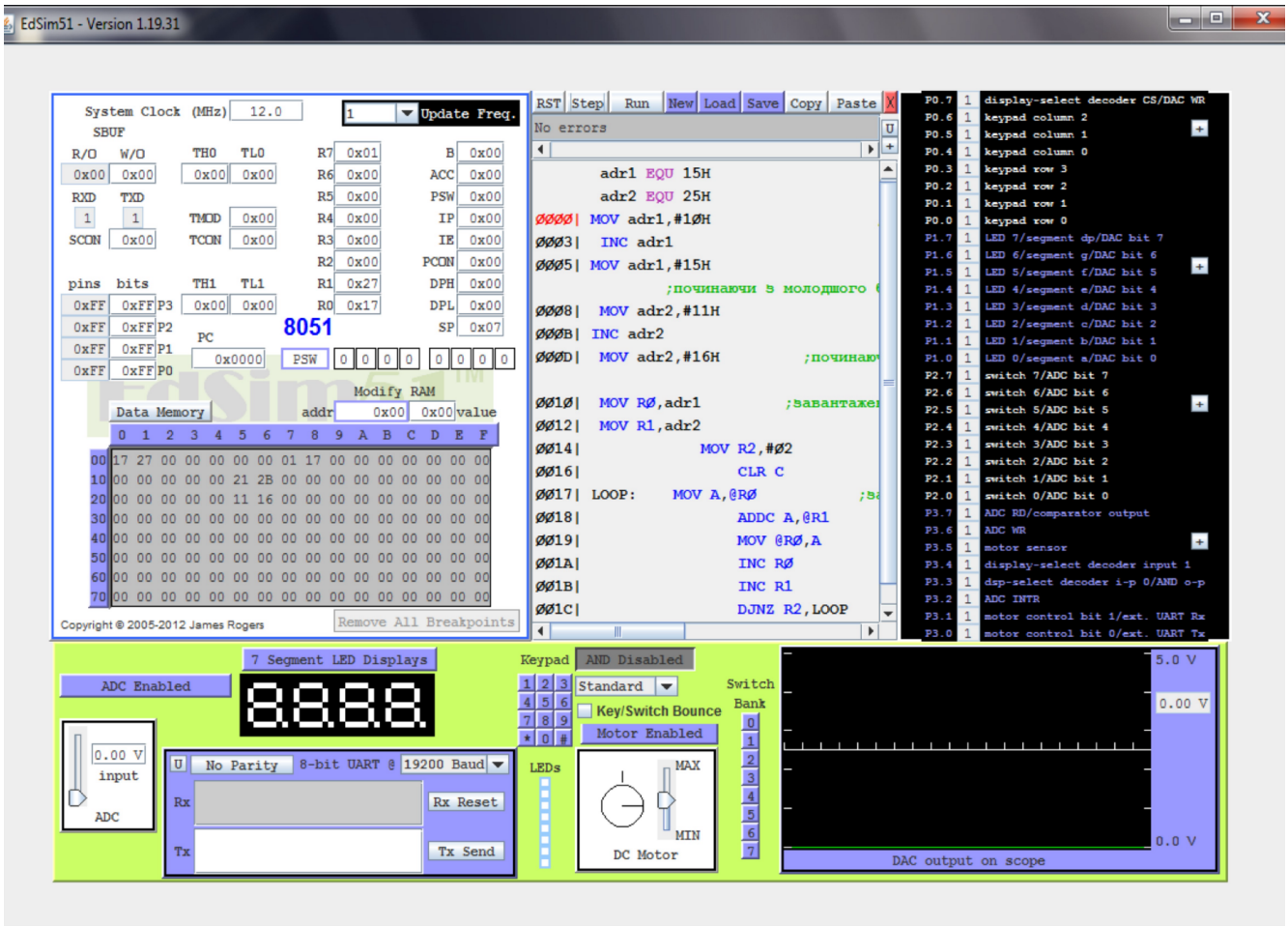

Рис. 1. Інтерфейс програмного симулятора

Якщо помилки відсутні, зліва від команд набраної програми з'являться адреси, і сама програма буде готова до виконання. Після асемблювання кнопка **"*Assm*"** зміниться на кнопку **"*Step*"**. Таким чином, є можливим виконувати програму покомандно **в кроковому режимі**, натискаючи кнопку **"*Step*"**після виконання кожної команди, або **в автоматичному режимі**, коли виконується вся програма, натиснувши один раз кнопку **"*Run*"**. В останньому випадку програму слід закінчувати директивою ***"End".***

При написанні програми можна користуватися для копіювання її фрагментів та вставки в будь-якому місці "Панелі коду Асемблера" кнопками **"*Copy*"** *та* **"*Past*"**.

Щоб зупинити виконання програми і скинути в початковий стан регістри мікроконтролера симулятора необхідно натиснути кнопку **"*Reset*".**

11. Контрольні запитання

1. Які команди входять в групу арифметичних і логічних команд?
2. Які команди входять в групу команд передачі управління?
3. Пояснити результати виконання програм додавання, множення, ділення і зсуву. Пояснити стан флажків регістра PSW.
4. Як здійснюється розрахунок часових затримок?
5. Пояснити роботу програми з невеликою затримкою.

Рекомендована література

1. Проектування мікропроцесорних систем керування : навчальний посібник, перевидання / Медвідь В.Р., Пісьціо В.П., Козбур І.Р. – Тернопіль : Вид-во ТНТУ імені Івана Пулюя, 2015. – 360 с.

2. Handbook of Microcontrollers/Predko Michael. NYc. McGraw-Hill. 1998. 861 p.

3. Бойко В. І., Гуржій А. М., Жуйков В. Я. та ін.Схемотехніка электронних схем: У 3 кн. Кн.3 Мікропроцесори та мікроконтролери: підручник. 2-ге вид., допов. і переробл. К.: Вища шк., 2004. 399 с.

4 Мілих В. І., Шавьолкін О. О. Електротехніка, електроніка та мікропроцесорна техніка: підручник; за ред. В. І. Мілих. 2-е вид. К.: Каравела, 2008. 688 с.