НАВЧАЛЬНО-МЕТОДИЧНА ЛІТЕРАТУРА

Тернопільський національний технічний університет імені Івана Пулюя

Кафедра автоматизації технологічних процесів і виробництв

Електроніка і мікропроцесорна техніка

Методичні вказівки для лабораторної роботи № 29

Програмування МП і8086 з використанням програмного емулятора “emu8086”

Тернопіль 2016

Методичні вказівки для лабораторної роботи №29. "Програмування МП і8086 з використанням програмного емулятора emu8086" з курсу "Електроніка і мікропроцесорна техніка". /Медвідь В.Р., Микулик П.М., Пісьціо В.П., Тернопіль: ТНТУ, 2016 - 15 с.

Для студентів напряму: 6.050202 "Автоматизоване управління технологічними процесами.

Методічні вказівки розглянуті і затверджені на засіданні кафедри автоматизації технологічних процесів і виробництв (протокол № 6 від 23.11.2015 року).

Лабораторна робота №29
ПРОГРАМУВАННЯ МП i8086 З ВИКОРИСТАННЯМ ПРОГРАМНОГО ЕМУЛЯТОРА “emu8086”

1. Структура мікропроцесора

Мікропроцесор i8086 має розрядність шини даних 16 біт, шини адреси – 20 біт. Схема мікропроцесора представлена на рис.1:

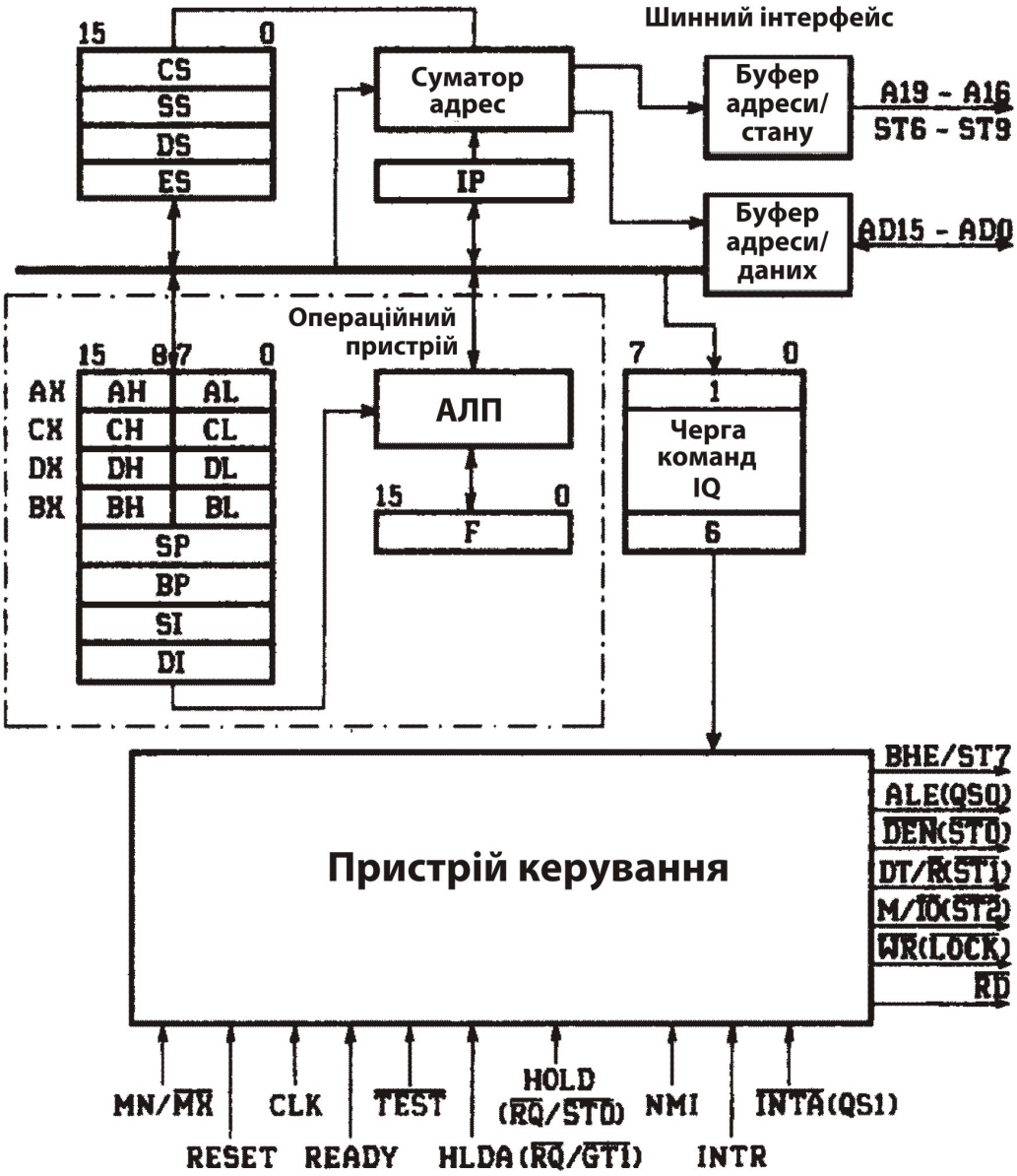

Рис.1

У мікропроцесорі є вісім 16-розрядних **регістрів загального призначення**:
AX – акумулятор,
BX – базовий регістр,
CX – лічильник,
DX – регістр даних,
SP – покажчик стеку,
BP – покажчик бази,
SI - індекс джерела,
DI – індекс приймача.

В основному, регістри **AX**, **BX, CX** і **DX** використовуються для зберігання даних, регістри **SP**, **BP, SI, DI** – для адресації.

Для регістрів **AX, BX, CX, DX** можливо також окреме використання молодших і старших байтів (**AH, AL, BH, BL, CH, CL, DH, DL**).

Регістр **SP** (покажчик стеку) містить поточне значення вершини стеку, регістр **BP** (Base Pointer) використовується при деяких спеціальних формах адресації даних. Регістри **SI** і **DI** застосовуються при роботі з так званими рядками – послідовностями байтів або слів.

Регістр **SI** (індекс джерела) вказує на поточний оброблюваний елемент початкового рядка (рядка-джерела), а **DI** (індекс призначення) – на елемент результуючого рядка (рядка-приймача).

Регістр флагів **F**:

FH								FL							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x	x	x	x	OF	DF	IF	TF	SF	ZF	x	AF	x	PF	x	CF

Арифметичні флаги:

CF – флаг перенесення, фіксує значення перенесення з старшого біту результату при додаванні, або позичання при відніманні, а також значення висунутого біту при зсуві операнду;

PF – флаг паритету, фіксує парне число одиниць в молодшому байті результату;

AF – флаг допоміжного перенесення, фіксує перенесення з молодшої тетради (використовується при виконанні десяткового корекції);

ZF – флаг нульового результату;

SF – флаг знаку, дорівнює старшому розряду результату при роботі зі знаковою арифметикою;

OF – флаг переповнення.

Флаги управління:

DF – флаг напрямку обробки ланцюжків команд: DF=0 – в сторону великих адрес, DF=1 – у бік менших.

IF – флаг дозволу переривань: IF = 0 переривання забороняються, IF = 0 – дозволяються.

TF – флаг трасування: TF = 1 – кроковий режим роботи.

Блок сегментних регістрів складається з чотирьох 16-розрядних регістрів **CS, SS, DS, ES**, які зберігають базові адреси сегментів пам'яті: **CS** – сегмент команд, **SS** – сегмент стеку, **DS** – сегмент даних, **ES** – додатковий сегмент (також використовується для зберігання даних).

Покажчик команд **IP** зберігає зміщення наступної команди в поточному кодовому сегменті.

Організація пам'яті в мікропроцесорі

Пам'ять являє собою набір послідовно розташованих байтів, кожен з яких має 20-розрядну адресу.

Два суміжних байти можуть розглядатися як 16-бітове слово, адресою слова вважається адреса молодшого байту (вона менша, ніж адреса старшого байту).

Адресний простір має ємність 1 Мбайт і розбитий на сегменти ємністю 64 Кбайт, які характеризуються базами.

Початкові адреси чотирьох поточних сегментів записуються в сегментні регістри CS, SS, DS, ES; для переходу до іншої області пам'яті досить змінити вміст сегментного регістра.

Кожна комірка пам'яті характеризується **логічною і фізичною адресою**.

Фізична адреса – це 20-розрядне число, що однозначно визначає положення комірки в пам'яті.

Логічна адреса комірки складається з двох 16-бітових значень: **бази сегменту і зміщення** всередині сегменту відносно бази (визначає відстань від початку сегмента до цієї комірки).

Для перерахунку логічної адреси в фізичну база сегменту зсувається вліво на 4 біти і сумується зі зміщенням.

Якщо при додаванні виникає біт перенесення, він ігнорується; таким чином, після комірки пам'яті з адресою FFFFF йде комірка з адресою 00000, тобто виникає кільцева організація пам'яті (вона також притаманна кожному сегменту окремо).

Звернення до пам'яті.

До кожного сегменту пам'яті звернення здійснюється окремо.

Пряма передача інформації між сегментами неможлива, для цього необхідно використовувати регістри загального призначення.

Для звернення до пам'яті використовуються 4 регістри: **BX, SI, DI, BP**, а також зміщення, що задається безпосередньо величиною: d8 – 8 біт або d16 – 16 біт. Комбінуючи ці регістри і зміщення всередині квадратних дужок **[]**, можна задіяти будь-яку комірку пам'яті.

r/m	md=00	md=01	md=10
000 001 010 011	[BX + SI] [BX + DI] [BP + SI] [BP + DI]	[BX + SI + d8] [BX + DI + d8] [BP + SI + d8] [BP + DI + d8]	[BX + SI + d16] [BX + DI + d16] [BP + SI + d16] [BP + DI + d16]
100 101 110 111	[SI] [DI] d16 [BX]	[SI + d8] [DI + d8] [BP + d8] [BX + d8]	[SI + d16] [DI + d16] [BP + d16] [BX + d16]

Для прикладу візьмемо DS = 100, BX = 30, SI = 70.

У цьому випадку адреса [BX + SI] +25 буде перетворена мікропроцесором в наступну фізичну адресу: 100* 16 + 30 + 70 + 25 = 1725.

За замовчуванням для адресації даних завжди використовується сегментний регістр DS, крім режимів з регістром BP, коли в цьому випадку використовується сегментний регістр SS. Адреси задаються в шістнадцятковому коді.

2. Структура команд мікропроцесора

Двооперандні команди:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Код операції						d	s	md		reg			r/m		
disp L								disp H							

Біт **d** вказує напрямок передачі: **d = 1** - передача операнду або результату операції в регістр, вказаний в полі reg, **d = 0** - передеча з регістру reg.

Біт **S** показує формат даних: **s** = 1 - слово, **s** = 0 - байт

Поле **reg** визначає другий операнд, який обов'язково знаходиться в регістрі; поле **r/м** (регістр/ пам'ять) визначає перший операнд, який може знаходитися в регістрі або пам'яті.

reg, r/m	s=0	s=1	reg, r/m	s=0	s=1
000	AL	AX	100	AH	SP

001	CL	CX	101	CH	BP
010	DL	DX	110	DH	SI
011	BL	BX	111	BH	DI

Поле **md** показує, де міститься перший операнд, в пам'яті або в регістрі, а в разі розташування операнду в пам'яті визначає варіант використання зміщення disp, яке може перебувати в третьому і четвертому байтах команди:

md = 00 - операнд міститься в пам'яті, disp = 0 (зміщення відсутнє)

md = 01 - операнд міститься в пам'яті, disp = disp L (команда містить 8-бітове зміщення, яке розширюється зі знаком до 16 біт)

md = 10 - операнд міститься в пам'яті, disp = disp H, disp L (команда містить 16-бітове зміщення)

md = 11 - операнд міститься в регістрі.

Однооперандні команди:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Код операції							s	md		Код операції			r/m		
disp L								disp H							

Можуть містити від 2-х до 4-х байт в залежності від значень поля **s** та **md**

Команди з безпосередньою адресацією:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Код операції						s	w	md		Код операції			r/m		
disp L								disp H							
data L								data H							

Може містити до 3-х слів.

Поля **s** і **w** показують, як використовуються останні 2 байти команди:

s	w	
x	0	data=data L (8 разрядів)
0	1	data=data H, dataL (16 разрядів)
1	1	data=data L (розширення до 16-розрядного слова із знаком)

3. Система команд мікропроцессора

Команди пересилання даних

MOV op1, op2 - переслати операнд op2 в op1
PUSH op - записати операнд op в стек
POP op - витягти операнд op із стеку
XCHG op1, op2 - поміняти місцями значення операндів op1 і op2

Основні арифметичні команди

ADD op1, op2 - виконати додавання виду op1 = op1 + op2
INC op - збільшити вміст операнду op на одиницю (op = op + 1)
SUB op1, op2 - виконати віднімання виду op1 = op1 - op2
DEC op - зменшити вміст операнду op на одиницю (op = op - 1)
CMP op1, op2 - виконати порівняння операндів op1 і op2 (тобто, обчислити різницю (op1 - op2) і встановити флаги в регістрі F)
MUL op (op - байт) - виконати множення виду AX = AL * op

MUL op (op - слово) - виконати множення виду DXAX = AX * op
DIV op (op - байт) - ділення AX / op; частка - в AL, залишок - в AH
DIV op (op - слово) - ділення DXAX / op; частка - в AX, залишок - в DX

Всі арифметичні команди впливають на вміст регістру флагів F, проте, після команд множення і ділення стани флагів не визначені (довільні).

Команди зсуву даних:

SHL op - зсув операнду op (беззнаковое ціле) на 1 розряд вліво
SHR op - зсув операнду op (беззнаковое ціле) на 1 розряд вправо

Команди управління ходом програми:

LOOP label - організація циклу з лічильником в регістрі CX. Команда здійснює віднімання одиниці з цього регістру, і, якщо CX не дорівнює нулю, виконується перехід на мітку label. Ця мітка розміщується «вище» за програмою, тобто до команди LOOP.
JZ label - перехід на мітку label, якщо «нуль», тобто якщо ZF = 1
JNZ label - перехід на мітку label, якщо «Не нуль», тобто якщо ZF = 0
Обидві ці команди зазвичай вживаються в програмі після команд CMP, SUB або DEC.

4. Способи адресації

Регістрова адресація

Операнди можуть розташовуватися в будь-яких регістрах загального призначення і сегментних регістрах. У цьому випадку в тексті програми вказується назва відповідного регістру, наприклад **команда, яка копіює в регістр AX вміст регістра BX**, записується як

mov ax, bx

Безпосередня адресація

Деякі команди (всі арифметичні команди, крім поділу) дозволяють вказувати один з операндів безпосередньо в тексті програми, наприклад, команда

mov ax, 2

поміщає в регістр AX десяткове число 2.

Пряма адресація

Якщо відома адреса операнда, розташованого в пам'яті, можна використовувати цю адресу. Якщо операнд - слово, яке перебуває в сегменті, на який вказує ES, зі зміщенням від початку сегмента 0001, то команда

mov ax, es: [0001h]

помістить це слово в регістр AX.

Якщо селектор сегмента даних знаходиться в DS, ім'я сегментного регістра при прямій адресації можна не вказувати, DS використовується за умовчанням. Пряма адресація іноді називається адресацією по зсуву.

Непряма адресація

За аналогією з регістровими і безпосередніми операндами адреса операнду в пам'яті також можна не вказувати безпосередньо, а зберігати в будь-якому регістрі. До 80386 для цього можна було використовувати тільки BX, SI, DI і BP.

Наприклад, наступна **команда поміщає в регістр AX слово з комірки пам'яті, базова адреса сегменту якої знаходиться в DS, а зміщення - в BX:**

mov ax, [bx]

Як і в разі прямої адресації, DS завжди використовується за умовчанням.

Але **в реальних програмах, якщо зміщення беруть з регістру BP, то в якості сегментного регістра використовується SS.**

Адресація по базі із зміщенням

Тепер скомбінуємо два попередніх методи адресації: **команда**

mov ax, [bx] +2

поміщає в регістр AX слово, яке перебуває в сегменті, зазначеному в DS, зі зміщенням на 2 більшим, ніж число, що знаходиться в BX.

Так як слово займає рівно два байти, ця команда помістила в AX слово, яке безпосередньо слідує за тим, що є в попередньому прикладі. Така форма адресації використовується в тих випадках, коли в регістрі знаходиться адреса початку структури даних, а доступ треба здійснити до якого-небудь елементу цієї структури.

Інше важливе застосування адресації по базі із зсувом - доступ з підпрограми до параметрів, що подані в стекуі, використовуючи регістр BP в якості бази і номер параметру в якості зсуву.

До 80386 в якості базового регістру можна було використовувати тільки BX, BP, SI або DI і зміщення могло бути тільки байтом або словом (зі знаком).

За допомогою цього методу можна організовувати доступ до одновимірних масивів байтів: зміщення відповідає адресі початку масиву, а число в регістрі - індексу того елементу масиву, який треба обробити.

Адресація по базі з індексуванням

У цьому методі адресації зміщення операнда в пам'яті обчислюється як сума чисел, що містяться в двох регістрах, і зміщення, якщо воно зазначено:

mov ax, [bx + si] +2

у регістр AX поміщається слово з комірки пам'яті зі зміщенням, що дорівнює сумі чисел, що містяться в BX і SI, і числа 2.

З шістнадцятибітних регістрів так можна додавати тільки **BX + SI, BX + DI, BP + SI і BP + DI.** Так можна прочитати, наприклад, число з двовимірного масиву: якщо задана таблиця 10x10 байт, 2 - зміщення її початку від початку сегменту даних, BX = 20, а SI = 7, наведена команда прочитає слово, що складається з сьомого і восьмого байтів третього рядка.

5. Порядок виконання дій при роботі з емулятором «emu8086»

Для виконання роботи використовується програмний емулятор мікропроцесора emu8086. Вікно програми має слідуюиий вигляд (рис. 2):

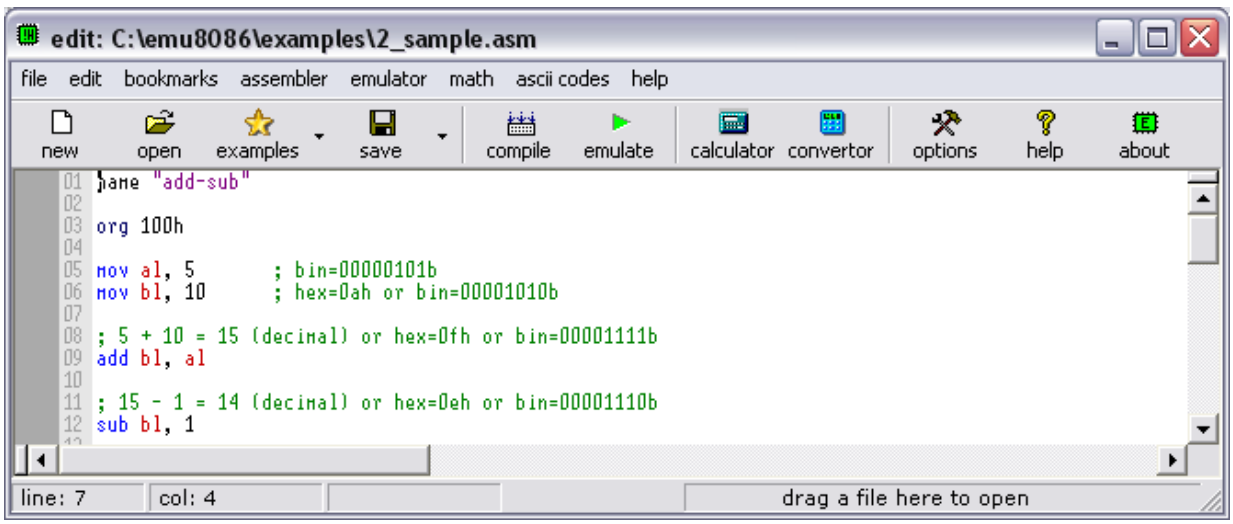

Рис. 2

Текст програми набирається в полі редактора порядково.

Коментарі записуються після знаку ";".

Числа допускається записувати в будь-якій системі числення, при цьому після числа ставиться позначення: **h - шістнадцяткове, b - бінарне, o - вісімкове, наприклад**: 1ah, 101b, 71o.

Числа без позначення вважаються десятковими.

Якщо в записі шістнадцяткового числа старшим розрядом є літера (A..F), то перед нею необхідно поставити «0».

Для створення циклів, умовних і безумовних переходів використовуються команди LOOP, JMP, JA, JC і т.п. і мітки виду **label:**

В кінці програми рекомендується ставити команду **RET**.

Повний список і опис усіх команд процесора можна знайти в меню help: Documentation and tutorials / 8086 Instruction Set.

Для запуску набраної програми слід натиснути на кнопку **emulate** на панелі інструментів. При цьому відкривається вікно емулятора (рис.3):

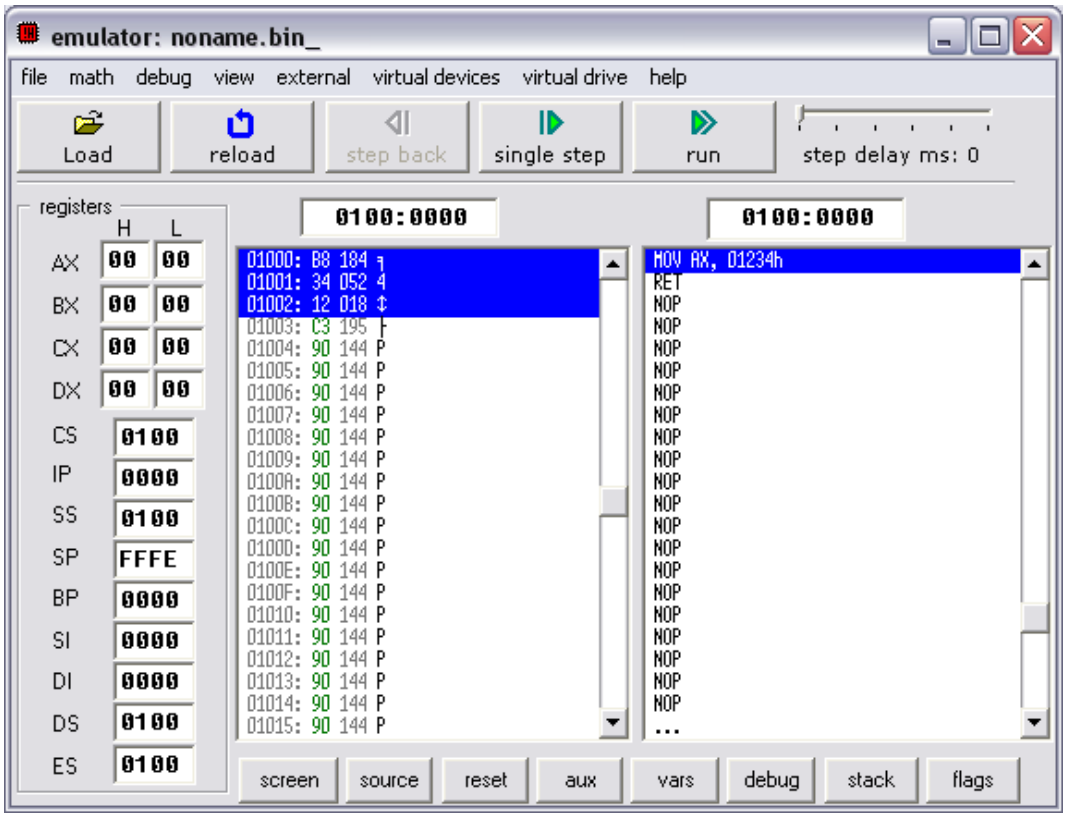

Рис.3

У вікні емулятора можна управляти режимом роботи програми: кнопка

single step дозволяє використовувати покроковий режим,

run - запустити програму на виконання.

У лівій частині вікна виводяться значення всіх регістрів загального призначення, їх можна переглядати і змінювати по ходу виконання програми; крім того, подвійне клацання миші по віконцях зі значенням регістра дозволяє вивести на екран вікно розширеного перегляду значень регістрів в різних кодуваннях.

У центральній частині вікна емулятора знаходяться номери і вміст комірок пам'яті (підсвічуються комірки, що відповідаютьі наступному рядку програми), в правій частині - оброблений текст виконуваної програми.

Вміст регістру флагів можна переглядати і змінювати за допомогою кнопки **flags** в лівому нижньому кутку.

За допомогою регулятора **step delay** можна встановлювати затримку між виконанням кроків програми.

Завдання №1

І.Ознайомтесь з работою емулятора:

1) запустите программу emu8086.exe.

В вікні привітання виберіть варіант **new**, далі виберіть **empty workspace**.

2) для прикладу напишіть п**рограму занесения числа 1234h в регістр AX**:

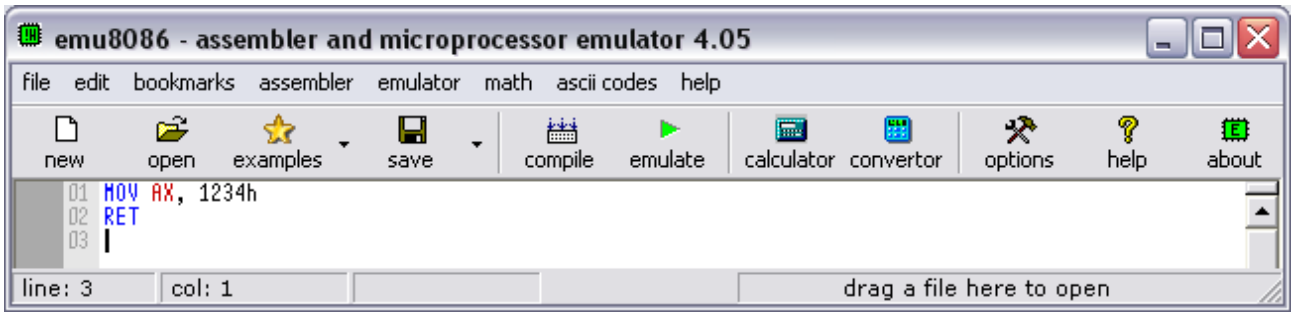

Рис.6

3) запустіть її натисканням **клавіші F5** або кнопки **emulate** на панелі інструментів, відкриються вікно емулятора (рис.3) і коду програми (рис.7):

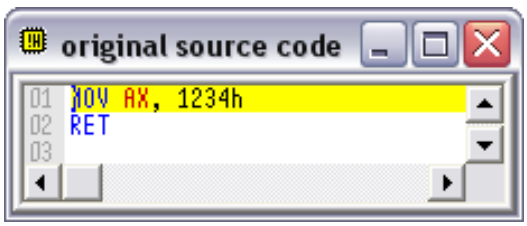

Рис.7

4) запустіть програму на виконання в **кроковому режимі** і простежте за зміною значення регістрів AX і IP (**вміст змінених при виконанні команди регістрів виділяється синім кольором**), потім перезавантажте програму натисканням кнопки **reload** і виконайте її за допомогою кнопки **run.**

5) перейдіть до вмісту регістру флагів, натиснувши кнопку **flags**; змініть значення флагу паритету PF на 1.

ІІ. Напишіть на самостійній підготовці відповідно до Вашого варіанту програми, що виконують ті чи інші дії, і виконайте їх в лабораторії:

ВАРІАНТ 1

Команда завантаження і пересилання даних (команда MOV)

1) завантажити регістр CX числом ABCDh,

2) переслати вміст CX в регістр AX,

3) завантажити комірку пам'яті поточного сегменту без зміщення (зміщення 0) числом FFh,

4) задати в якості бази сегменту DS = 200h, попередньо записавши це число в регістр DX; завантажити комірку 02100h числом AAh (зміщення 100h),

5) записати в регістр BX зміщення 200h і завантажити комірку 02200h числом BBh (зміщення [BX]),

6) записати в регістр SI індекс 20 і завантажити комірку 02220h числом CCh, використовуючи сегмент з базою DS = 200h і зміщення [BX + SI],

7) завантажити комірку 02222h числом DDh, використовуючи сегмент з базою DS = 200h і зміщення [BX + SI + 2].

Одноопернадні команди

1) завантажити регістр DX числом 5,
2) збільшити вміст регістру DX на одиницю (команда INC),
3) перенести вміст регістра DX в комірку пам'яті 01100h,
4) зменшити вміст комірки 01100h на одиницю (команда DEC),
5) інвертувати вміст регістру DX (команда NOT).

ВАРІАНТ 2

Двоопернадні команди

а) Арифметичні операції:

1) додати числа 1Bh і 1Ch, використовуючи регістри BX і CX, результат записати в CX (команда ADD),
2) відняти від вмісту регістра CX константу 10b, результат записати в AX (команда SUB),
3) вміст регістру AX розділити на 10h, попередньо записавши дільник в регістр BX (команда DIV)
4) залишок від ділення (знаходиться в регістрі DX) помножити на 2 (команда MUL).

б) Логічні операції:

1) записати в комірку пам'яті 01100h число 101101b, в регістр AX число 110001b, виконати для них логічне множення, результат записати в комірку 01100h (команда AND),
2) записати в регістр AX число 111010b, виконати логічне додавання між вмістом AX і комірки пам'яті 01100h, результат записати в регістр AX (команда OR),
3) виконати операцію “сума по модулю 2” для вмісту регістра AX і числа 001001b (команда XOR).

ВАРІАНТ 3

Команди зсуву

1) занести в регістр BX число 25h, помножити його на 2, використовуючи операцію арифметичного зсуву (команда SAL),
2) занести в комірку пам'яті 01200h число 40h, розділити його на 4, використовуючи операцію арифметичного зсуву (команда SAR).

Команди управління програмою

а) Цикл (команда LOOP)

1) занести в регістр CX число 10 (кількість повторень циклу),
2) встановити в якості бази сегменту DS = 0100h, занести в регістр BX зміщення 0101h,
3) встановити мітку початку виконання циклу виду label:
4) тіло циклу: занести число AAh в комірку пам'яті, зміщення якої щодо бази DS зазначено в регістрі BX, і збільшити вміст BX на 1,
5) після запису тіла циклу перейти по мітці label.

Після виконання програми число AAh має бути записано в комірку з 01101h по 0110Ah.

ВАРІАНТ 4

Команди управління програмою

б) Умовний перехід (команда JC)

1) занести число FFh в регістр AL,
2) збільшити вміст регістру AL на AAh,
3) якщо біт перенесення не дорівнює одиниці, записати в регістр BX число ABCDh, інакше завершити програму (перехід по мітці до кінця програми).

в) Безумовний перехід (команда JMP)

1) занести в ккомірку 01100h число C3h (код команди RET),

2) написати програму додавання чисел Ah і Bh, використовуючи регістри DL і DH,
3) для закінчення виконання програми перейти за адресою 01100h.

Завдання №2

1. Виконується на самостійній підготовці

Ознайомитися з роботою программного емулятора emu8086 мікропроцесора i8086, користуючись методичними вказівками «Методичні вказівки по роботі з програмним емулятором emu8086».

2. Виконуєтсья в лабораторіі

Виконайте в лабораторії відповідно до Вашого варіанту програми, що виконують ті чи інші дії:

Приклад виконання програми **встановлення режиму дисплею** за допомогою команди ВВІД/ВИВІД ПО ПРЕРИВАННЮ INT 10H **(AH=0)**:

В AL вказується номер режиму
AL=0: Текстовий чорно-білий 40x25
AL=1: Текстовий кольоровий 40x25
AL=2: Текстовый чорно-білий 80x25
AL=3: Текстовий кольоровий 80x25
AL=4: Графічний кольоровий 320x200

Наприклад, для встановлення кольорового графічного режиму необхідно:

```
mov ah,0
mov al,4
int 10h
```

ВАРІАНТ 1

Завдання 1

Записати та виконати команди в кроковому режимі роботи :

```
org 100h      ;задається адреса наступної команди

              ; встановити режим відео
mov ax, 3     ; задаються параметри  дисплею 80x25, 16 colors, 8 pages (ah=0, al=3)
int 10h       ; виконується команда переривання
```

Записати вміст регістрів МП, які змінюються в результаті виконання команд, в табл.1.

Таблиця 1

Регістр	IP	SP	AH	AL	BH	BL	CH	CL	DH	DL	CS	DS	SS	ES	BP	SI	DI	F
Команда 1																		
Команда 2																		
Команда 3																		
Команда n																		

Завдання 2

Записати та виконати команди в кроковому режимі роботи:

```
org 100h
                ; встановити значення сегментного регістру DS
mov    ax, 0b800h
mov    ds, ax
```

Записати вміст регістрів МП, що змінюються в результаті виконання команд (табл.1).

Завдання 3

Записати та виконати команди додавання та віднімання в кроковому режимі роботи:

```
org 100h
mov al, 5       ; bin=00000101b
mov bl, 10      ; hex=0ah or bin=00001010b
                ; 5 + 10 = 15 (decimal) or hex=0fh or bin=00001111b
add bl, al
                ; 15 - 1 = 14 (decimal) or hex=0eh or bin=00001110b
sub bl, 1
```

Записати вміст регістрів МП, що змінюються в результаті виконання команд (табл.1).

ВАРІАНТ 2

Завдання 1

Записати та виконати підпрограму, що використовує команду **TEST** для перевірки значення потрібного біту з регістра, який множиться на 1 (всі решта бітів - на нуль). Виконує операцію логічного множення двох операндів. Не змінює значення регістру.

```
mov bl, 8
    test bl, 10000000b
    jz zero
    mov bl, 1
zero:  int 21h
```

Записати вміст регістрів МП, що змінюються в результаті виконання команд (табл.1).

Завдання 2

Записати та виконати команди, які оперують з числами, записаними в різних системах числення.

```
org 100h
                        ; завантаження двійкового числа:
                        ; (hex: 5h)
mov al, 00000101b
                        ; завантаження шістнадцяткового числа
mov bl, 0ah
                        ; завантаження вісімкового числа
                        ; (hex: 8h)
mov cl, 10o
                        ; 5 + 10 = 15 (0fh)
add al, bl ; додавання двох чисел, результат записується в перший регістр
                        ; 15 - 8 = 7
sub al, cl              ; віднімання друго числа від першого, результат записується в
                        ;перший регістр.
```

Записати вміст регістрів МП, що змінюються в результаті виконання команд (табл.1).

Завдання 3

Записати та виконати арифметичні та логічні команди.

```
org   100h
mov    ah, 09h
mov    al, 05h
                          ; al = al + ah =
                          ; = 09h + 05h = 0eh
add    al, ah
xor    ah, ah
```

Записати вміст регістрів МП, що змінюються в результаті виконання команд (табл.1).

ВАРІАНТ 3

Завдання 1

Записати та виконати команди програмування портів з використанням регістра DX для їх адресації. Для цього відкрити опцію «virtual devices» вікна програмного емулятора і вибрати закладку «simple io test».

```
org  0100h
     mov dx,112          ;задає адресу порта (112 число десяткове)
     mov al,099h         ;дані для виводу
     out dx,al           ;вивід даних в порт
L1:                      ;початок циклу
     mov dl,110          ;задає адресу порта (110 число десяткове)
     mov al,02Fh         ;дані для виводу
     out dx,al           ;вивід даних в порт
     jmp L1

     mov dx,112          ;задає адресу порта (112 число десяткове)
     mov al,23h          ;дані для вводу (записати в вікно порту 112)
     in al,dx            ;ввід даних  з порту 112
hlt
```

Завдання 2

Записати та виконати команди виводу даних на світлодіодний дисплей (адреса дисплею199, десяткове). Для цього відкрити опцію «virtual devices» вікна програмного емулятора і вибрати закладку «LED Display».

```
mov ax, 1234
out 199, ax              ; вивід числа 1234 на дисплей

mov ax, -5678
out 199, ax              ; вивід числа -5678 на дисплей
hlt
```

Завдання 3

Записати та виконати команди виводу даних на світлодіодний дисплей (адреса дисплею199, десяткове). Для цього відкрити опцію «virtual devices» вікна програмного емулятора і вибрати закладку «LED Display».

```
mov ax, 0
x1:
  out 199, ax            ; обнулити покази дисплею
  inc ax                 ; збільшити число, що виводиться, на одиницю ( в кроко-
                         ; вому режимі)
jmp x1
hlt
```

ВАРІАНТ 4

Завдання 1

Записати та виконати команди обміну даними із стеком. Для цього відкрити опцію «stack» вікна програмного емулятора (нижній ряд).

```
org 100h
mov    ah, 09h
mov    al, 05h
mov    bh, 03h
mov    bl, 02h
```

```
call adr
adr:
  push ax     ; зберегти вміст ax в стеку
  push bx     ; зберегти вміст bx в стеку
  inc ax      ; збільшити вміст ax на «1»
  inc bx      ; збільшити вміст bx на «1»
  pop bx      ; повернути попереднє значення в ax
  pop ax      ; повернути попереднє значення в bx
ret           : повернути адресу повернення в IP
hlt
```

Завдання 2

Записати та виконати команди для організації циклів.

```
  mov cx, 00005h   ; завантажити лічильник циклів (регістр CX)
L1:
  loop L1      ; зменшення вмісту CX на одиницю і перехід на наступну
               ; команду, якщо CX=0
  mov ax,00abch
  hlt
```

Записати вміст регістрів МП, які змінюються в результаті виконання команд (табл.1).

Список рекомендованої літератури

4) 1. Самофалов К. Г., Викторов О. В. Микропроцессоры. – Б-ка инеженера – 2-е изд., перераб. и доп. - К: Техника, 1989.-312 с.

5) 2. Шевкопляс Б.В. Микропроцессорные структуры. Инженерные решения: Справочник. 2-е изд., перераб. и доп. -М.: Радио и связь, 1990, -512с.

6) 3. Угрюмов Е.П. Цифровая схемотехника. Спб.: BHV, 2001. 528 с.

4. Злобин В. К-, Григорьев В. Л. 58 Программирование арифметических операций в микропроцессорах: Учеб. пособие для технических вузов.— М.: Высш. шк., 1991. —303 с.: ил.