МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

**Український державний університет
науки і технологій**

Кафедра «Автоматика та телекомунікації»

В авторській редакції

В. І. Профатилов

МІКРОПРОЦЕСОРНІ ЗАСОБИ АВТОМАТИЗАЦІЇ

Навчально-методичні рекомендації
до лабораторних занять

Електронне видання

ДНІПРО
2024

УДК 004.31:656.25(076.5)
П 84

Автор:
В. І. Профатилов

Електронне видання

Схвалено Групою забезпечення якості освітньої програми
174.1.02 «Автоматика та автоматизація на транспорті»
Протокол № 3 від 01.02.2024

Схвалено Групою забезпечення якості освітньої програми
273.1.04 «Системи керування рухом поїздів»
Протокол № 5 від 15.01.2024

П 84 **Профатилов В. І.** Мікропроцесорні засоби автоматизації : навчально-методичні рекомендації до лабораторних занять / В. І. Профатилов ; Укр. держ. ун-т науки і технологій. – Електрон. вид. – Дніпро : УДУНТ, 2024. – 68 с.

Навчально-методичні рекомендації призначені для виконання лабораторних занять студентами, які здобувають освітній ступінь «бакалавр» по спеціальностям 174 «Автоматизація, комп'ютерно-інтегровані технології та робототехніка» (ОПП Автоматика та автоматизація на транспорті) та 273 «Залізничний транспорт» (ОПП Системи керування рухом поїздів) денної форми навчання.

Навчально-методичні рекомендації містять 9 лабораторних занять, у яких розглянуті арифметичні та логічні операції над двійковими та шістнадцятковими числами, система команд мікропроцесора 8086, а також приклади програмування на асемблері найбільш типових задач, таких як реалізація циклів та обробка масивів. Навчально-методичні рекомендації містять основні теоретичні відомості, порядок виконання лабораторних занять та питання для самоконтролю.

Іл. 5. Табл. 14. Бібліогр. 11 назв.

ЗМІСТ

ВСТУП

Метою вивчення дисципліни «Мікропроцесорні засоби автоматизації» є засвоєння основ мікропроцесорної техніки, елементної бази та архітектури мікропроцесорних пристроїв, навиків програмування мікропроцесорів на асемблері, експлуатації та принципів побудови мікропроцесорних засобів автоматизації на базі стандартних мікропроцесорних комплектів.

Студенти, що навчаються за освітньою програмою «Автоматика та автоматизація на транспорті», в результаті навчання по дисципліні отримають компетентності та будуть знати мікропроцесорну техніку на рівні, необхідному для розв'язання типових задач і проблем автоматизації та зв'язку, розробляти прикладне програмне забезпечення для мікропроцесорних систем управління на базі мікропроцесорів, вільно користуватись сучасними комп'ютерними та інформаційними технологіями для вирішення професійних завдань, використовувати прикладні та спеціалізовані комп'ютерно-інтегровані середовища для вирішення задач автоматизації та зв'язку.

Студенти, що навчаються за освітньою програмою «Системи керування рухом поїздів», в результаті навчання по дисципліні отримають компетентності та будуть мати навики використання інформаційних і комунікаційних технологій, здатність застосовувати знання мікропроцесорної техніки в обсязі, необхідному для розуміння процесів в автоматизованих системах керування рухом поїздів, пристроях залізничної автоматики, вміти розробляти проектно-конструкторську та технологічну документацію зі створення, експлуатації, ремонту та обслуговування систем керування рухом поїздів, пристроїв залізничної автоматики та їх елементів, використовуючи сучасні програмні засоби.

В результаті виконання лабораторних занять студенти будуть знати систему команд та мати навики програмування на асемблері для мікропроцесорів сімейства x86 на рівні, необхідному для розв'язання типових задач, вміти розробляти прикладне програмне забезпечення мікропроцесорних систем автоматизації, яке вирішує реальні задачі, уміння користуватися сучасними спеціалізованими програмними й апаратними засобами проектування мікропроцесорних систем, а також мати практичні навики діагностування і тестування мікропроцесорних пристроїв за допомогою сучасних апаратних та програмних засобів.

Вимоги до оформлення звіту з лабораторних занять:

– звіт можна оформляти у рукописному вигляді в зошиті або друкованому вигляді за допомогою комп'ютерної техніки на листах формату А4;

– якщо звіт виконується у друкованому вигляді, то необхідно дотримуватися вимог оформлення, що наведені у додатку Б;

– титульний лист звіту повинен містити інформацію, яка однозначно

ідентифікує його виконавця: назву дисципліни та вид занять, прізвище та ім’я студента, номер академічної групи, номер шифру або варіанту (якщо це є у вимогах лабораторних занять);

– звіт повинен включати номер лабораторних занять, тему та мету лабораторних занять;

– структура звіту повинна відповідати вимогам розділу лабораторних занять «Зміст звіту».

Вимоги з охорони праці та правила поведінки під час проведення лабораторних занять:

Перед початком проведення першого лабораторного заняття студент повинен пройти інструктаж з правил техніки безпеки та правил поводження у приміщенні, де проводяться заняття, а також розписатися в журналі проведення інструктажів. Студент, який не пройшов такий інструктаж, не допускається до виконання лабораторних занять.

Студент зобов’язаний виконувати вимоги правил техніки безпеки та правила поводження у приміщенні, де проводяться заняття. Якщо студент порушує вимоги правил техніки безпеки або правила роботи на комп’ютері, то викладач має право відсторонити студента від виконання лабораторних занять.

Якщо лабораторні заняття виконуються на комп’ютері, наданому університетом, то студент повинен виконувати наступні правила:

– забороняється змінювати налаштування операційної системи та програмного забезпечення, яке використовується для виконання лабораторних занять, окрім випадків, визначених викладачем;

– забороняється запускати стороннє програмне забезпечення, що не використовується для виконання лабораторних занять без дозволу викладача;

– у разі виникнення питань чи непередбаченої роботи обладнання необхідно повідомити викладача;

– забороняється підключати до комп’ютера власні накопичувачі на FLASH–пам’яті та копіювати з них на університетський комп’ютер файли без дозволу викладача.

Лабораторні заняття 1

СИСТЕМИ ЧИСЛЕННЯ В МІКРОПРОЦЕСОРНІЙ ТЕХНІЦІ

Мета роботи

Вивчити двійкову та шістнадцяткову системи числення. Навчитися переводити числа з двійкової та шістнадцяткової систем числення в десяткову та навпаки.

1. Короткі відомості з теорії

В сучасній мікропроцесорній техніці основною системою числення є двійкова система, алфавіт якої складається з двох цифр: «0» та «1». Значне широке поширення двійкової системи в мікропроцесорній техніці зумовлено такими її перевагами:

– найбільш проста та надійна технічна реалізація апаратної частини пристроїв на базі двопозиційних елементів (в основному використовуються транзисторні ключі);

– можливість використовувати апарат математичної логіки (алгебру Буля) без будь–яких додаткових перетворень.

У той же час двійкова система числення має й низку недоліків:

– вона не є загально прийнятною, тобто незручна для звичайних людей, що звикли до використання десяткової системи числення;

– при введенні даних або виведенні результатів необхідні додаткові перетворення з десяткової системи у двійкову та назад, що вимагає додаткових витрат часу;

– громіздкість запису великих двійкових чисел порівнянні з десятковими.

Для усунення останнього недоліку в мікропроцесорній техніці програмісти часто використовують шістнадцяткову систему, що відрізняється більш компактною формою запису. Крім цього, шістнадцяткова система числення має й інші переваги:

– зручність та простота переведення чисел із двійкової у шістнадцяткову систему числення та назад (чотири розряди двійкового числа дорівнюють одному розряду шістнадцяткового числа);

– зручність для програмістів при використанні команд адресації.

Алфавіт найбільш поширених систем числення, що використовуються в мікропроцесорній техніці, наведено в табл. 1.1.

Форми запису чисел у мікропроцесорній техніці:

– десяткове число – за умовчанням, або буква «*d*» наприкінці – 100*d*;

– двійкове число – буква «*b*» наприкінці – 100*b*;

– шістнадцяткове число – буква «*h*» наприкінці – 10F*h*;

– вісімкове число – буква «*o*» або «*q*» наприкінці – 100*q*.

Для запису двійкових чисел у мікропроцесорній техніці використовують різну кількість розрядів, що пов’язано з технічною реалізацією конкретного мікропроцесорного пристрою. У мікропроцесорній техніці використовують такі формати двійкових чисел:

– біт (1 розряд) – може набувати значення «0» або «1»;
– байт (8 розрядів) – діапазон значень 0...255 ($2^8 – 1$);
– слово (16 розрядів) – діапазон значень 0...65535 ($2^{16} – 1$);
– подвійне слово (32 розряди) – діапазон значень 0...4294967295 ($2^{32} – 1$).

Таблиця 1.1 – Алфавіт систем числення

Десяткова	Двійкова	Двійково–десяткова	Вісімкова	Шістнадцяткова
0	0000	0000	0	0
1	0001	0001	1	1
2	0010	0010	2	2
3	0011	0011	3	3
4	0100	0100	4	4
5	0101	0101	5	5
6	0110	0110	6	6
7	0111	0111	7	7
8	1000	1000	10	8
9	1001	1001	11	9
10	1010	0001.0000	12	A
11	1011	0001.0001	13	B
12	1100	0001.0010	14	C
13	1101	0001.0011	15	D
14	1110	0001.0100	16	E
15	1111	0001.0101	17	F
16	10000	0001.0110	20	10

Переведення числа з десяткової системи числення у будь–яку іншу здійснюється за таким правилом [9]:

– для переведення цілого числа N_{S1} із системи з основою $S1$ ($S1$=10) в іншу систему з основою $S2$ ($S2$=2, 8, 16) треба число N_{S1} послідовно ділити на основу $S2$ доти, поки не буде отримана частка менше $S2$;

– число N_{S2} у новій системі числення запишеться у вигляді залишків від ділення, починаючи з останнього.

Приклади переведення чисел із десяткової системи числення у двійкову та шістнадцяткову наведені нижче.

Перевести десяткове число $N1 = 59_{10}$ у двійкове $N2$:

$$
\begin{array}{r|l}
59 & 2 \\
\underline{58} & 29 \\
1 & \underline{28} \quad 14 \\
 & \quad 1 \quad \underline{14} \quad 7 \\
 & \quad\quad 0 \quad \underline{6} \quad 3 \\
 & \quad\quad\quad 1 \quad \underline{2} \quad 1 \\
 & \quad\quad\quad\quad 1
\end{array}
\qquad N2 = 111011b
$$

Перевести десяткове число $N1 = 59$ у шістнадцяткове число *N*2:

$$
\begin{array}{r|l}
59 & 16 \\
\underline{48} & 3 \\
11 &
\end{array}
\qquad N2 = 3Bh
$$

Переведення числа з будь–якої іншої системи числення в десяткову здійснюється за таким правилом [9]:

– число *N*1 з основою *S*1 (*S*1=2, 8, 16) представляють у вигляді суми ступенів основи, користуючись його позиційним записом;

– після цього підраховують значення десяткових еквівалентів окремих розрядів і, додавши їх, одержують число *N*2 у десятковій системі.

Приклади переведення чисел із двійкової та шістнадцяткової систем числення в десяткову наведені нижче.

Перевести двійкове число *N*1 у десяткове:

$$111011b = 1\cdot 2^5 + 1\cdot 2^4 + 1\cdot 2^3 + 0\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0 = 32 + 16 + 8 + 2 + 1 = 59.$$

Перевести шістнадцяткове число *N1* у десяткове:

$$3Bh = 3\cdot 16^1 + B\cdot 16^0 = 48 + 11 = 59.$$

Для переведення чисел із двійкової системи числення в шістнадцяткову або навпаки використовують таке правило: чотири розряди двійкового числа дорівнюють одному розрядові шістнадцяткового числа. Для переведення двійкового числа в шістнадцяткове його розбивають на частини по чотири розряди, починаючи з молодших бітів (тобто справа наліво). Якщо остання (старша) частина двійкового числа має менше чотирьох розрядів, то до неї дописують ліворуч нулі. Після цього кожну тетраду двійкового числа переводять у шістнадцяткову цифру відповідно до табл. 1.1. Для переведення шістнадцяткового числа у двійкове виконують зворотну операцію, замість кожної шістнадцяткової цифри записують чотири двійкових розряди відповідно до табл. 1.1. Приклади переведення чисел із двійкової системи числення в шістнадцяткову систему та навпаки [9]:

$$0011.1011b \to 3Bh;$$
$$1FAh \to 0001.1111.1010b.$$

2. Порядок виконання роботи

1. Використовуючи літературу та конспект лекцій, вивчити алфавіт двійкової та шістнадцяткової систем числення, а також правила переведення чисел з однієї системи в іншу.

2. За заданим варіантом (табл. 1.2) перевести числа з однієї системи чис-

лення в іншу, і результати занести у звіт:

– задані десяткові числа перевести у двійкову та шістнадцяткову системи числення;

– задані двійкові числа перевести в шістнадцяткову та десяткову системи числення;

– задані шістнадцяткові числа перевести у двійкову та десяткову системи числення;

– задані двійково-десяткові числа перевести в десяткову систему числення.

Таблиця 1.2 – Завдання до лабораторної роботи

Номер варіанта	Десяткове число	Двійкове число	Шістнадцяткове число	Двійково–десяткове число
1	2	3	4	5
1	208 96 137 246	10010101b 11101111b 11010000b 10111000b	3EH 54H B1H 2EH	0100.0000 0011.0011 1001.0011 0101.0110
2	161 180 116 187	10010101b 10110111b 11011001b 11000000b	ADH 40H 8BH 1DH	0011.0111 1001.0100 0100.1000 0001.1000
3	100 209 179 126	11001101b 01010010b 11110011b 11110000b	18H 86H EFH 77H	1000.0001 0010.0000 0101.0111 0110.0010
4	123 91 105 164	01010111b 10001111b 11000101b 10100100b	48H 8FH B8H AAH	0001.0001 0111.0101 0011.1000 0110.0110
5	111 138 249 212	10111011b 11001001b 01001100b 10101100b	3FH 54H 6EH E2H	0101.0100 1001.0100 0010.0010 0011.0001
6	122 211 287 183	01010011b 10011101b 01110101b 10011010b	CEH 21H 78H D8H	0111.0111 0101.0010 1001.1000 0010.0111
7	52 196 149 252	01000010b 01111010b 11101011b 10111001b	59H 7DH 90H 89H	0111.0011 0111.1000 0001.1001 0001.0101

Закінчення табл. 1.2

1	2	3	4	5
8	260	10011111b	78H	1001.0101
	49	11101101b	CCH	0101.0111
	211	10101110b	9BH	0110.0101
	161	01011100b	39H	0010.0110
9	202	10111000b	EFH	1000.0011
	141	11011110b	14H	0001.0001
	177	11100101b	FBH	0111.0010
	108	11011001b	B6H	1001.0111
10	118	10010101b	E0H	0011.0111
	254	10000110b	3FH	0010.0110
	154	01110101b	BEH	1000.0001
	191	11010101b	F1H	1001.0001
11	146	11011011b	40H	0111.0011
	69	01010011b	C0H	0100.0011
	216	11010010b	9BH	1001.0110
	141	01000011b	6BH	0001.0010
12	208	01000011b	88H	1001.0111
	203	00111101b	79H	0101.0100
	191	10010101b	FCH	0100.0100
	168	10101000b	DCH	0110.0001
13	80	01001010b	ADH	0100.0001
	313	11001001b	CCH	1000.0000
	133	01010000b	9DH	0010.0001
	63	10100001b	DAH	0001.0011
14	204	10101001b	62H	1001.0001
	147	00011101b	B1H	0011.0010
	85	11011110b	57H	0100.0010
	164	00100111b	A9H	0001.0111
15	160	10011000b	3BH	1000.0010
	171	01100101b	D8H	0100.1000
	258	01111110b	56H	0010.0111
	82	10010101b	EEH	0101.0101

3. Зміст звіту

1. Алфавіт систем числення, що використовуються в мікропроцесорній техніці (привести 20 перших чисел, починаючи з нуля).

2. Правила переведення чисел з однієї системи числення в іншу.

3. Результати переведення заданих чисел з однієї системи в іншу, а також проміжні обчислення, які при цьому виконувалися (варіант завдання видається викладачем).

4. Відповіді на контрольні запитання (за завданням викладача).

4. Контрольні запитання

1. Переваги двійкової системи числення.
2. Недоліки двійкової системи числення.
3. Переваги шістнадцяткової системи числення.
4. Що таке біт та які він має значення?
5. Що таке байт та який діапазон його значень?
6. Що таке слово та який діапазон його значень?
7. Що таке подвійне слово?
8. Правило переведення чисел із десяткової системи у двійкову систему.
9. Правило переведення чисел із десяткової системи у шістнадцяткову.
10. Правило переведення чисел із двійкової системи у десяткову.
11. Правило переведення чисел із шістнадцяткової у десяткову систему.
12. Алфавіт шістнадцяткової системи числення.
13. Правило переведення чисел із двійкової у шістнадцяткову систему числення.
14. Правило переведення чисел із шістнадцяткової у двійкову систему числення.

Лабораторні заняття 2

АРИФМЕТИЧНІ ТА ЛОГІЧНІ ОПЕРАЦІЇ НАД ДВІЙКОВИМИ ТА ШІСТНАДЦЯТКОВИМИ ЧИСЛАМИ

Мета роботи

Навчитися виконувати арифметичні й логічні операції над двійковими та шістнадцятковими числами.

1. Короткі відомості з теорії

Арифметичні операції над двійковими та шістнадцятковими числами здійснюються за тими самими правилами, що і для десяткових чисел, за винятком правил переносу при додаванні або позики при відніманні. При додаванні у двійковій системі перенос здійснюється, якщо сума дорівнює або більше числа два, а в шістнадцятковій – якщо більше 15. При відніманні багаторозрядних чисел може виникнути необхідність у позичанні з найближчого старшого розряду, який не дорівнює нулю. У двійковій системі позика одиниці зі старшого розряду дає двійку в молодший розряд і одиниці в усі нульові розряди, що містяться між молодшим розрядом і розрядом, із якого здійснюється позика. У шістнадцятковій системі позика одиниці зі старшого розряду дає в молодший розряд число 16, а в інші розряди, що рівні нулю, число 15.

Правила додавання та віднімання двійкових чисел:

$$0+0=0; \qquad 0-0=0;$$
$$1+0=1; \qquad 1-0=1;$$
$$0+1=1; \qquad 1-1=0;$$
$$1+1=10; \qquad 10-1=1.$$

Приклад 1. Додати два двійкових числа:

$$\begin{array}{r} 1101b=13 \\ + \quad 101b=5 \\ \hline 10010b=18. \end{array}$$

Приклад 2. Додати два шістнадцяткових числа:

$$\begin{array}{r} 68FEh \\ + \quad 234Fh \\ \hline 8C4Dh. \end{array}$$

Приклад 3. Виконати операцію віднімання двійкових чисел:

$$\begin{array}{r} 10010b=18 \\ - \quad 101b=5 \\ \hline 01101b=13. \end{array}$$

Приклад 4. Виконати операцію віднімання шістнадцяткових чисел:

$$\begin{array}{r} \mathrm{F000}h \\ - \quad 800\mathrm{F}h \\ \hline 6\mathrm{FF1}h. \end{array}$$

Для зображення від'ємних чисел у мікропроцесорній техніці використовують кодування. В сучасних мікропроцесорах використовується додатковий код, який дозволяє проводити арифметичні операції додавання та віднімання для знакових і без знакових чисел за одними і тими самими алгоритмами, що значно спрощує будову мікропроцесора і зменшує кількість його команд. Для відображення знака числа в мікропроцесорній техніці використовують старший розряд: якщо він дорівнює нулю, то число додатне, а якщо дорівнює одиниці, то число від'ємне.

Запис додатного двійкового числа в додатковому коді збігається із записом двійкового числа без знака. Від'ємне двійкове число в додатковому коді одержують за таким правилом [9]:

– перевести модуль десяткового числа у двійкове число, при цьому обов'язково доповнити число зліва необхідною кількістю нулів, щоб був присутній старший (знаковий) розряд – для байта це вісім розрядів, а для слова – 16 розрядів;

– в отриманому двійковому числі, необхідно інвертувати всі розряди, тобто замінити одиниці на нулі та навпаки;

– додати до молодшого розряду одиницю.

Приклад. Представити числа -1 та -128 у додатковому восьмирозрядному коді:

$$-1_{10} \to 0000.0001b \to 1111.1110b + 1b = 1111.1111_{\text{додат.код}}$$

$$-128_{10} \to 1000.0000b \to 0111.1111b + 1b = 1000.0000_{\text{додат. код}}$$

Приклад виконання завдання «Арифметичні операції над числами у додатковому код»:

Необхідно виконати операцію 33 – 66 в двійковому коді. Дану операцію можна замінити на операцію 33 + (–66) => і отримати результат –33 в двійковому додатковому коді.

1. Переведемо обидва числа в двійковий код, при цьому від'ємне число необхідно представити в додаткову коді:

33 → 0010.0001b

–66 → 0100.0010b → 1011.1101b +1b → 1011.1110b

2. Виконаємо операцію додавання:

	0	0	1	0.	0	0	0	1	b	=	33
+	1	0	1	1.	1	1	1	0	b	=	– 66
	1	1	0	1.	1	1	1	1	b	=	– 33

3. Перевірка результату (необхідно від отриманого результату відняти одиницю та про інвертувати отримане число):

1101.1111b – 1b = 1101.1110b → 0010.0001b => |33|

Логічні операції проводяться тільки над двійковими числами. Тому, якщо число задане в десятковій або шістнадцятковій системі числення, його треба попередньо перевести у двійкове, виконати логічну операцію, а потім результат знову перевести з двійкової системи числення у вихідну систему числення. На відміну від арифметичних операцій, у яких розряди числа пов'язані між собою, тобто можливі переноси або позика, у логічних операціях усі розряди двійкового числа є незалежними. Сучасні мікропроцесори підтримують виконання чотирьох логічних операцій:

– **інвертування** – синоніми логічної операції: НІ, NOT, ЗАПЕРЕЧЕННЯ. При інвертуванні двійкового числа, кожен його біт змінює свій стан на протилежне значення: $0 \to 1, 1 \to 0$.

– логічне множення – синоніми логічної операції: І, AND, КОН'ЮН–КЦІЯ. Цю операцію позначають: AND або &. Правила логічного множення:

$$0 \text{ AND } 0 = 0;$$

$$1 \text{ AND } 0 = 0;$$

$$0 \text{ AND } 1 = 0;$$

$$1 \text{ AND } 1 = 1.$$

– **логічне додавання** – синоніми логічної операції: АБО, OR, ДИЗ'ЮНКЦІЯ. Цю операцію позначають OR. Правила логічного додавання:

$$0 \text{ OR } 0 = 0;$$

$$1 \text{ OR } 0 = 1;$$

$$0 \text{ OR } 1 = 1;$$

$$1 \text{ OR } 1 = 1.$$

– **логічна операція XOR** – синонім логічної операції: додавання по модулю два, виключне АБО. Логічну операцію позначають XOR або ⊕. Правила логічної операції XOR:

$$0 \text{ XOR } 0 = 0;$$

$$0 \text{ XOR } 1 = 1;$$

$$1 \text{ XOR } 0 = 1;$$

$$1 \text{ XOR } 1 = 0.$$

Приклад виконання завдання «Логічні операції над шістнадцятковими числами»:

Необхідно виконати логічні операції AND, OR і XOR над заданою парою чисел в шістнадцятковій системі: 57DFh та 2BA3h.

57DFh → 0101.0111.1101.1111b
2BA3h → 0010.1011.1010.0011b
AND 03 83h ← 0000.0011.1000.0011b
OR 7FFFh ← 0111.1111.1111.1111b
XOR 7C7Ch ← 0111.1100.0111.1100b

2. Порядок виконання роботи

1. Використовуючи літературу й конспект лекцій, вивчити правила виконання арифметичних і логічних операцій над двійковими та шістнадцятковими числами.

2. За заданим варіантом (табл. 2.1) виконати арифметичні операції над двійковими та шістнадцятковими числами (варіант завдання видається викладачем).

Таблиця 2.1 – Індивідуальні завдання до лабораторної роботи

Номер варіанта	Арифметичні дії над двійковими числами		Арифметичні дії над шістнадцятковими числами	
1	2	3	4	5
1	10010011b+11001010b 11010110b+00011010b 10010001b+00110101b	10011000b–10000011b 11001000b–01000111b 10010110b–01001100b	482Eh+7A81h 17F5h+2A51h 3B1Ch+1921h	7E2Dh–7AF2h 1F2Eh–1ECFh 38A4h–185Eh
2	11101100b+01010100b 10010110b+10010100b 10111110b+11000010b	01011101b–00010110b 01110110b–01010010b 11101001b–10010010b	7873h+2901h 7E87h+106Ah 4B5Fh+70A4h	3E84h–1087h D569h–1E27h CBAAh–1BEAh
3	00111000b+11111010b 01010010b+01110001b 01001101b+00111110b	11000000b–10001011b 11110100b–01010011b 11110001b–11010001b	1C2Ah+5F2Eh 380Eh+7FA5h 1A3Ch+3E60h	6853h–422Fh 9C1Dh–7DA9h 98C3h–1A45h

Закінчення табл. 2.1

1	2	3	4	5
4	10010100b+01110110b 11100010b+01010011b 10100110b+11011010b	11101001b–11010110b 11010111b–00011010b 01110001b–00110011b	262Ah+149Bh 3DD9h+4C14h 4B81h+46E1h	F37Eh–949Ch F2BAh–1871h F408h–100Bh
5	11010000b+00010010b 01011001b+01010110b 10010111b+10011011b	10111010b–10101011b 11111010b–01100100b 11100010b–01010110b	1B28h+4AD5h 3B39h+23C5h 14FEh+38B9h	EF26h–2D9Ch 619Eh–2BE1h F6DBh–ACA4h
6	01100101b+01101100b 01000110b+01011011b 11100100b+11011001b	11111100b–10011011b 10000111b–01111100b 01110100b–01101011b	520Fh+6D6Eh 3D7Dh+1EC9h 5E84h+6C78h	FE6Ch–56BAh DE30h–2A50h 3E5Ch–33AFh
7	11101001b+10111000b 01010110b+11101011b 10100101b+11010011b	11011100b–10001111b 01001100b–00111110b 10100100b–10000101b	33A7h+71F6h 73F5h+6F2Eh 51E2h+24ADh	FD98h–D86Eh DBA6h–6449h D4CAh–BBA9h
8	00011101b+00111111b 11100010b+01100111b 11011001b+11011010b	11100110b–11001000b 10111101b–01101000b 11101001b–10101111b	7ABFh+67FDh 6FF2h+21A7h 1AE9h+4DE8h	C84Eh–33D9h 5CCAh–1EF2h 33E3h–23E8h
9	11011111b+11011000b 11101100b+01101001b 11100010b+01110010b	10001001b–01011100b 11100100b–10011100b 11001011b–00101110b	7468h+29F2h 64C4h+5A61h 5768h+3EBAh	AF1Ah–2D2Bh 3CC3h–2A55h C02Ah–97B5h
10	10110111b+01110110b 10101010b+11100010b 00111010b+00110100b	10111010b–10010010b 11100001b–11011011b 11100100b–11010101b	3EB2h+18C5h 67BAh+1B4Fh 7730h+77B3h	FA24h–48FCh D0C8h–7388h 4D9Eh–1CC4h
11	01001111b+10011010b 01110011b+00101001b 11111010b+10011110b	10100111b–00110001b 01010011b–00100101b 11011001b–10110110b	1B2Dh+64A6h 3026h+7DDEh 5C10h+6362h	7F1Fh–365Bh A304h–15F4h 4264h–1D5Fh
12	11110011b+11011100b 10100001b+01000010b 01010101b+10011001b	00101000b–00011110b 01101110b–01010100b 11110111b–00101011b	5C67h+4447h 5754h+7621h 69D9h+1658h	F323h–7CEEh 8063h–676Ah 5C34h–448Dh
13	00010110b+01100111b 10101001b+10110010b 11011011b+10000001b	01111011b–01001111b 11110011b–00101110b 01000000b–00010011b	4732h+3936h 1899h+1227h 365Bh+1F24h	EAA8h–7CFFh 3838h–3359h 61D6h–2C48h
14	11001100b+01111010b 01001101b+10110011b 10010110b+10011101b	10100001b–01010101b 11010110b–10111001b 10100010b–10011001b	325Eh+4F03h 1141h+39FEh 4D64h+76A6h	C94Eh–7FACh BD70h–9991h B1A8h–1EB0h
15	01010001b+11101000b 10010011b+10101111b 10110001b+10010101b	11101011b–11010100b 01100110b–01011010b 11101110b–00111010b	16BAh+5B3Eh 109Eh+6FB7h 5CC7h+5E7Fh	CA17h–AEB6h B090h–8E54h C30Ch–85E5h

3. За заданим варіантом (табл. 2.2) перевести десяткові числа в додатковий код, виконати над ними арифметичні дії і результат знову перевести в десяткову систему числення (варіант завдання видається викладачем).

4. За заданим варіантом (табл. 2.2) виконати логічні операції AND, OR і XOR над заданими шістнадцятковими числами (варіант завдання видається викладачем).

Таблиця 2.2 – Індивідуальні завдання до лабораторної роботи

Номер варіанта	Арифметичні операції над числами у додатковому коді	Логічні операції над шістнадцятковими числами	
		Перше число	Друге число
1	2	3	4
1	–91 + 78 61 – 113 74 – 121	BA7EH 3A9AH 21E1H	595FH CD54H EE07H
2	–114 + 83 –96 + 42 25 – 76	5C30H C847H 6EECH	19E2H 26A3H DBFCH
3	46 – 91 57 – 77 –94 + 8	7306H 82C3H D1B1H	1337H E3C9H C504H
4	30 – 80 77 – 93 –63 + 40	E3E7H B21DH AECEH	1CABH 5298H F146H
5	37 – 59 –23 + 108 –85 + 53	745DH 242CH BEC4H	DBA3H 3F7EH BD3BH
6	46 – 111 –32 + 29 105 – 106	7CF3H 71E9H FE59H	7C77H A59EH 8895H
7	103 – 128 –115 + 102 7 – 80	DC9DH 8DD1H 2FF6H	F851H 837DH 7217H
8	–91 + 34 –53 + 30 33 – 113	1510H 3347H 8608H	B012H 1CF5H 7878H
9	–77 + 61 –55 + 65 12 – 73	3824H 1EEFH 4E62H	242DH B214H D844H
10	–116 + 70 66 – 122 –91 + 84	4E38H B813H B697H	2189H B2FFH 3EBBH
11	90 – 94 –104 + 97 –89 + 33	D34AH F9C5H BA6CH	2860H 58B5H FB85H
12	30 – 95 14 – 116 –88 + 39	991CH CC93H 7598H	A329H 68BDH C242H
13	26 – 73 –61 + 14 37 – 68	8F12H 5F2FH 717DH	19BCH 3D53H C074H

Закінчення табл. 2.2

1	2	3	4
14	17 – 67 64 – 76 –116 + 47	6213H 11A9H 9ABAH	25F0H 4B1BH 4BEEH
15	44 – 110 37 – 93 –109 + 83	5FA9H 49D8H F1D0H	1824H B15FH 5A44H

3. Зміст звіту

1. Правила арифметичних та логічних операцій над двійковими числами.
2. Результати арифметичних та логічних операцій над двійковими та шістнадцятковими числами, а також проміжні обчислення, які при цьому виконувалися.
3. Відповіді на контрольні запитання (за завданням викладача).

4. Контрольні запитання

1. Правила додавання двійкових чисел.
2. Правила віднімання двійкових чисел.
3. Для чого використовується додатковий код?
4. Правило отримання від’ємного числа в додатковому коді.
5. Що таке інвертування?
6. Правила логічного множення AND двійкових чисел.
7. Правила логічного додавання OR двійкових чисел.
8. Правила логічної операції XOR двійкових чисел.
9. Різниця між арифметичними та логічними операціями.
10. Діапазон восьмирозрядних двійкових чисел у додатковому коді?
11. Діапазон шістнадцятирозрядних двійкових чисел у додатковому коді?

Лабораторні заняття 3

ПРОГРАМУВАННЯ НА АСЕМБЛЕРІ МІКРОПРОЦЕСОРА 8086

Мета роботи

Вивчити регістри та організацію пам’яті мікропроцесора 8086, а також навчитися виконувати компіляцію та тестування програм на асемблері мікропроцесора 8086.

1. Короткі відомості з теорії

Система команд мікропроцесора 8086 є базовою для сімейства мікропроцесорів x86 фірми Intel (8086, 80286, 80386, 80486, Pentium та інші), тобто

програма, що написана на асемблері мікропроцесора 8086, без жодних змін працюватиме на старших моделях. Для будь-якого мікропроцесора програма є набором команд, кожній з яких відповідає свій машинний код. Навіть найпростіші восьмирозрядні мікропроцесори можуть мати декілька сотень команд, а система команд сучасних 32-розрядних мікропроцесорів – декілька тисяч команд, що робить програмування в машинному коді дуже складним. Для спрощення програмування на машинній мові для кожного мікропроцесора розробляється асемблер, що являє собою символьну форму запису машинної мови, яка більш звична для людей.

Для розробки програмного забезпечення на асемблері x86 мікропроцесорів найчастіше використовуються компілятори MASM фірми Microsoft або TASM (TURBO ASSEMBLER) фірми Borland. Сучасні компілятори з мови асемблера зазвичай підтримують синтаксис одного з цих, або обидва види компіляторів. На відміну від мов високого рівня, таких як Сі або Паскаль, асемблер дозволяє одержувати найефективніші програми, які матимуть набагато менший розмір і виконуються з найбільшою швидкістю. Асемблер зазвичай використовують при розробці програмного забезпечення мікропроцесорних систем реального часу, базових модулів операційних систем, програм призначених для керування зовнішніми пристроями (драйверів), а також компіляторів для мов високого рівня. Для розробки програми на асемблері програміст обов'язково повинен знати будову мікропроцесора, тобто склад та призначення його регістрів, організацію пам'яті, методи адресації даних та ін., а також вивчити одну з мов асемблера.

Структура регістрів мікропроцесора 8086 наведена на рис. 3.1. Мікропроцесор 8086 має 14 програмно доступних регістрів, які за призначенням можна розбити на чотири групи:

– регістри загального призначення – *AX, BX, CX, DX, SI, DI, BP, SP*;

– сегментні регістри – *CS, DS, SS, ES*;

– регістр (покажчик) команд – *IP*;

– регістр – *Flags*.

Усі регістри мікропроцесора 8086 є 16-розрядними, тобто можуть оперувати з даними у вигляді слова. Регістри загального призначення (РЗП) можна використовувати в будь-яких арифметичних і логічних операціях. Окрім цього частина РЗП має свої особливості:

– регістри ***AX, BX, CX, DX*** можна використовувати як два 8-розрядні регістри, що дозволяє звертатися окремо до старшого та молодшого байтів цих регістрів;

– регістри ***SI, DI, BP, BX*** можна використовувати як **модифікатори адреси** при звертанні мікропроцесора до чарунок пам'яті;

– регістр ***SP*** використовується для роботи із **стеком**, який є областю пам'яті мікропроцесорної системи, що працює за принципом: «останній прийшов – перший пішов».

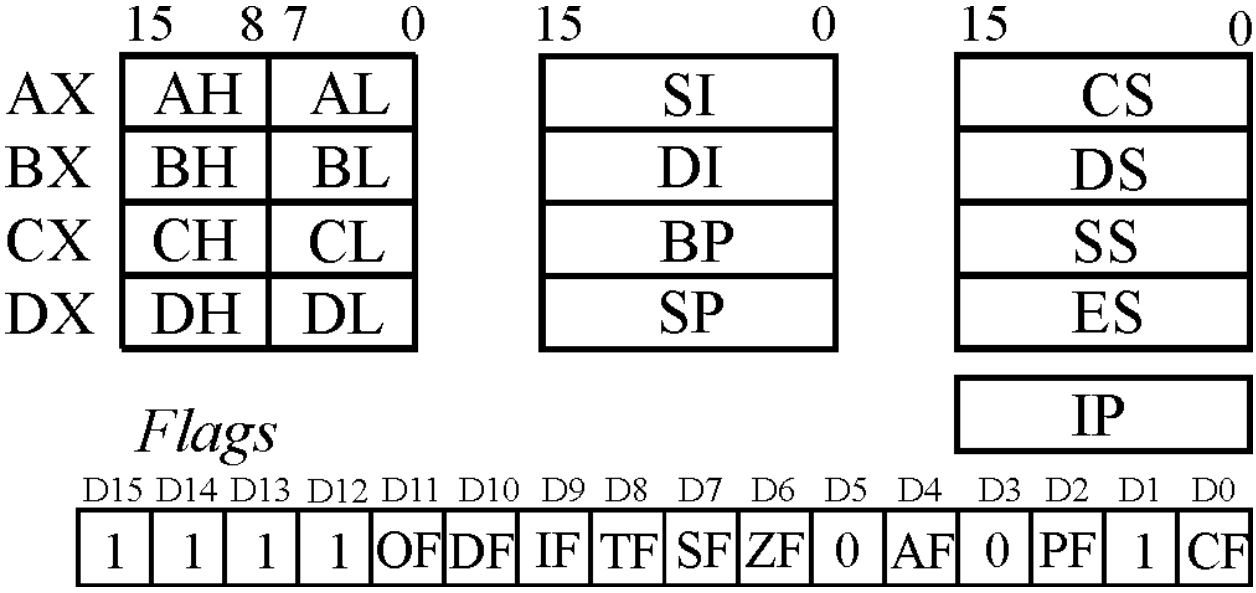


Рис. 3.1. Регістри мікропроцесора 8086

Усі регістри мікропроцесора 8086 є 16-розрядними, тобто можуть оперувати з даними у вигляді слова. Регістри загального призначення (РЗП) можна використовувати в будь-яких арифметичних і логічних операціях. Окрім цього частина РЗП має свої особливості:

– регістри ***AX, BX, CX, DX*** можна використовувати як два 8-розрядні регістри, що дозволяє звертатися окремо до старшого та молодшого байтів цих регістрів;

– регістри ***SI, DI, BP, BX*** можна використовувати як **модифікатори адреси** при звертанні мікропроцесора до чарунок пам'яті;

– регістр ***SP*** використовується для роботи із **стеком**, який є областю пам'яті мікропроцесорної системи, що працює за принципом: «останній прийшов – перший пішов».

Сегментні регістри застосовуються для сегментації адреси при звертанні мікропроцесора до чарунок пам'яті, що дозволяє скоротити розмір команд. Мікропроцесор 8086 підтримує одночасну роботу з чотирма сегментами:

– ***CS*** – сегмент **команд**, який указує на початок області пам'яті, де розміщуються команди програми;

– ***DS*** – сегмент **даних**, який указує на початок області пам'яті, де розміщуються дані програми;

– ***SS*** – сегмент **стека**, який указує на початок області пам'яті, де розміщується стек програми;

– ***ES*** – **додатковий** сегмент **даних**, який використовується програмістом на свій розсуд.

У мікропроцесорі 8086 розмір сегмента обмежено об'ємом 64 Кбайти ($2^{16} = 65536$). Ці регістри не можна використовувати в арифметичних і логічних операціях. У сегментні регістри можна тільки записувати дані або читати з них.

Покажчик команд ***IP*** використовується для зберігання початкової адреси команди, яка повинна бути виконана наступною, після закінчення поточної команди. Фізична адреса команди визначається мікропроцесором за допомогою регістрів *CS* і *IP*. Регістр *CS* указує на початок сегмента команд, а регістр *IP* – на адресу команди, яка відлічується від початку сегмента команд. Вміст регістра *IP* можуть змінювати тільки команди переходів, у жодних арифметичних або логічних операціях цей регістр брати участі не може.

Регістр ***Flags*** складається з 16 незалежних бітів, з яких у мікропроцесорі 8086 використовуються тільки дев'ять. Усі регістри *Flags* можна розбити на дві групи: біти умов та біти стану.

Біти умов автоматично змінюються самим мікропроцесором після кожної арифметичної або логічної операції й фіксують властивості результату:

– ***CF*** – біт **переносу**, використовується в арифметичних операціях над числами без знака. Біт установлюється в одиницю, якщо при додаванні або відніманні результат не вміщується в діапазон чисел, який здатний зберегти регістр або елемент пам'яті;

– ***OF*** – біт **переповнення**. Його призначення аналогічно біту переносу, тільки використовується для чисел із знаком;

– ***ZF*** – біт **нуля**. Установлюється в одиницю, якщо результат арифметичної або логічної операції рівний нулю;

– ***SF*** – біт **знака**. Установлюється в одиницю, якщо результат операції над знаковим числом вийшов від'ємним;

– ***PF*** – біт **парності**. Установлюється в одиницю, якщо у восьми молодших бітах результату міститься парне число одиниць;

– ***AF*** – біт **додаткового переносу**. Використовується при двійково–десятковій арифметиці. Установлюється в одиницю, якщо при додаванні відбувся перенос із молодшої половини байта в старшу половину, а також, якщо при відніманні відбулася позика зі старшої половини байта в молодшу половину байта.

Біти стану необхідно змінювати програмно, а стан цих бітів має вплив на режими роботи мікропроцесора:

– ***DF*** – біт **напряму** встановлює напрям перегляду рядків даних (одномірних масивів): 0 – прямий напрям перегляду рядка від початку до кінця, 1 – зворотний порядок перегляду від кінця до початку;

– ***IF*** – біт **переривання**: 0 – переривання заборонені, 1 – переривання дозволені;

– ***TF*** – біт **трасування**: 1 – включений режим трасування, тобто після виконання кожної команди мікропроцесор викликає переривання, що можна використовувати для тестування програми.

Мікропроцесор 8086 працює з пам'яттю, яка складається з чарунок розміром у байт. Кожна чарунка пам'яті має свою унікальну адресу, при формуванні якої мікропроцесор 8086 використовує 20-розрядну шину адреси. Максимальний об'єм пам'яті, із яким може працювати мікропроцесор 8086, складає 1 Мбайт (діапазон адреси 00000*h*…*FFFFFh*). Мікропроцесор 8086 може

працювати як із байтами, так і зі словами, що розміщуються в пам'яті у двох сусідніх чарунках пам'яті. Причому молодші вісім розрядів слова розміщуються в першій чарунці пам'яті, а старші вісім біт – у другій, тобто слово зберігається в пам'яті в перевернутому вигляді.

Уся пам'ять мікропроцесорного пристрою умовно розподіляється мікропроцесором 8086 на сегменти. Фізичну адресу чарунки пам'яті формує мікропроцесор, а програміст працює з логічними адресами усередині сегмента. Фізична адреса формується шляхом додавання початкової адреси сегмента та зсуву в його межах. У мікропроцесорі 8086 початкова адреса сегмента зберігається в сегментних регістрах. Для отримання фізичної 20-розрядної адреси чарунки пам'яті мікропроцесор виконує такі дії:

– множить вміст сегментного регістра на 16 (додає справа чотири нулі);

– додає до отриманого результату вміст відповідного регістра (залежить від типу сегмента): $CS \cdot 16 + IP$, $SS \cdot 16 + SP$, $DS(ES) \cdot 16 + (BX, BP, SI, DI)$.

Використання сегментації пам'яті дозволяє указувати в командах мікропроцесора 8086 тільки зсув у межах сегмента, внаслідок чого зменшується довжина команд, а отже, і розмір програм.

Кожна команда на асемблері займає окремий рядок, що складається з трьох полів:

МІТКА КОМАНДА КОМЕНТАР

Кожне поле відділяється одне від одного за допомогою пропусків (число пропусків може бути довільним). Обов'язковою є наявність тільки поля команди, решта полів можуть бути відсутніми. За відсутністю поля мітки поле команди все одно повинне починатися з пропуску, тобто поле команди повинне розміщуватися як мінімум із другої позиції рядка.

У **полі мітки** розміщується символічне ім'я елемента пам'яті, у якій зберігається відзначена команда. Мітка є словом завдовжки не більше 31 символу, що починається з літери. У мітці допускається використовувати латинські букви, цифри та символ підкреслення «_». Мітка завжди повинна починатися з першої позиції рядка, і якщо вона вказує на команду мікропроцесора, то повинна закінчуватися символом двокрапка «:». Забороняється використовувати в як мітку зарезервовані слова асемблера: імена команд і регістрів мікропроцесора 8086, імена директив і операторів асемблера.

У **полі команд** записується мнемонічне позначення команди мікропроцесора або директиви асемблера, а також значення операндів, над якими виконується операція.

Поле коментарів використовується програмістом для розміщення пояснень до дій, що виконуються в цьому місті програми. Це поле починається після символу крапки з комою «;». Поле повністю ігнорується компілятором асемблера, тому в ньому допускається використання будь-яких символів, у тому числі й українських та російських літер.

У кінці програми **обов'язково** треба указувати директиву **END** – кінець програми на асемблері.

Розробка програми на асемблері. Алгоритм розробки програми на мові асемблера включає такі етапи:

1. **Підготовка початкового тексту** програми на асемблері. Для набору й редагування тексту програми можна використовувати будь-який текстовий редактор. Текст програми необхідно зберегти у файлі з розширенням ***.asm**.

2. **Компіляція програми**. Здійснюється за допомогою компілятора асемблера, який перевіряє початковий текст програми на наявність помилок і, якщо їх немає, створює файл із заданим розширенням, наприклад ***.exe**, який містить машинний код, призначений для виконання на будь-якому мікропроцесорі сімейства x86. Якщо ж асемблер знайде в тексті програми помилки, то процес компіляції буде перерваний і на екран виводиться повідомлення про помилку. У цьому випадку необхідно повернутися на перший етап і виправити помилки в початковому тексті програми.

3. **Тестування програми**. На цьому етапі програміст здійснює тестування своєї програми з метою перевірки правильності її роботи відповідно до заданого алгоритму. Для тестування програми на асемблері використовуються емулятори що імітують роботу реального мікропроцесора, наприклад «emu8086».

2. Порядок виконання роботи

1. Запустити програму-емулятор мікропроцесора 8086 «emu8086». Натиснути у вікні «welcome...» кнопку «new», вибрати у вікні «choose code template» шаблон для програми з розширенням EXE і натиснути кнопку «ОК».

2. У текстовий редактор буде завантажено шаблон для компіляції exe–файлу. Набрати навчальну програму в текстовому редакторі та зберегти під ім’ям «Lr3.asm». Текст на мові асемблера не відрізняє регістр символів, тому команди та мітки можна набирати як великими так і малими літерами.

3. Запустити компіляцію програми натиснувши кнопку «compile». Якщо помилок у програмі немає, то буде створений файл «Lr3.exe» і треба натиснути кнопку «Save» (Зберегти), а у вікні «assembler status» буде активовано кнопку «run», що дозволяє запустити програму на виконання. Якщо в початковому тексті програми будуть знайдені помилки, то у вікні «assembler status» буде вказана кількість помилок і номери рядків, у яких вони містяться. Після усунення помилок необхідно повторити пункт 3.

4. Для тестування роботи програми «Lr3.exe» необхідно запустити емулятор (кнопка «emulate») та програму на виконання шляхом натиснення кнопки «run». Також кнопка «run» присутня у вікні «assembler status» після компіляції програми.

Після запуску програми з’явиться вікно для виведення текстових повідомлень «emulator screen», що є аналогом екрана монітора в DOS-режимі (текстовий режим 80x25 символів). Для очистки вікна використовується кнопка «clear screen». Для виведення текстових повідомлень українською або російською мовою необхідно вибрати шрифт з підтримкою кирилиці, наприклад

«Courier New». Для вибору типу та розміру шрифту необхідно натиснути кнопку «change font».

Навчальна програма на асемблері мікропроцесора 8086

```
; Лабораторна робота 3
;_____________________________________
; сегмент даних програми
DATA SEGMENT
TEXT1 DB "Введіть своє прізвище або ім’я", 13, 10, '$'
BUFFER DB 15, ?, 15 DUP ('   '), 10, '$'
TEXT2 DB 13, 10, "Вітаю $"
TEXT3 DB 13, 10, "Для виходу із програми натисніть будь–яку клавішу$"
ends; кінець сегмента даних
;_____________________________________
; сегмент стека програми на 256 байт
STACK SEGMENT
        DW 128 DUP(0)
ends; кінець сегмента стека
;_____________________________________
; сегмент коду програми
CODE SEGMENT
START:
; налаштування сегментних регістрів для доступу до даних
        MOV AX, DATA
        MOV DS, AX
        MOV ES, AX
;_____________________________
; Основна програма
;вивід тексту на екран
        MOV AH,9
        LEA DX, TEXT1
        INT 21H
;очистка буфера вводу клавіатури
        MOV AL,0
        MOV AH,0CH
        INT 21H
;ввід рядка з клавіатури
        LEA DX,BUFFER
        MOV AH,0AH
        INT 21H
;вивід рядка
        MOV AH,9
        LEA DX,TEXT2
        INT 21H
        MOV BL,BUFFER+1
```

```
        MOV BH,0
        MOV BUFFER[BX]+2,'$'
        MOV AH,9
        LEA DX,BUFFER+2
        INT 21H
;_____________________________
; вихід із програми у Windows ;
вивід рядка-підказки на екран
        LEA DX, TEXT3
        MOV AH, 9
        INT 21H
; очікування натиснення будь-якої клавіші
        MOV AH, 1
        INT 21H
; повернення в операційну систему
        MOV AX, 4C00H
        INT 21H
ends; кінець сегмента коду

END START; кінець програми та місце входу в основну програму
```

3. Зміст звіту

1. Структура регістрів мікропроцесора 8086 та короткий опис особливостей їх використовування.
2. Етапи розробки програми на асемблері.
3. Навчальна програма на асемблері мікропроцесора 8086.

4. Контрольні запитання

1. Призначення мови програмування – асемблера.
2. Перелік та призначення РЗП мікропроцесора 8086.
3. Призначення сегментних регістрів мікропроцесора 8086.
4. Призначення біт регістра *Flags*.
5. Призначення покажчика команд *IP*.
6. Організація пам’яті в мікропроцесорі 8086.
7. Як формується фізична 20-розрядна адреса в мікропроцесорі 8086?
8. Структура команди на асемблері.
9. Етапи розробки програми на асемблері.

Лабораторні заняття 4

КОМАНДИ ПЕРЕМІЩЕННЯ ДАНИХ МІКРОПРОЦЕСОРА 8086

Мета роботи

Вивчити команди переміщення даних та команди для роботи зі стеком мікропроцесора 8086, а також навчитися тестувати розроблені програми за допомогою емулятора «emu8086».

1. Короткі відомості з теорії

Кожна команда мікропроцесора 8086 складається з коду операції, який указує функціональне призначення команди, а також з операндів, які беруть участь в операції. Мікропроцесор 8086 підтримує такі типи операндів:

– константи – *i*8 (байт), *i*16 (слово);

– вміст регістрів – *r*8 (байт), *r*16 (слово);

– вміст чарунки пам'яті – *m*8 (байт), *m*16 (слово);

– сегментний регістр – *sr* (слово).

Основну масу команд будь-якого мікропроцесора складають команди переміщення, які дозволяють переміщати дані з пам'яті в регістри мікропроцесора та назад. Мікропроцесор 8086 має велику кількість команд переміщення даних, але всі вони записуються у форматі [4]:

$$MOV\ op1,\ op2\ \ (op1 \Leftarrow op2).$$

За командою *MOV* на місце першого операнда переміщується значення другого операнда, при цьому значення другого операнда не змінюється. Допустимі комбінації операндів команди пересилки *MOV* наведені в табл. 4.1.

Таблиця 4.1 – Допустимі комбінації операндів команди *MOV*

*op*1	*op*2
*r*8	*i*8, *r*8, *m*8
*m*8	*i*8, *r*8
*r*16	*i*16, *r*16, *m*16, *sr*
*m*16	*i*16, *r*16, *sr*
sr	*r*16, *m*16 (не можна записувати в регістр *CS*)

Для перестановки місцями даних до складу мікропроцесора 8086 входить команда *XCHG*, яка має формат [4]:

$$XCHG\ op1,\ op2\ \ (op1 \Leftrightarrow op2).$$

Команда *XCHG* переставляє місцями значення першого та другого операндів. Допустимі комбінації операндів команди *XCHG* наведені в табл. 4.2.

Таблиця 4.2 – Допустимі комбінації операндів команди XCHG

*op*1	*op*2
*r*8	*r*8, *m*8
*m*8	*r*8
*r*16	*r*16, *m*16
*m*16	*r*16

При використанні команд мікропроцесора 8086 *MOV* і *XCHG* необхідно звернути увагу, що в них **відсутні команди переміщення даних з однієї чарунки пам'яті в іншу в обхід мікропроцесора.**

Використовуючи в програмі змінні та константи, необхідно зарезервувати для них місце в пам'яті мікропроцесорного пристрою. Для цього використовуються такі директиви асемблера [7]:

DB (define byte) – **резервує одну** чарунку пам'яті під змінну розміром у байт;

DW (define word) – **резервує дві** чарунки пам'яті під змінну розміром у слово;

DD (define double word) – **резервує чотири** чарунки пам'яті під змінну розміром у подвійне слово;

EQU (equal, дорівнює) – **визначає константу**.

Формат директиви асемблера DB (директиви DW і DD мають аналогічний формат):

< ім'я змінної > DB < операнд >, < операнд >

Операнд задається або у вигляді числа в діапазоні від –128 до 255, або у вигляді символу невизначеності «?», за яким чарунка пам'яті резервується під змінну, але її значення не визначається, тобто може бути будь-яким.

Операнди директив DW і DD відрізняються тільки діапазоном чисел змінної. Значення змінних можна задавати в десятковій, двійковій або шістнадцятковій формах запису, а також у вигляді рядків символів (тоді символи переводяться автоматично в коди стандарту *ASCII*). Використовуючи числа як операнди, необхідно указувати їхню систему числення. За умовчанням число вважається десятковим, якщо воно подано в 16–ій формі, то в кінці нього ставиться символ – *h*, якщо число двійкове, то в кінці ставиться символ – *b*. У директиві може бути кілька операндів, і тоді вони відокремлюються один від одного комами. Приклад застосування директив завдання змінних:

X1 DB 34h, 0FAh; резервуються дві чарунки пам'яті розміром у байт під змінну *X*1 з початковими значеннями 34*h* і *FAh*.

*Y*2 *DW* ?; резервується дві чарунки пам'яті під змінну *Y*2 розміром у слово з невизначеним початковим значенням.

Директива *EQU* призначена для опису констант. Вона має такий формат: <ім'я константи > *EQU* < операнд >. Директива *EQU* не резервує в пам'яті місце під константу, просто скрізь, де в програмі траплятиметься ім'я константи, асемблер замість нього підставлятиме значення операнда. Одна директива *EQU* може мати тільки один операнд. Приклад застосування директиви *EQU*:

*MAS*1 *EQU* 100; значення константи *MAS*1 = 100.

Для зберігання та переміщення даних в обчислювальних системах на базі мікропроцесора 8086 часто використовують програмний стек, який є спеціальною областю пам'яті, робота з якою ведеться за правилом: елемент, записаний у стек останнім, читається першим. Стек зручно використовувати для роботи з упорядкованими даними, оскільки їх розташування наперед відоме. Максимальний розмір стека мікропроцесора 8086 складає 64 Кбайти. Для роботи із стеком мікропроцесор використовує два регістри:

– ***SS*** – у сегментному регістрі стека зберігається **адреса початку** сегмента стека (основа стека, яка завжди фіксована);

– ***SP*** – покажчик стека, у якому зберігається **поточне положення** вершини стека (змінюється після кожного звернення до стека).

Для роботи із стеком мікропроцесор 8086 має такі команди [4]:

– запис у стек слова: *PUSH op* $(op = r16, m16, sr)$ – команда *PUSH* (записувати) зменшує значення регістра на два $(SP = SP - 2)$, а потім розміщує операнд у двох сусідніх чарунках пам'яті стека, на які вказує регістр *SP*;

– читання слова зі стека: *POP op* ($op = r16, m16, sr$ (окрім *CS*)) – команда *POP* читає слово з вершини стека і переміщує його в операнд, а потім збільшує вміст регістра *SP* на два $(SP = SP + 2)$;

– запис регістра *Flags* у стек: *PUSHF*;

– читання регістра *Flags* із стека: *POPF*.

Більш сучасні мікропроцесори сімейства x86 мають ряд додаткових команд роботи зі стеком:

– запис у стек константи: *PUSH i16*;

– запис у стек усіх регістрів загального призначення: *PUSHA*;

– читання зі стека всіх регістрів загального призначення: *POPA*.

При написанні програми необхідно завжди резервувати як мінімум 128 байт під сегмент стека, навіть якщо програма сама його не використовує. Стек необхідний для операційної системи, під керуванням якої завжди працюють програми користувача.

При використанні команд пересилки даних може виникнути ситуація, коли асемблер не може визначити розмір операнда. Для явної вказівки типу операнда або його тимчасової зміни використовують оператор асемблера PTR, який має формат

<тип> PTR <операнд>, де <тип> – це BYTE, WORD або DWORD.

Наприклад, якщо задана змінна Y розміром в слово, а треба записати нуль тільки в його молодший байт, то необхідно використовувати оператор PTR:

```
Y DW 1234h;
MOV BYTE PTR Y, 0;
```

$Y = 1200h$.

Якщо цю команду написати без оператора PTR, то змінна Y стала б рівна нулю, оскільки нулі були записані в обидві чарунки пам'яті, у яких зберігається змінна *Y*:

```
Y DW 1234h;
MOV Y, 0;
```

$Y = 0000h$.

Для тестування програм, написаних на асемблері, необхідно використовувати спеціальні програми-емулятори, які дозволяють проглядати значення регістрів мікропроцесора та чарунок пам'яті під час виконання програми, а також виконувати програму в покроковому або автоматичному режимі з можливістю зупинки програми на будь-якій команді програми. Однією з таких програм є емулятор мікропроцесора 8086 «emu8086». Ця програма дозволяє проводити тестування файлів із розширенням «exe» або «com».

Після компіляції програми, емулятор викликається шляхом натиснення кнопки «emulate». Емулятор підтримує такі режими виконання програми, що тестується:

– автоматичний режим виконання програми – кнопка «run» або клавіша F9;

– покроковий режим роботи програми – кнопка «single step» або клавіша F8;

– повернення програми на один крок назад – кнопка «step back» або клавіша F6;

– покроковий режим роботи програми, окрім підпрограм, які виконуються в автоматичному режимі – Shift+F8 (debug \ step over).

– автоматичне виконання програми до команди, яка відзначена курсором (жовтий рядок) – Ctrl+F8 (debug \ run until);

– перезавантаження програми – кнопки «reload» та «reset» або клавіша F4.

Для окремого завантаження програми в емулятор необхідно натиснути кнопку «Load» або викликати меню: file \ load executable. Виділити необхідний файл і натиснути кнопку «Відкрити».

Для перегляду результатів роботи програми у вікні емулятора виводиться значення усіх регістрів мікропроцесора 8086 та код програми на асемблері. Додатково можна вивести на екран такі вікна:

– змінні, що оголошені в сегменті даних – кнопка «vars» або меню «view / variables». Значення змінних можна відображати у двійковій (bin), шістнадцятковій (hex), вісімковій (octal), десятковій (signed або unsigned) системах числення та в текстовому форматі (ASCII);

– вміст стека програми – кнопка «stack» або меню «view / stack»;

– значення регістра *Flags* – кнопка «flags» або меню «view / flags»;

– вікно емуляції екрана монітора – кнопка «screen» або меню «view / emulator screen»;

– значення чарунок пам'яті – кнопка «aux / memory» або меню «view / memory».

2. Порядок виконання роботи

1. За алгоритмом, який наведено на рис. 4.1, написати програму на асемблері мікропроцесора 8086, використовуючи шаблон для програми з розширенням EXE, що дозволить запускати програму під керуванням операційної системи Windows.

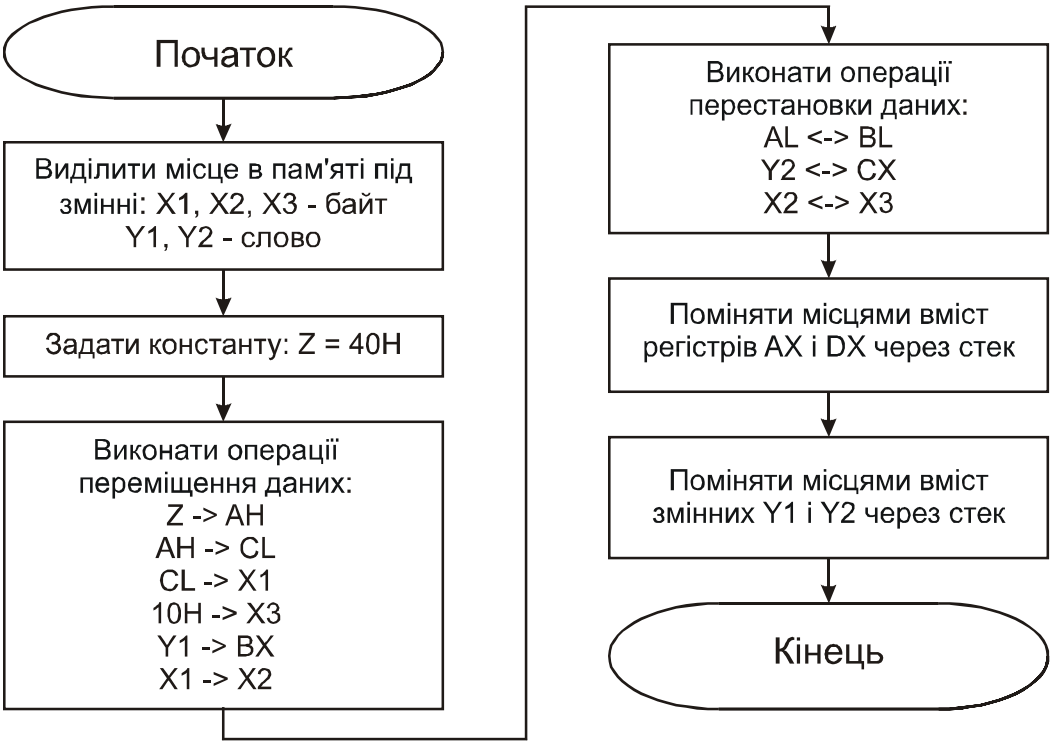


Рис. 4.1. Алгоритм для програми на асемблері мікропроцесора 8086

2. Запустити емулятор «emu8086» та набрати програму в текстовому редакторі й зберегти її під ім’ям Lr4.asm.

3. Виконати компіляцію програми та отримати файл Lr4.exe, що призначений до виконання.

4. Запустити емулятор (кнопка «emulate»). Вивести на екран додаткові вікна: вікно змінних (кнопка «vars») та вікно стека (кнопка «stack»). Виконати програму Lr4.exe в покроковому режимі (кнопка «single step»), спостерігаючи за змінами вмісту регістрів мікропроцесора, чарунок стека та значення змінних після виконання кожної команди. Коли програма досягне команди виходу із програми в Windows, записати у звіт значення регістрів мікропроцесора та значення змінних X1, X2, X3, Y1 та Y2.

3. Зміст звіту

1. Список команд переміщення даних та роботи зі стеком мікропроцесора 8086 з коротким описом.

2. Алгоритм і програма на асемблері з коментарями до кожної операції алгоритму.

3. Результати тестування програми: значення регістрів мікропроцесора, що використовуються в програмі, і значення змінних X1, X2, X3, Y1 та Y2.

4. Контрольні запитання

1. Призначення команди MOV.

2. Призначення команди XCHG.
3. Директиви для резервування місця в пам’яті під змінні.
4. Призначення директиви EQU.
5. Призначення оператора асемблера PTR.
6. Принцип зберігання даних у стеку.
7. Команди роботи із стеком мікропроцесора 8086.
8. Режими виконання програми, що підтримує емулятор «emu8086».

Лабораторні заняття 5

АРИФМЕТИЧНІ КОМАНДИ МІКРОПРОЦЕСОРА 8086

Мета роботи

Вивчити арифметичні команди мікропроцесора 8086 та навчитися використовувати їх при розробці програмного забезпечення мікропроцесорних пристроїв.

1. Короткі відомості з теорії

Мікропроцесор 8086 має такі команди додавання та віднімання [4]:

– додавання: ADD *op1, op2* $(op1 = op1 + op2)$;

– віднімання: SUB *op1, op2* $(op1 = op1 - op2)$;

– додавання з урахуванням біта переносу: ADC *op1, op2* $(op1 = op1 + op2 + CF)$;

– віднімання з урахуванням біта переносу: SBB *op1, op2* $(op1 = op1 - op2 - CF)$;

– інкремент (збільшення на одиницю): INC *op1* $(op1 = op1 + 1)$;

– декремент (зменшення на одиницю): DEC *op1* $(op1 = op1 - 1)$;

– зміна знака числа: NEG *op1* $(op1 = -op1)$.

Допустимі комбінації операндів вказаних команд наведені в табл. 5.1.

Таблиця 5.1 – Допустимі комбінації операндів у арифметичних командах

*op*1	*op*2
*r*8	*i*8, *r*8, *m*8
*m*8	*i*8, *r*8
*r*16	*i*16, *r*16, *m*16
*m*16	*i*16, *r*16

Особливості використання арифметичних команд додавання та віднімання мікропроцесора 8086:

– додавання чисел без знака здійснюється за модулем 2^n, де n – розмір елемента пам’яті або регістра мікропроцесора 8086, при цьому біт переносу

CF фіксує, чи було переповнення. Якщо при додаванні байт сума буде більше 255, то біт переносу буде встановлено *CF* = 1, інакше *CF* = 0. При додаванні слів біт переносу буде встановлено *CF* = 1, якщо сума буде більше 65535;

– віднімання чисел без знака також здійснюється за модулем 2^n, де n – розмір елемента пам'яті або регістра мікропроцесора, при цьому біт переносу *CF* фіксує, чи була позика. Якщо при відніманні результат виявиться менше нуля, то біт *CF* = 1, інакше *CF* = 0;

– знакові числа в мікропроцесорі 8086 при додаванні або відніманні повинні бути представлені в додатковому коді;

– якщо при додаванні знакових чисел, сума вийде більше 127 (для байтів) або 32767 (для слів), то встановлюється біт переповнення *OF* = 1;

– якщо при відніманні знакових чисел, результат вийде менше –128 (для байта) або –32768 (для слова), то біт *OF* = 1;

– додавання та віднімання знакових чисел та чисел без знака в мікропроцесорі 8086 здійснюється за одними і тими самими алгоритмами, тому мікропроцесор при виконанні цих операцій одночасно фіксує результат у бітах переносу *CF* та переповнення *OF*;

– якщо при додаванні або відніманні чисел результат рівний нулю, то мікропроцесор встановлює біт нуля *ZF* = 1;

– після виконання операцій додавання або віднімання старший біт результату завжди заноситься у біт знака *SF*.

Множення та ділення знакових чисел та чисел без знака виконується за різними алгоритмами, тому мікропроцесор 8086 має у своєму складі два типи команд [4]:

– множення чисел без знака: MUL *op*;

– множення знакових чисел: IMUL *op*;

– ділення чисел без знака: DIV *op*;

– ділення чисел із знаком: IDIV *op*.

При множенні чисел допустимі такі комбінації операндів:

– для байта – $AX = AL \times op \;\; (op = r8,\, m8)$;

– для слів – $DX, AX = AX \times op \;\; (op = r16,\, m16)$.

Як видно з комбінації операндів, при множенні результат виходить у подвоєному форматі, тобто якщо, наприклад, ми працювали з байтами, то після множення доводиться переходити на обробку слів, що не завжди зручно. Для того щоб визначити, обробляти подвійний формат чи буде достатньо й одинарного формату, треба після множення проаналізувати біти *CF* і *OF*:

– $OF = CF = 1$ – результат займає подвійний формат;

– $OF = CF = 0$ – результат займає одинарний формат, тобто старші розряди рівні нулю.

Особливістю команд множення мікропроцесора 8086 є те, що в нього відсутні команди множення та ділення на константу.

При діленні чисел допустимі такі комбінації операндів:

– ділення слова на байт: $AL = AX \text{ div } op \;(op = r8, m8)$, $AH = AX \bmod op$, де

div – ціла частина від ділення, mod – залишок від ділення;

– ділення подвійного слова на слово: $DX = (DX, AX) \bmod op$, $AX = (DX, AX) \operatorname{div} op \; (op = r16, m16)$.

При виконанні арифметичних операцій, особливо команд множення та ділення, часто виникає необхідність у зміні розміру числа. Для цього до складу команд мікропроцесора 8086 ввели дві команди:

– розширення байта до слова: CBW $(AL \to AX)$;

– розширення слова до подвійного слова: CWD $(AX \to DX, AX)$.

Зразок програми, що обчислює наступну математичну функцію

$$Y = X_1 X_3 - \left(4X_2 + \frac{X_3}{X_1}\right) - \frac{X_2^2}{X_1}$$

```
; Лабораторна робота № 5
include 'emu8086.inc'; підключення загальних функцій емулятора 8086
;_____________________________________
; сегмент даних програми
DATA SEGMENT
pkey db 13,10,"Для виходу із програми натисніть будь–яку клавішу$"
X1 DW 4
X2 DW 8
X3 DW 16
x4 DW ?; зміна для зберігання проміжних даних
Y DW 0
ENDS
;_____________________________________
; сегмент стека програми
STACK SEGMENT
    DW 128 DUP(0)
ENDS
;_____________________________________
; сегмент коду програми
CODE SEGMENT
START:
; налаштування сегментних регістрів
    MOV AX, DATA
    MOV DS, AX
    MOV ES, AX
;_____________________________
; обчислення математичної функції
    MOV AX, 4
    MUL X2
    MOV X4, AX; обчислення 4*X2
```

```
    MOV AX, X3
    CWD
    DIV X1; обчислення X3/X1
    ADD X4, AX; результат обчислення у дужках
    MOV AX, X2
    MUL X2; обчислення X2^2
    CWD
    DIV X1; обчислення X2^2/X1
    MOV BX, AX
    MOV AX, X1
    MUL X3; обчислення X1*X3
    SUB AX, X4; обчислення X1*X3–ДУЖКА
    SUB AX, BX
    MOV Y, AX; зберігання результату
; вивід результату на екран
    PRINT 'Y='
    MOV AX, Y
    CALL PRINT_NUM_UNS; вивід без знакових чисел із регістра AX
; для виводу знакових чисел використовується функція PRINT_NUM
;_____________________________
; вихід із програми у Windows
; вивід рядка на екран
    LEA DX, PKEY
    MOV AH, 9
    INT 21H
; очікування натиснення клавіші
    MOV AH, 1
    INT 21H
; повернення в операційну систему
    MOV AX, 4C00H
    INT 21H
ENDS
DEFINE_PRINT_NUM_UNS; – для беззнакових чисел
; DEFINE_PRINT_NUM – для знакових чисел
end start; місце входу в програму
```

2. Порядок виконання роботи

1. За заданим варіантом (табл. 5.2) скласти програму на асемблері мікропроцесора 8086, яка обчислює задану математичну функцію. Зразок програми наведено вище.

Таблиця 5.2 – Індивідуальні завдання до лабораторної роботи

Номер варіанта	Тип і розмір змінних	Математична функція
1	2	3
1	Беззнаковий байт	$Y = 6(X_1 + X_2) + X_3^2\left(\frac{X_2}{X_1} - X_1\right)$
2	Знаковий байт	$Y = \left(X_3 - X_1^2\right)\cdot 12 + \left(\frac{X_2}{X_1} + X_3\right)\cdot X_2$
3	Беззнакове слово	$Y = \frac{X_1^2}{2} + X_3 - \left(10X_2 - X_1\right)$
4	Знакове слово	$Y = X_1 + X_2 + X_3\left(\frac{X_3}{X_1X_2} - 16X_1^3\right)$
5	Беззнаковий байт	$Y = \left(X_1 + \frac{X_2}{X_3}\right)^2 + 6(X_2 - X_3)$
6	Знаковий байт	$Y = X_2X_1^2 + \left(X_3 - \frac{X_1}{X_2}\right)\cdot 7$
7	Беззнакове слово	$Y = X_1X_2X_3 + \frac{3\left(X_1 + X_2^2 + X_3\right)}{X_1 - X_3}$
8	Знакове слово	$Y = 5X_1^2 - \frac{X_1}{X_2} + 5X_3$
9	Беззнаковий байт	$Y = 2\left(X_3 + X_1X_2^2\right) - X_1\left(\frac{X_1}{X_3} - 2\right)$
10	Знаковий байт	$Y = \frac{X_1^2 + 2X_2X_3 - X_1}{X_1}$
11	Беззнакове слово	$Y = (X_1X_3)^2 - \left(X_2 + \frac{X_3}{X_1}\right)\cdot 20$
12	Знакове слово	$Y = X_2^3 - X_1 + 12X_3 - \left(\frac{X_1}{X_2} + 9\right)$
13	Беззнаковий байт	$Y = X_1 + \frac{X_2}{X_3}\left(X_2^3 - 6X_1\right)$
14	Знаковий байт	$Y = X_3 - X_1\left(7X_2^2 + X_1\right) + \frac{X_3}{X_1}$
15	Беззнакове слово	$Y = X_1X_2^2 - (2X_1 - X_3) + \frac{X_3}{X_2}$

Примітка. Усі змінні повинні зберігатися в ОЗП у сегменті даних. Результат математичної функції також необхідно зберегти в ОЗП у сегменті даних.

2. Набрати свою програму в текстовому редакторі та відкомпілювати за допомогою емулятора мікропроцесора 8086 «emu8086».

3. Провести тестування роботи програми в покроковому режимі.

4. Підставити в програму реальні значення змінних і перевірити відповідність роботи програми заданому алгоритму. Для спрощення тестування роботи програми необхідно при виборі вихідних даних підібрати такі значення *X*1, *X*2 та *X*3, щоб при діленні остаток від ділення був рівний нулю, а результат *Y* був більше нуля. Результат обчислення заданої математичної функції записати у звіт.

3. Зміст звіту

1. Список арифметичних команд мікропроцесора 8086 з коротким описом.
2. Програма на асемблері мікропроцесора 8086 з коментарями до кожної операції математичної функції.
3. Результат обчислення програми.
4. Відповіді на контрольні запитання (за завданням викладача).

4. Контрольні запитання

1. Призначення біта переносу *CF* для команд додавання та віднімання.
2. Призначення біта переповнення *OF* для команд додавання та віднімання.
3. Спосіб представлення знакових чисел у мікропроцесорі 8086.
4. У яких випадках після виконання арифметичних команд установлюється біт нуля *ZF*?
5. Чому буде рівний біт знака *SF* після виконання арифметичних операцій?
6. Діапазон знакових чисел розміром у байт.
7. Діапазон знакових чисел розміром у слово.
8. Особливості команд множення в мікропроцесорі 8086.
9. Призначення бітів *OF* і *CF* у командах множення мікропроцесора 8086.
10. Як здійснюється ділення цілих чисел у мікропроцесорі 8086?

Лабораторні заняття 6

ЛОГІЧНІ КОМАНДИ Й КОМАНДИ ЗСУВУ МІКРОПРОЦЕСОРА 8086

Мета роботи

Вивчити логічні команди й команди зсуву мікропроцесора 8086, а також навчитися використовувати їх у розробці програмного забезпечення мікропроцесорних пристроїв.

1. Короткі відомості з теорії

Система команд мікропроцесора 8086 дозволяє виконувати такі логічні операції [4]:

– заперечення: NOT op1 ($op1 = \text{not } op1$);

– логічне множення: AND op1, op2 ($op1 = op1 \text{ and } op2$);

– логічне додавання: OR op1, op2 ($op1 = op1 \text{ or } op2$);

– додавання за модулем два: XOR op1, op2 ($op1 = op1 \text{ xor } op2$);

– перевірка: TEST op1, op2 ($op1 \text{ and } op2$) – дана команда є аналогом логічного множення AND, але результат операції нікуди не записується, а тільки встановлює біти умов.

Допустимі комбінації операндів вказаних команд наведені в табл. 6.1.

Таблиця 6.1 – Допустимі комбінації операндів у логічних командах

*op*1	*op*2
*r*8	*i*8, *r*8, *m*8
*m*8	*i*8, *r*8
*r*16	*i*16, *r*16, *m*16
*m*16	*i*16, *r*16

Логічні команди мікропроцесора 8086 використовуються для роботи з окремими бітами операндів. Логічні команди міняють всі біти умов, але на практиці для аналізу використовують тільки біт нуля *ZF*, оскільки решта бітів для бітових операцій не несе ніякої інформації.

Команда **логічного множення AND** часто використовується для встановлення значення окремих бітів операнда *op*1 рівним нулю шляхом накладення на нього маски (*op*2), у якій біти, які треба встановити в нуль, також рівні нулю, а біти, які не потрібно змінювати, повинні бути рівні одиниці.

Наприклад, якщо необхідно записати нулі в біти з номерами 0, 2 та 6 регістра *AL*, то потрібно виконати команду:

AND *AL*, 10111010*b*.

Команду **логічного додавання *OR*** можна використовувати для установки окремих розрядів операнда *op*1 в одиницю. Маска для такої операції формується за таким правилом: біти, які необхідно встановити в одиницю, повинні бути також рівні одиниці, а біти, які не потрібно змінювати, повинні бути рівні нулю. Наприклад, необхідно встановити в одиницю біти з номерами 1, 3 та 7 регістра *BH*. Для цього виконаємо таку команду:

OR *BH*, 10001010*b*.

Логічна команда **додавання по модулю два XOR** використовується для інвертування окремих розрядів операнда *op*1. Маска складається за таким правилом: біти, які необхідно інвертувати, повинні бути рівні одиниці, а біти, які потрібно залишити без змін, повинні бути рівні нулю.

Наприклад, якщо необхідно інвертувати біти з номерами 2, 4 та 6 регістра *CH*, то для цього треба виконати таку команду:

XOR *CH*, 01010100*b*.

Мікропроцесор 8086 має такі **команди зсуву** [4]:

– логічний зсув уліво: SHL *op*1, *op*2 (рис. 6.1, а);

– логічний зсув управо: SHR *op*1, *op*2 (рис. 6.1, б);

– арифметичний зсув уліво: SAL *op*1, *op*2 (рис. 6.1, в);

– арифметичний зсув управо: SAR *op*1, *op*2 (рис. 6.1, г);

– циклічний зсув уліво: ROL *op*1, *op*2 (рис. 6.1, д);

– циклічний зсув управо: ROR *op*1, *op*2 (рис. 6.1, е);

– циклічний зсув уліво через біт переносу *CF*: RCL *op*1, *op*2 (рис. 6.1, ж);

– циклічний зсув управо через біт CF: RCR op1, op2 (рис. 6.1, з).

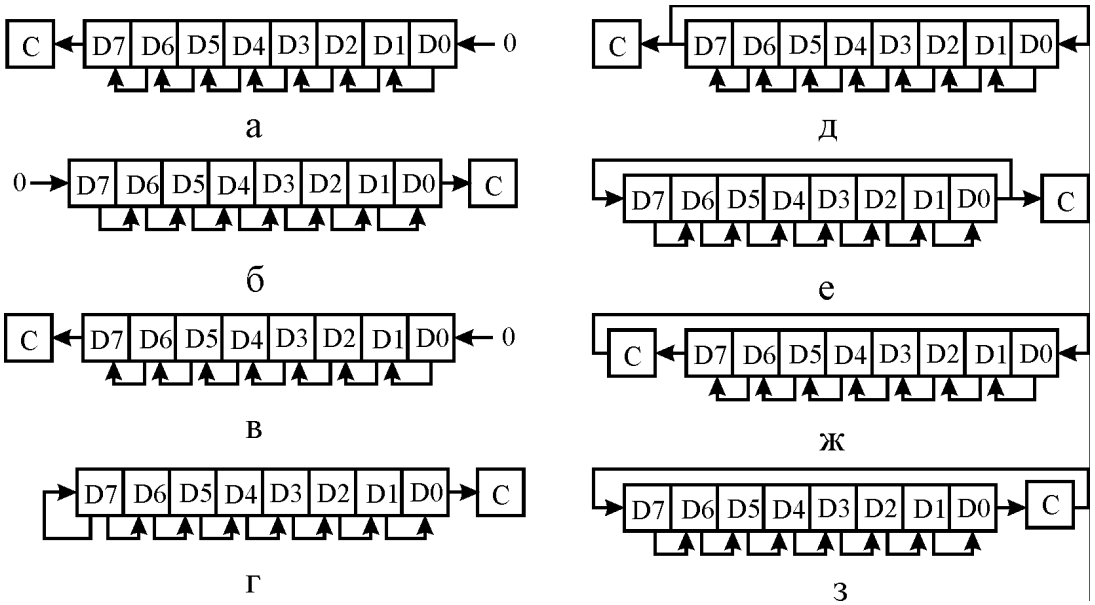


Рис. 6.1. Графічне пояснення до команд зсуву: а – логічний зсув уліво; б – логічний зсув управо; в – арифметичний зсув уліво; г – арифметичний зсув управо; д – циклічний зсув уліво; е – циклічний зсув управо; ж – циклічний зсув уліво через біт переносу CF; з – циклічний зсув управо через біт переносу CF

Перший операнд команд зсуву є набором біт, які необхідно зсунути в певну сторону. Як перший операнд можна використовувати: *r*8, *m*8, *r*16 та *m*16. Другий операнд визначає на скільки розрядів необхідно зсунути біти першого операнда. Якщо зсув необхідно провести на один розряд, то в команді на місці другого операнда ставиться «1». Якщо необхідно зсунути на *n* розрядів, то в команді на місці другого операнда указується ім'я регістра *CL*, а число *n* повинне бути спочатку записано в цей регістр. При використанні в команді зсуву регістра *CL* його вміст не змінюється. Наприклад, для виконання операції зсуву треба використовувати такі команди:

– SHL AX, 1; – логічний зсув регістра *AX* на один розряд уліво;

– MOV CL, 3; – запис у регістр *CL* константи «3»;

– SHR BX, CL; – логічний зсув регістра *BX* на три розряди вправо.

Команди логічного зсуву часто використовують для швидкого множення (зсув уліво) або ділення (зсув управо) цілих беззнакових чисел на число 2^n,

де n – число зсувів. Наприклад, якщо в регістрі *AL* було записано число вісім (00001000*b*), то після логічного зсуву вліво на один розряд, вміст регістра *AL* стане рівним 16 (00010000*b*), що відповідає операції множення на два. Якщо ж вміст регістра *AL* зсунути вправо на два розряди, то це буде відповідно, ділення на чотири – $AL = 2$ (00000010*b*).

Команди арифметичного зсуву призначені для швидкого множення або ділення знакових цілих чисел на число 2^n, оскільки логічний зсув дає помилку при діленні від'ємних чисел.

Команди циклічного зсуву використовують для перестановки місцями частин вмісту регістра або чарунки пам'яті. Наприклад, якщо потрібно поміняти місцями старші та молодші біти регістра *AH*, то потрібно застосувати такі команди:

– MOV CL, 4; $CL = 4$;

– ROL AH, CL; циклічний зсув вмісту регістра *AH* на чотири розряди.

Використання логічних команд і команд зсуву, що працюють з окремими бітами операндів, дозволяє упаковувати декілька змінних в одну чарунку пам'яті, і як наслідок, більш економно використовувати пам'ять мікропроцесорних систем при написанні програми на асемблері мікропроцесора 8086.

2. Порядок виконання роботи

1. За заданим варіантом (табл. 6.2) скласти програму на асемблері мікропроцесора 8086, використовуючи зразок програми з лабораторної роботи 5.

Таблиця 6.2 – Індивідуальні завдання до лабораторної роботи

Номер варіанта	Тип і розмір змінних	Задача
1	2	3
1	Беззнакове слово	Обчислити: $Y1 = (X1 \text{ and } X3 \text{ xor } X2) \text{ or } X1 \text{ or not } X2$; *Y*2=*Y*1, а потім інвертувати 0, 3 та 7 біти змінної *Y*2
2	Беззнаковий байт	Обчислити: $Y1 = (X3 \text{ and } X2) \text{ or } (\text{not } X2 \text{ xor } X1) \text{ or } X1$; *Y*2=*Y*1, а потім прирівняти нулю 1, 4 та 6 біти змінної *Y*2
3	Знакове слово	Обчислити: $Y1 = (X2 \text{ or not } X3) \text{ and } (\text{not } X1 \text{ and } X3) \text{ xor } X1$; *Y*2=*Y*1, а потім установити в одиницю 2, 5 та 7 біти змінної *Y*2
4	Знаковий байт	Обчислити: $Y1 = X3 \text{ or } X1 \text{ and } (X1 \text{ and } X2 \text{ xor not } X3)$; $Y2 = Y1 \cdot 16$
5	Беззнакове слово	Обчислити: $Y1 = X1 \text{ and } X2 \text{ or } (\text{not } X1 \text{ and } X2 \text{ xor } X3)$; $Y2 = Y1 \cdot 4$

Закінчення табл. 6.2

1	2	3
6	Беззнаковий байт	Обчислити: $Y1 = (X1 \text{ xor } X3) \text{ or } X2 \text{ and } (X1 \text{ or } X2 \text{ xor } X3)$; $Y2 = Y1/8$
7	Знакове слово	Обчислити: $Y1 = X1 \text{ and } X2 \text{ and } (X3 \text{ or not } X1) \text{ xor } X2$; $Y2 = Y1 \cdot 8$
8	Знаковий байт	Обчислити: $Y1 = \text{not } X3 \text{ and } X1 \text{ or not } X2 \text{ xor } (X1 \text{ and } X2 \text{ or } X3)$; $Y2 = Y1/2$
9	Беззнакове слово	Обчислити: $Y1 = (X2 \text{ and } X1) \text{ or not } X3 \text{ xor } X1 \text{ and } (X2 \text{ or } X3)$;ї $Y2 = Y1 \cdot 2$
10	Знаковий байт	Обчислити: $Y1 = \text{not } X1 \text{ or } X2 \text{ xor}(X2 \text{ and } X1 \text{ xor } X3) \text{ or } X1$; $Y2 = Y1/4$
11	Беззнакове слово	Обчислити: $Y1 = X3 \text{ or } X1 \text{ xor } (\text{not } X2 \text{ and } X1 \text{ or not } X3) \text{ and } X1$; $Y2 = Y1 \cdot 8$
12	Знаковий байт	Обчислити: $Y1 = X1 \text{ or } X2 \text{ or not } X3 \text{ and } (X1 \text{ xor } X2 \text{ xor } X3)$; $Y2 = Y1/8$
13	Беззнакове слово	Обчислити: $Y1 = (X3 \text{ and } X2 \text{ or } X1) \text{xor not } X1 \text{ or not } X2$; $Y2=Y1$, а потім інвертувати 1, 2 та 6 біти змінної $Y2$
14	Беззнаковий байт	Обчислити: $Y1 = (X1 \text{ and } X2) \text{ or } (X3 \text{ xor not } X1) \text{ and } X2$; $Y2=Y1$, а потім прирівняти нулю 3, 4 та 5 біти змінної $Y2$
15	Беззнакове слово	Обчислити: $Y1 = (\text{not } X2 \text{ or } X1) \text{ and } (X1 \text{ and not } X3) \text{ xor } X2$; $Y2=Y1$, а потім установити в одиницю 0, 1 та 7 біти змінної $Y2$

Примітка. Усі змінні повинні зберігатися в ОЗП у сегменті даних. Результати логічних функції також необхідно зберегти в ОЗП у сегменті даних.

2. Набрати свою програму в текстовому редакторі та відкомпілювати за допомогою емулятора мікропроцесора 8086 «emu8086».

3. Провести тестування роботи програми в покроковому режимі.

4. Підставити в програму реальні значення змінних і перевірити відповідність роботи програми до заданої логічної функції. Результат обчислення заданої логічної функції записати у звіт.

3. Зміст звіту

1. Список логічних команд і команд зсуву з коротким описом.
2. Програма на асемблері мікропроцесора 8086 з коментарями до кожної операції логічної функції.
3. Результати обчислення програми.
4. Відповіді на контрольні запитання (за завданням викладача).

4. Контрольні запитання

1. Для чого в систему команд мікропроцесора 8086 включені логічні команди та команди зсуву?
2. Для чого можна використовувати команду логічного множення *AND*?
3. Для чого можна використовувати команду логічного додавання *OR*?
4. Для чого можна використовувати логічну команду *XOR*?
5. Основне призначення команд логічного зсуву.
6. Основне призначення команд арифметичного зсуву.
7. Основне призначення команд циклічного зсуву.
8. Яким чином у мікропроцесорі 8086 здійснюється зсув на *n* розрядів?

Лабораторні заняття 7

КОМАНДИ ПОРІВНЯННЯ, УМОВНИХ ПЕРЕХОДІВ ТА ОРГАНІЗАЦІЇ ЦИКЛІВ МІКРОПРОЦЕСОРА 8086

Мета роботи

Вивчити команди порівняння, умовних переходів та організації циклів мікропроцесора 8086, а також навчитися використовувати їх для реалізації умовного розгалуження, операторів вибору та організації детермінованих циклів типу «FOR», а також циклів з умовами «WHILE» та «DO WHILE» на асемблері мікропроцесора 8086.

1. Короткі відомості з теорії

Для реалізації алгоритмів з розгалуженням або циклічними повтореннями до складу команд мікропроцесора 8086 включена спеціальна група команд, яка дозволяє реалізовувати операції порівняння, безумовного та умовного переходів, а також організації циклів. Усі команди переходів у мікропроцесорі 8086 є відносними, тобто в них указується різниця між адресою команди, на яку здійснюється перехід, і адресою команди, яка йде відразу за командою переходу. Використання відносних переходів дозволяє розміщувати програму в будь–якій області пам’яті мікропроцесорної системи. Відносний перехід записується у вигляді числа із знаком у додатковому коді ($-128...+127$ або $-32768...+32767$), що дозволяє здійснювати переходи як

уперед, так і назад. Обчислення відносних переходів уручну є досить трудомісткою задачею. Тому при програмуванні на асемблері необхідно користуватися тільки мітками, а задача обчислення відносного переходу покладається на компілятор.

Безумовний перехід виконується завжди без будь-яких умов і призначений для зміни послідовного порядку виконання команд. Мікропроцесор 8086 має два типи команд безумовного переходу [4]:

– **прямий перехід**: JMP <мітка>, де мітка вказує на команду, якій треба передати керування;

– **непрямий перехід**: JMP *op*, де $op = r16$ або $m16$. У цій команді адреса переходу береться з указаного регістра або елемента пам'яті. Непрямий перехід використовується, коли адреса переходу наперед невідома, а обчислюється в ході виконання програми.

Умовний перехід здійснюється тільки в разі виконання якої-небудь умови. Досить часто перед командою умовного переходу виконується операція порівняння яких-небудь двох величин.

Команда порівняння двох операндів: CMP *op*1, *op*2 (*op*1 – *op*2). [4]

Дана команда аналогічна команді віднімання SUB за винятком того, що результат операції нікуди не записується, зате встановлюються всі біти умов. Допустимі комбінації операндів команди порівняння наведені в табл. 7.1.

Таблиця 7.1 – Допустимі комбінації операндів у команді порівняння CMP

*op*1	*op*2
*r*8	*i*8, *r*8, *m*8
*m*8	*i*8, *r*8
*r*16	*i*16, *r*16, *m*16
*m*16	*i*16, *r*16

Мікропроцесор 8086 має три групи команд умовного переходу:

– команди умовного переходу, які розміщуються відразу після команди порівняння CMP (табл. 7.2);

– команди умовного переходу за станом певного біта регістра *Flags* (табл. 7.2);

– команди умовного переходу за значенням регістра CX (табл. 7.2).

Недоліком усіх команд умовного переходу є те, що всі вони здійснюють тільки короткий відносний перехід у межах $-128...+127$ (це приблизно 30 – 40 команд). Якщо адреса відносного переходу виходитиме за цей діапазон, то асемблер при компіляції зафіксує помилку.

Мікропроцесор 8086 має команди, що призначені для реалізації детермінованих циклів типу «FOR» [4]:

– LOOP <мітка> ($CX = CX - 1$, якщо $CX \neq 0$, то перехід на мітку);

– LOOPE/LOOPZ <мітка> ($CX = CX - 1$, якщо ($CX \neq 0$ and $ZF = 1$), то перехід на мітку);

– LOOPNE/LOOPNZ <мітка> ($CX = CX - 1$, якщо ($CX \neq 0$ and $ZF = 0$), то

перехід на мітку).

Таблиця 7.2 – Список команд умовного переходу мікропроцесора 8086

	Команда	Умова для переходу
Група 1	JE <мітка>	$op1 = op2$ (E – equal (дорівнює)) – для усіх чисел
	JNE <мітка>	$op1 <> op2$ (N – not (не дорівнює)) – для усіх чисел
	JL/JNGE <мітка>	$op1 < op2$ (L – less (менше)) – для чисел зі знаком
	JLE/JNG <мітка>	$op1 <= op2$ – для чисел зі знаком
	JG/JNLE <мітка>	$op1 > op2$ (G – greater (більше)) – для чисел зі знаком
	JGE/JNL <мітка>	$op1 >= op2$ – для чисел зі знаком
	JB/JNAE <мітка>	$op1 < op2$ (B – below (менше)) – для чисел без знака
	JBE/JNA <мітка>	$op1 <= op2$ – для чисел без знака
	JA/JNBE <мітка>	$op1 > op2$ (A – above (більше)) – для чисел без знака
	JAE/JNB <мітка>	$op1 >= op2$ – для чисел без знака
Група 2	JZ <мітка>	$ZF = 1$ – біт нуля
	JNZ <мітка>	$ZF = 0$ – біт нуля
	JC <мітка>	$CF = 1$ – біт переносу
	JNC <мітка>	$CF = 0$ – біт переносу
	JS <мітка>	$SF = 1$ – біт знака
	JNS <мітка>	$SF = 0$ – біт знака
	JO <мітка>	$OF = 1$ – біт переповнення
	JNO <мітка>	$OF = 0$ – біт переповнення
	JP <мітка>	$PF = 1$ – біт парності (паритету)
	JNP <мітка>	$PF = 0$ – біт парності (паритету)
Група 3	JCXZ <мітка>	Перехід, якщо вміст регістра $CX = 0$

Як видно з опису команд, на асемблері мікропроцесора 8086 в якості лічильника циклів використовується 16-розрядний регістр CX. Таким чином, бажано регістр CX не використовувати в тілі циклу. Якщо ж усе-таки регістр CX необхідно використовувати в тілі циклу, то його вміст обов'язково потрібно зберегти в якій-небудь вільній чарунці пам'яті, а в кінці циклу відновити звідти.

Команда **LOOP** мікропроцесора 8086 фактично замінює собою три окремі команди:

– декремент регістра CX (DEC CX);

– перевірка вмісту регістра CX на нуль (CMP CX, 0);

– умовний перехід на мітку, якщо біт нуля *ZF* = 0 (JNZ <мітка>).

Команди **LOOPZ/LOOPE** і **LOOPNZ/LOOPNE** використовуються для реалізації детермінованих циклів, у яких необхідно передбачити можливість дострокового виходу з циклу. Ці команди, окрім перевірки регістра *CX* на нуль, також аналізують стан біта нуля *ZF*. Звичайно перед цими командами розміщується команда, результат якої змінює стан біта нуля *ZF*. Команда LOOPZ/LOOPE змушує цикл повторюватися, поки вміст регістра *CX* не стане рівним нулю, але тільки за умови, якщо попередня команда фіксуватиме ну-

льовий результат, тобто біт *ZF* =1. Якщо хоча б в одному проході циклу буде зафіксований ненульовий результат (біт *ZF* =0), то відбувається достроковий вихід із циклу. Команда LOOPNZ/LOOPNE аналогічна команді LOOPZ/LOOPE, але достроковий вихід із циклу здійснюється, якщо попередня команда зафіксує нульовий результат, тобто біт *ZF* =1. Назви команд LOOPZ і LOOPE, а також LOOPNZ і LOOPNE є синонімами. Команди дострокового виходу з циклу найчастіше використовують для пошуку першого елемента в *N*-послідовності, що задовольняє певну умову.

Приклад реалізації на асемблері мікропроцесора 8086 оператора умовного розгалуження типу IF <умова> *op1* ELSE *op2*. Наприклад, реалізуємо на асемблері мікропроцесора 8086 алгоритм, наведений на рис. 7.1 (*X, Y* і *Z* – числа зі знаком розміром у байт).

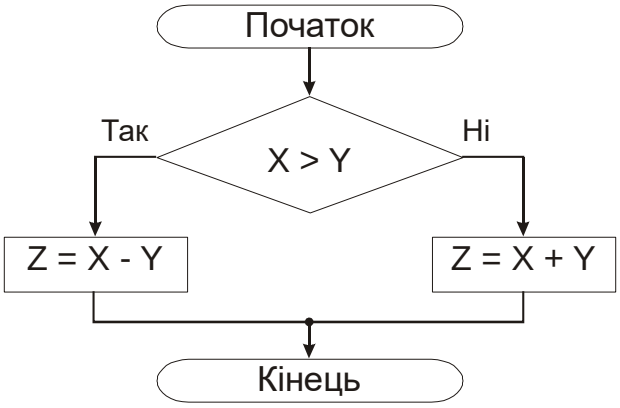


Рис. 7.1. Алгоритм умовного розгалуження

```
; Лабораторна робота № 7.1
include 'emu8086.inc'; підключення загальних функцій емулятора 8086
;————————————————————————
; сегмент даних програми
DATA SEGMENT
PKEY DB 13,10,"Для виходу із програми натисніть будь–яку клавішу$"
X DB ?; резервування елемента пам’яті під змінну X
Y DB ?; резервування елемента пам’яті під змінну Y
Z DB ?; резервування елемента пам’яті під змінну Z
ENDS
;————————————————————————
; сегмент стека програми
STACK SEGMENT
    DW 128 DUP(0)
ENDS
;————————————————————————
; сегмент коду програми
CODE SEGMENT
```

```
START:
; налаштування сегментних регістрів
    MOV AX, DATA
    MOV DS, AX
    MOV ES, AX
;_____________________________
; основна програма
    MOV AH, X; запис числа X у регістр AH
    CMP AH, Y    ; порівняння змінних X і Y
    JG M1; перехід на мітку M1, якщо  X > Y
    ADD AH, Y; обчислити  AH = AH(X) + Y
    JMP M2; безумовний перехід на мітку M2
M1: SUB AH, Y;  обчислити  AH = AH(X) – Y
M2: MOV Z, AH; помістити результат із регістра AH у змінну Z
; вивід результату на екран
    PRINT 'Z='
    MOV AL, Z
    MOV AH, 0
    CALL PRINT_NUM; вивід знакових чисел із регістра AX
;_____________________________
; вихід із програми у Windows
; вивід рядка на екран
    LEA DX, PKEY
    MOV AH, 9
    INT 21H
; очікування натиснення клавіші
    MOV AH, 1
    INT 21H
; повернення в операційну систему
    MOV AX, 4C00H
    INT 21H
ENDS
DEFINE_PRINT_NUM; – для знакових чисел
END START; місце входу в програму
```

Приклад реалізації на асемблері мікропроцесора 8086 оператора вибору типу SWITCH:

```
 SWITCH <ім’я змінної>
     case 0: op1; break;
     case 1: op2; break;
     case 2: op3; break;
     default: op4; break;
```

Для прикладу реалізуємо на асемблері мікропроцесора 8086 алгоритм, наведений на рис. 7.2 (*X* і *Y* – слова без знака).

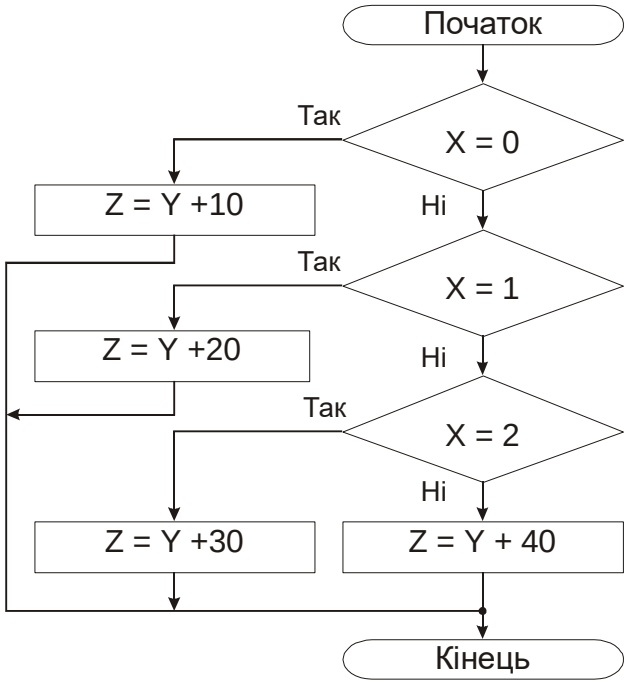


Рис. 7.2. Алгоритм оператора вибору

```
; Лабораторна робота № 7.2
include 'emu8086.inc'; підключення загальних функцій емулятора 8086
;_____________________________________
; сегмент даних програми
DATA SEGMENT
PKEY DB 13,10,"Для виходу із програми натисніть будь-яку клавішу$"
X DW ?; резервування пам’яті під змінну X
Y DW ?; резервування пам’яті під змінну Y
Z DW ?; резервування пам’яті під змінну Z
ENDS
;_____________________________________
; сегмент стека програми
STACK SEGMENT
    DW 128 DUP(0)
ENDS
;_____________________________________
; сегмент коду програми
CODE SEGMENT
START:
; налаштування сегментних регістрів
    MOV AX, DATA
    MOV DS, AX
    MOV ES, AX
;_____________________________
; основна програма
        MOV AX, Y; записати змінну Y у регістр AX
```

```
    CMP X, 0; порівняння числа X з константою, рівною нулю
    JZ M1; перехід на мітку M1, якщо X = 0
    CMP X, 1; порівняння X з константою «1»
    JE M2; перехід на мітку M2, якщо X = 1
    CMP X, 2; порівняння X з константою «2»
    JE M3; перехід на мітку M3, якщо X = 2
    ADD AX, 40; обчислити AX = AX(Y) + 40
    JMP M4; безумовний перехід на мітку M4
M1: ADD AX, 10; обчислити AX = AX(Y) + 10
    JMP M4; безумовний перехід на мітку M4
M2: ADD AX, 20; обчислити AX = AX(Y) + 20
    JMP M4; безумовний перехід на мітку M4
M3: ADD AX, 30; обчислити AX = AX(Y) + 30
M4: MOV Z, AX; помістити результат із регістра AX у змінну Z
; вивід результату на екран
    PRINT 'Z='
    MOV AX, Z
    CALL PRINT_NUM_UNS; вивід без знакових чисел із регістра AX
;_____________________________
; вихід із програми у Windows
; вивід рядка на екран
    LEA DX, PKEY
    MOV AH, 9
    INT 21H
; очікування натиснення клавіші
    MOV AH, 1
    INT 21H
; повернення в операційну систему
    MOV AX, 4C00H
    INT 21H
ENDS
DEFINE_PRINT_NUM_UNS; – для беззнакових чисел
END START; місце входу в програму
```

Приклад реалізації на асемблері мікропроцесора 8086 циклу типу «FOR». Розглянемо використання команди LOOP для реалізації детермінованого циклу на такому прикладі. Необхідно вміст регістра *AX* додавати до попереднього значення цього регістра *N*–число разів, де *N* – змінна типу байт (0…255).

```
N DB ?; резервування пам'яті в сегменті даних під змінну N
    MOV CL, N; запис змінної N у регістр CL
    MOV CH, 0; CH = 0, оскільки для лічильника використовується регістр CX
    JCXZ M2; вихід із циклу, якщо N = 0
M1: ADD AX, AX; AX = AX + AX
```

LOOP M1; кінець циклу
M2: .....; продовження програми

Приклади реалізації на асемблері мікропроцесора 8086 операторів організації циклів типу «WHILE» та «DO WHILE». Для реалізації циклів із передумовою та умовою в кінці в мікропроцесорах 8086 немає спеціальних команд, але їх можна реалізувати за допомогою команд переходів.

Цикл із передумовою має такий алгоритм роботи:

WHILE <умова>
<оператор>

Оператор циклу виконується, поки виконується умова. Вихід із циклу здійснюється, якщо умова дає результат FALSE.

Реалізуємо на асемблері мікропроцесора 8086 такий цикл із передумовою: WHILE $X > 0$ DO *op*1 (де X – байт без знака).

X DB ?; резервуємо місце в сегменті даних під змінну X
M2: CMP X, 0; порівняння змінної X з нулем
JBE M1; вихід із циклу, якщо $X <= 0$
*op*1; тіло циклу
JMP M2; повернення на початок циклу
M1: ....; продовження програми

Цикл з умовою у кінці має такий алгоритм роботи:

DO
<оператор>
WHILE <умова>

Цикл повторюється, поки виконується умова, таким чином цикл буде обов'язково виконаний хоча б один раз. Як тільки перевірка умови дасть результат FALSE, то відбувається вихід із циклу.

Реалізуємо на асемблері мікропроцесора 8086 такий цикл з умовою у кінці: DO *op*1 WHILE $X = 0$ (де X – знакове слово).

X DW ?; резервуємо місце в сегменті даних під змінну X
M1: *op*1; тіло циклу
CMP X, 0; порівняння змінної X з нулем
JE M1; перехід на початок циклу, якщо X = 0

2. Порядок виконання роботи

1. За заданим алгоритмом розробити програму на асемблері мікропроцесора 8086, варіант вибирається з табл. 7.3 за завданням викладача.

Таблиця 7.3 – Індивідуальні завдання на лабораторну роботу

Номер варіанта	Тип даних	Алгоритм

1
2
3
4
	Байт зі знаком
	Байт без знака
	Слово зі знаком
	Слово без знака
		IF (X = Y) Z1 = X * Y; ELSE Z1 = DIV (X / Y);
		FOR (int I = 0; I < *N*; I++)
		{
		X = X + 1; Y = Y – 1; Z2 = X * Y;
		}

5
6
7
8
	Байт зі знаком
	Байт без знака
	Слово зі знаком
	Слово без знака
		IF (X < > Y) Z1 = X * Y; ELSE Z1 = DIV (X / Y);
		WHILE (X =N)
		{
		Z2 = X * Y; X = X – 1; Y = Y + 1;
		}

9
10
11
12
	Байт зі знаком
	Байт без знака
	Слово зі знаком
	Слово без знака
		IF (X < Y) Z1 = X * Y; ELSE Z1 = DIV (X / Y);
		DO
		{
		X = X – 1; Y = Y + 1; Z2 = DIV (X / Y);
		}
		WHILE *Y* > *N*;

13
14
15
16
	Байт зі знаком
	Байт без знака
	Слово зі знаком
	Слово без знака
		SWITCH (X)
		{
		case 10: Z = Y * 2; break;
		case 20: Z = Y * 4; break;
		case 30: Z = DIV (Y / 2); break;
		default: Z = DIV (Y / 4); break;
		}
		FOR (int I = 0; I < *N*; I++)
		{
		X = X – 1; Y = Y + 1; Z2 = DIV (X / Y);
		}

17
18
19
20
	Байт зі знаком
	Байт без знака
	Слово зі знаком
	Слово без знака
		FOR (int I = 0; I < *N*; I++)
		{
		X = X + 1; Y = Y – 1; Z1 = X * Y;
		}
		SWITCH (X)
		{
		case 0: Z2 = DIV (Y / 2); break;
		case 1: Z2 = DIV (Y / 4); break;
		case 2: Z2 = Y * 2; break;
		default: Z2 = Y * 4; break;
		}

2. Набрати свою програму в текстовому редакторі та відкомпілювати за допомогою емулятора мікропроцесора 8086 «emu8086».

3. Провести тестування роботи програми в покроковому режимі.

4. Підставити в програму реальні значення змінних і перевірити відповідність роботи програми до заданого алгоритму. Результат показати викладачу.

3. Зміст звіту

1. Список команд, що були вивчені в цій лабораторній роботі з коротким описом.
2. Заданий алгоритм програми.
3. Програма на асемблері мікропроцесора 8086 з коментарями.
4. Відповіді на контрольні запитання (за завданням викладача).

4. Контрольні запитання

1. Для чого потрібна група команд безумовного та умовного переходів?
2. Особливості команд переходів мікропроцесора 8086.
3. Типи безумовних переходів мікропроцесора 8086.
4. Як реалізується операція порівняння в мікропроцесорі 8086?
5. Типи умовних переходів у мікропроцесорі 8086.
6. Недоліки команд умовного переходу мікропроцесора 8086.
7. Призначення та робота команди LOOP мікропроцесора 8086.
8. Призначення та робота команди LOOPZ/LOOPE мікропроцесора 8086.
9. Призначення та робота команди LOOPNZ/LOOPNE мікропроцесора 8086.
10. Які оператори можна реалізувати за допомогою команд умовного переходу?
11. Призначення та робота умовного оператора IF–THEN.
12. Призначення та робота оператора вибору CASE.
13. Які типи циклів використовуються при програмуванні на асемблері мікропроцесора 8086?
14. За допомогою яких команд умовного переходу можна перевірити умову $op1 = op2$?
15. За допомогою яких команд умовного переходу можна перевірити умову $op1 <> op2$?
16. За допомогою яких команд умовного переходу можна перевірити умову $op1 < op2$?
17. Реалізувати на асемблері оператор: WHILE $X < 0$ DO *op1*.
18. Реалізувати на асемблері оператор: REPEAT *op1* UNTIL $X <> 0$.
19. Реалізувати на асемблері оператор: WHILE $X = 0$ DO *op1*.
20. Реалізувати на асемблері оператор: REPEAT *op1* UNTIL $X >= 10$.
21. Реалізувати на асемблері оператор: WHILE $X <= 5$ DO *op1*.
22. Реалізувати на асемблері оператор: REPEAT *op1* UNTIL $X > 12$.

Лабораторні заняття 8

ОБРОБКА МАСИВІВ НА АСЕМБЛЕРІ МІКРОПРОЦЕСОРА 8086

Мета роботи

Вивчити команди мікропроцесора 8086 для обробки масивів, а також навчитися розв'язувати типові задачі обробки одномірних масивів на асемблері.

1. Короткі відомості з теорії

При написанні програм дуже часто необхідно виконувати обробку рядків символів (так на асемблері називаються одномірні масиви). Звісно рядки символів можна обробляти звичайними командами мікропроцесора за допомогою детермінованих циклів типу FOR. Але для обробки рядків більш ефективніше використовувати спеціальні команди для роботи з рядками, які дозволяють одною або парою команд виконати необхідну операцію над усіма символами рядками. Мікропроцесор 8086 має п'ять команд для роботи з рядками. Але так як рядки символів в мікропроцесорі 8086 можуть складатись як з байтів так і зі слів, то кожна команда роботи з рядками має дві модифікації. Якщо останній символ команди роботи з рядками «B» (Byte), то ця команда використовується для роботи з одномірними масивами байт, а якщо останній символ команди роботи з рядками «W» (Word), то ця команда використовується для роботи з одномірними масивами слів. Використання команд роботи з рядками при обробці одномірних масивів скорочує розмір програм та підвищує швидкість обробки даних. [4]

Таблиця 9.1. – Команди роботи з рядками

Команда	Призначення команди для роботи з рядками	d, крок зміни адреси наступного елемента масиву	
		прапор напряму DF = 0	прапор напряму DF = 1
CMPSB	Порівняння рядків з байт	+1	–1
CMPSW	Порівняння рядків зі слів	+2	–2
SCASB	Сканування рядка з байт	+1	–1
SCASW	Сканування рядка зі слів	+2	–2
MOVSB	Переміщення рядка з байт	+1	–1
MOVSW	Переміщення рядка зі слів	+2	–2
STOSB	Зберігання рядка з байт	+1	–1
STOSW	Зберігання рядка зі слів	+2	–2
LODSB	Завантаження рядка з байт	+1	–1
LODSW	Завантаження рядка зі слів	+2	–2

Команда порівняння рядків CMPS (compare strings) порівнює пару елементів двох рядків адреси яких задаються сегментними та індексними регістрами: [DS:SI] для символу першого рядка та [ES:DI] – для символу друго-

го рядка. Так як місце знаходження обох операндів команди відомо мікропроцесору, тому в команді вони не вказуються. Дана команда аналогічна команді CMP яка виконує операцію віднімання (SUB) за винятком того, що результат операції нікуди не записується, але по результатам операції модифікуються всі біти умов в регістрі прапорів мікропроцесора 8086. Після виконання команди значення індексних регістрів SI та DI автоматично змінюється на крок *d* (дивись табл. 9.1), вказуючи таким чином на наступні елементи обох рядків (масивів). **Команда CMPSB (CMPSW)** виконує наступні дії: [DS:SI] – [ES:DI]; SI = SI ± d; DI = DI ± d.

Команда сканування рядка SCAS (scan string) порівнює зміст регістра AL або AX мікропроцесора з символом рядка на який вказує пара регістрів [ES:DI]. Дана команда також аналогічна команді CMP яка виконує операцію віднімання за винятком того, що результат операції нікуди не записується, але по результатам операції модифікуються всі біти умов. Після виконання команди значення індексного регістру DI автоматично змінюється на крок *d* (дивись табл. 9.1), вказуючи таким чином на наступний елемент рядка (масиву). **Команда SCASB (SCASW)** виконує наступні дії: AL (AX) – [ES:DI]; DI = DI ± d.

Команду сканування рядка SCAS найчастіше використовують для пошуку в рядку елемента, який дорівнює або не дорівнює заданому символу в регістрі AL або AX.

Команда переміщення рядка MOVS (move string) переміщує символ (байт або слово) із одного рядка (адреса задається парою регістрів [DS:SI]) в інший рядок (адреса задається парою регістрів [ES:DI]). Так як місце знаходження обох операндів команди відомо мікропроцесору, тому в команді вони не вказуються. Після виконання команди значення індексних регістрів SI та DI автоматично змінюється на крок *d* (дивись табл. 9.1), вказуючи таким чином на наступні елементи обох рядків (масивів). **Команда MOVSB (MOVSW)** виконує наступні дії: [DS:SI] => [ES:DI]; SI = SI ± d; DI = DI ± d.

Команду переміщення рядка MOVS можна використовувати для швидкого переміщення змісту масиву із однієї області пам’яті в іншу.

Команда зберігання рядка STOS (store string) записує зміст регістра AL або AX мікропроцесора в чарунки пам’яті на яку вказує пара регістрів [ES:DI]. Після виконання команди значення індексного регістру DI автоматично змінюється на крок *d* (дивись табл. 9.1). **Команда STOSB (STOSW)** виконує наступні дії: AL (AX) => [ES:DI]; DI = DI ± d.

Команда завантаження рядка LODS (load string) записує в регістр AL або AX мікропроцесора зміст чарунок пам’яті на яку вказує пара регістрів [DS:SI] після чого значення індексного регістру SI автоматично змінюється на крок *d* (дивись табл. 9.1). **Команда LODSB (LODSW)** виконує наступні дії: [DS:SI] => AL (AX); SI = SI ± d.

Також для роботи з рядками мікропроцесор 8086 має ряд додаткових команд [4]**:**

1. **CLD** (clear DF) – скидання прапору напряму DF = 0;

2. **STD** (set DF) – встановлення прапору напряму DF = 1;

3. **LEA** *r*16, < ім'я змінної >. Ця команда обчислює виконавчу адресу змінної (другий операнд) і заносить її в регістр (перший операнд), наприклад:

LEA BX, X; завантаження адреси змінної *X* в регістр *BX*.

Використовуючи цю команду, можна завантажити в індексний регістр початкову адресу масиву, наприклад:

X DB 10 DUP (?); масив *X* на 10 елементів;

LEA *SI, X*; запис у регістр *SI* початкової адреси масиву *X*.

4. **XLAT** – команда перекодування: у регістр *AL* записується байт із чарунки пам'яті з адресою *[BX+AL]*. Команду XLAT можна використовувати для перекодування символів. Наприклад, необхідно перетворити число від 0 до 15 у відповідний йому символ шістнадцяткової системи числення 0…F:

H16 DB '0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F'; масив символів

LEA BX, H16; у регістр *BX* записуємо початкову адресу масиву символів;

XLAT; AL = H16 [AL] – у регістр *AL* записується код символу.

Розміщення рядків (масивів) в пам'яті мікропроцесорної системи.

Для розміщення масиву в пам'яті мікропроцесорної системи в асемблері використовується спеціальна конструкція повторення, у вигляді директива **DUP,** яка має такий формат: <ім'я масиву> <тип даних> <кількість елементів масиву> DUP (початкове значення).

Приклади використання директиви DUP для створення масивів:

X DB 10 DUP (0) – створюється **масив байтів** з ім'ям *X*, що складається з 10 елементів із початковим значенням усіх елементів, які дорівнюють нулю.

Y DW 30 DUP (?) – створюється **масив слів** з 30 елементів із невизначеним початковим значенням усіх елементів.

Z DB 10 DUP (5 DUP (?)) – створюється **двовимірний масив байтів**, що є матрицею розміром 10×5 елементів із невизначеним початковим значенням усіх елементів.

Для звернення до окремих елементів масиву в асемблері мікропроцесора 8086 використовується модифікація адреси. В якості модифікатора адреси в мікропроцесорі 8086 можна використовувати чотири регістри: *BX, SI, DI* та *BP*. Регістр *BP* не рекомендується використовувати для обробки масивів, оскільки він за умовчанням адресує елементи пам'яті в сегменті стека, а не в сегменті даних, де звичайно розташовуються всі масиви. Якщо регістр використовується як модифікатор, то він указується в команді у квадратних дужках, наприклад: MOV AX, X[BX]. За такою командою в регістр *AX* буде зчитаний вміст чарунки пам'яті, адреса якої буде рівна сумі адреси змінної *X* та вмісту регістра *BX*: ADR = *X*+[*BX*]. Таким чином, для звертання до *i*–елемента масиву, в команді необхідно вказати ім'я масиву, а в регістр–модифікатор необхідно записати номер (індекс) елемента масиву – число *i*. Перший елемент масиву має номер – нуль, а останній елемент масиву – $(N-1)$, де N – число елементів масиву. Оскільки для індексації елементів масиву використовується 16–розрядний регістр, то розмір масиву в мікропроцесорі 8086 не може перевищувати 64 Кбайти (65536 елементів).

Для обробки двовимірних масивів у асемблері мікропроцесора 8086 використовується модифікація за двома регістрами. При зверненні до елемента двовимірного масиву необхідно вказувати два індекси: номер рядка й номер стовпця елемента. У мікропроцесорі 8086 перший модифікатор обов'язково вказується в регістрі *BX* або *BP*, а другий модифікатор, у регістрі *SI* або *DI*. Інші варіанти пар регістрів модифікаторів заборонені. Наприклад, треба адресувати елемент масиву з розміром 10×10 байт:

```
X DB 10 DUP (10 DUP (?)); двовимірний масив 10×10 байт
MOV SI, 5; п'ятий стовпець масиву
MOV BX, 5; п'ятий рядок масиву
MOV AH, X[BX][SI]; запис у регістр AH елемента масиву.
```

Обробку одновимірних масивів на асемблері розберемо на такому прикладі. Задано масив із 100 байт. Необхідно обчислити суму елементів цього масиву та помістити її у змінну *SUM*. Текст програми на асемблері, що дозволяє розв'язати цю задачу, наведений нижче.

```
; Лабораторна робота 8
include 'emu8086.inc'; підключення загальних функцій емулятора 8086
;_____________________________________
; сегмент даних програми
DATA SEGMENT
PKEY DB 13,10,"Для виходу із програми натисніть будь–яку клавішу$"
MAS1 DB 100 DUP (?); масив на 100 байт
SUM DB ?; змінна для збереження результату
ENDS
;_____________________________________
; сегмент стека програми
STACK SEGMENT
    DW 128 DUP(0)
ENDS
;_____________________________________
; сегмент коду програми
CODE SEGMENT
START:
; налаштування сегментних регістрів
    MOV AX, DATA
    MOV DS, AX
    MOV ES, AX
;_____________________________
; початок обробки одновимірного масиву
    LEA SI, MAS1; завантажуємо в індексний регістр SI адресу масиву MAS1
    MOV SUM, 0; обнуляємо змінну для зберігання суми елементів масиву
    CLD; скидаємо прапор напряму DF = 0
    MOV CX, 100; організовуємо цикл на 100 байт
```

```
M1: LODSB; записуємо в регістр AL i–ий елемент масиву
    ADD SUM, AL; обчислення суми елементів масиву
    LOOP M1; кінець циклу
; вивід результату на екран
    PRINT 'Сума елементів масиву = '
    MOV AL, SUM
    MOV AH, 0
    CALL PRINT_NUM_UNS; вивід беззнакових чисел із регістра AX
;————————————————
; вихід із програми у ОС Windows
; вивід рядка на екран
    LEA DX, PKEY
    MOV AH, 9
    INT 21H
; очікування натиснення клавіші
    MOV AH, 1
    INT 21H
; повернення в операційну систему
    MOV AX, 4C00H
    INT 21H
ENDS
DEFINE_PRINT_NUM_UNS; – для беззнакових чисел
END START; місце входу в програму
```

2. Порядок виконання лабораторної роботи

1. За заданим варіантом розробити алгоритм та програму на асемблері. Виконуючи індивідуальне завдання, необхідно звернути увагу на те, що під час обробки масиву слів для обчислення адреси наступного елемента масиву необхідно збільшувати індексний регістр на два.

2. Набрати свою програму в текстовому редакторі та відкомпілювати за допомогою емулятора мікропроцесора 8086 «emu8086».

3. Провести тестування роботи програми в покроковому режимі.

4. Підставити в програму реальні значення змінних і перевірити відповідність роботи програми до заданого алгоритму. Результат показати викладачу.

Варіанти індивідуальних завдань

1. Задати два масиви із 10 байт кожен. Порівняти *i*-елементи масивів та вивести на екран повідомлення: «Масиви однакові» або «Масиви різні».

2. Задати два масиви із 8 слів кожен. Порівняти *i*-елементи масивів та вивести на екран повідомлення: «Масиви однакові» або «Масиви різні».

3. Заданий масив із 10 байт. Знайти кількість елементів масиву, значення яких більше константи 80*h*, та помістити результат у змінну *RES*.

4. Заданий масив із 13 слів. Знайти кількість елементів масиву, значення

яких більше константи 0AF00*h*, та помістити результат у змінну *RES*.

5. Заданий масив із 15 байт. Знайти кількість елементів масиву, значення яких менше константи 0C0*h*, та помістити результат у змінну *RES*.

6. Заданий масив із 9 слів. Знайти кількість елементів масиву, значення яких більше константи 80FF*h*, та помістити результат у змінну *RES*.

7. Заданий масив із 12 байт. Знайти кількість елементів масиву, значення яких дорівнює константі 7F*h*, та помістити результат у змінну *RES*.

8. Заданий масив із 12 слів. Знайти кількість елементів масиву, значення яких дорівнює константі 7FFF*h*, та помістити результат у змінну *RES*.

9. Заданий масив із 14 байт. Знайти кількість елементів масиву, значення яких не дорівнює константі 0AF*h*, та помістити результат у змінну *RES*.

10. Заданий масив із семи слів. Знайти кількість елементів масиву, значення яких не дорівнює константі 0FFF0*h*, та помістити результат у змінну *RES*.

11. Заданий масив із 11 слів. Знайти елемент масиву, що має мінімальне значення, та помістити результат у змінну *MIN*.

12. Заданий масив із 13 байт. Знайти елемент масиву, що має мінімальне значення, та помістити результат у змінну *MIN*.

13. Заданий масив із 15 слів. Знайти елемент масиву, що має максимальне значення, та помістити результат у змінну *MAX*.

14. Заданий масив із 12 байт. Знайти елемент масиву, що має максимальне значення, та помістити результат у змінну *MAX*.

15. Заданий масив із 10 байт. Знайти елемент масиву, що має мінімальне значення, та помістити результат у змінну *MIN*.

16. Заданий масив із 12 слів. Знайти елемент масиву, що має мінімальне значення, та помістити результат у змінну *MIN*.

17. Заданий масив із 14 байт. Знайти елемент масиву, що має максимальне значення, та помістити результат у змінну *MAX*.

18. Заданий масив із 7 слів. Знайти елемент масиву, що має максимальне значення, та помістити результат у змінну *MAX*.

3. Зміст звіту

1. Список команд та директив асемблера, що були вивченні в цій лабораторній роботі, з коротким описом.
2. Алгоритм і програма на асемблері з коментарями.
3. Результат роботи програми.
4. Відповіді на контрольні запитання (за завданням викладача).

4. Контрольні запитання

1. Призначення директиви асемблера DUP.
2. Що таке модифікація адреси в мікропроцесорі 8086?
3. Який максимальний розмір масиву може обробити мікропроцесор 8086 і чому?
4. Для чого в мікропроцесорі 8086 використовується модифікація по двох регістрах?
5. Призначення та робота команди LEA.

6. Призначення та робота команди XLAT.

7. Задати одномірний масив на 20 байт із початковим значенням, рівним одиниці.

8. Задати двовимірний масив розміром 10×5 байт із довільним початковим значенням.

9. Скільки чарунок пам'яті займе масив: *X* DW 10 DUP (8 DUP (?))?

10. Скільки чарунок пам'яті займе масив: *Y* DD 6 DUP (6 DUP (0))?

Лабораторні заняття 9

КОМАНДИ ВВОДУ–ВИВОДУ МІКРОПРОЦЕСОРА 8086

Мета роботи

Навчитися використовувати команди вводу-виводу мікропроцесора 8086 для керування зовнішніми пристроями.

1. Короткі відомості з теорії

До складу будь-якої мікропроцесорної системи, окрім пам'яті, обов'язково входять пристрої вводу-виводу (ПВВ), за допомогою яких здійснюються операції обміну даними між мікропроцесором та зовнішніми пристроями. Будь-який ПВВ є портом із певною адресою. Мікропроцесор 8086 може адресувати 65536 портів вводу-виводу (діапазон адрес 0...FFFF*h*), чого цілком достатньо для реалізації мікропроцесорної системи будь-якої складності. Найчастіше всього порт вводу-виводу є регістром розміром у байт, який використовується або для обміну даними між мікропроцесором і зовнішнім пристроєм, або для керування зовнішнім пристроєм.

Мікропроцесор 8086 має чотири команди вводу–виводу [4]:

– читання з порту байта: IN *AL, n* ($AL \Leftarrow \text{port}\, n$);

– читання з порту слова: IN *AX, n* ($AX \Leftarrow \text{port}\, n$);

– запис у порт байта: OUT *n, AL* ($\text{port}\, n \Leftarrow AL$);

– запис у порт слова: OUT *n, AX* ($\text{port}\, n \Leftarrow AX$).

Адресу порту *n* в мікропроцесорі 8086 можна задавати двома способами:

– як константу *i*8, якщо адреса порту розташована в діапазоні від 0 до 255 (0...FF*h*);

– як вміст регістра *DX*, якщо адреса порту 16–розрядна (0000*h*…FFFF*h*).

Використання регістра *DX* як покажчика адреси порту дозволяє не тільки задавати адресу будь-якого ПВВ, але й обчислювати його під час виконання програми, оскільки регістр *DX* може використовуватися в будь-яких арифметичних і логічних операціях. Програмування зовнішніх пристроїв за допомогою команд вводу-виводу вимагає від програміста хорошого знання схемотехніки та режимів роботи програмованого пристрою. Алгоритм обміну даними через порти вводу-виводу залежить від конкретного зовнішнього пристрою. Звичайно кожний ПВВ має в своєму складі регістри даних, через які

здійснюється обмін даними між мікропроцесором і зовнішнім пристроєм, а також регістри керування та стану, за допомогою яких здійснюється керування зовнішнім пристроєм та контроль за його станом.

Емулятор мікропроцесора 8086 «emu8086» має декілька вбудованих віртуальних пристроїв, що дозволяють навчитися використовувати команди вводу-виводу мікропроцесора 8086 для керування зовнішніми пристроями: семисегментний LED індикатор (LED Display), покроковий двигун (Stepper motor), керування роботом (Robot), керування світлофорами на перехресті (Traffic Lights), контроль та керування температурою (Thermometer), віртуальний принтер (Printer) та інші. Подивитися повний список віртуальних пристроїв емулятора можна через меню емулятора «virtual devices». Віртуальний пристрій це спеціальна програма, що написана на мові високого рівня C++ або Visual Basic, й імітує роботи якогось зовнішнього пристрою. В емуляторі «emu8086» можна також самостійно розробляти віртуальні пристрої, що імітують роботу необхідних зовнішніх пристроїв. Наприклад, стрілочний перевід, світлофор та інше.

Приклад програми, що виводить числа в діапазоні знакового слова (–32768...+32767) на семисегментний індикатор наведений нижче. Для виводу знакового слова на індикатори потрібно розмістити число в регістрі AX і записати в порт 199.

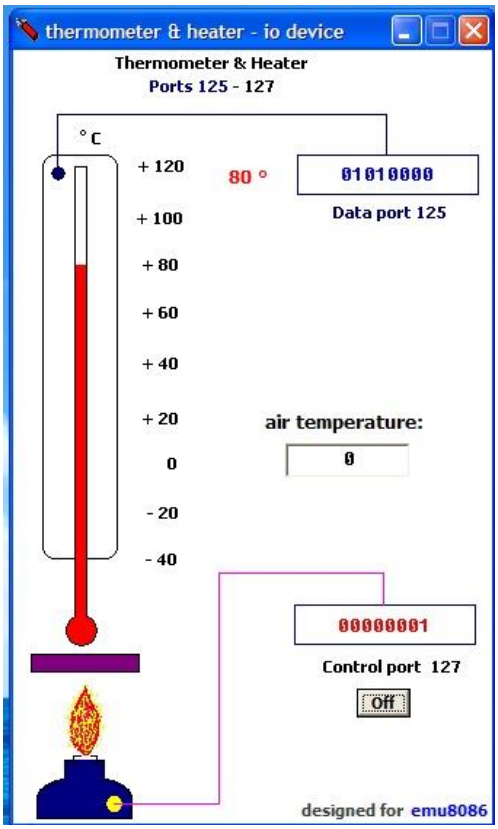
```
; Лабораторна робота № 9.1
#start = led_display.exe# ; завантажити віртуальний пристрій LED_Display
;_____________________________________
; сегмент даних програми
DATA SEGMENT
PKEY DB "Для виходу із програми натисніть будь–яку клавішу$"
ENDS
;_____________________________________
; сегмент стека програми
STACK SEGMENT
    DW   128  DUP(0)
ENDS
;_____________________________________
; сегмент коду програми
CODE SEGMENT
START:
; налаштування сегментних регістрів
    MOV AX, DATA
    MOV DS, AX
    MOV ES, AX
;_____________________________
    MOV AX, 12345; запис числа в регістр AX
    OUT 199, AX; вивід числа на індикатори
;_____________________________
; вихід із програми у Windows
; вивід рядка на екран
```

```
    LEA DX, PKEY
    MOV AH, 9
    INT 21H
; очікування натиснення клавіші
    MOV AH, 1
    INT 21H
; повернення в операційну систему
    MOV AX, 4C00H
    INT 21H
ENDS
END START; місце входу в програму
```

2. Порядок виконання лабораторної роботи

1. Набрати в текстовому редакторі програму на асемблері мікропроцесора 8086, яка підтримує температуру в діапазоні від 60 до 80 ^{0}C. Читання поточної температури здійснюється через порт 125. Температура може перебувати в діапазоні знакового байта –128 до +127 ^{0}C. Вмикання та вимикання нагрівача здійснюється через порт 127: запис лог.1 – вмикає нагрівач, запис лог.0 – вимикає нагрівач.

```
; Лабораторна робота № 9.2
#start = thermometer.exe# ; завантажити віртуальний пристрій
;_____________________________________
; сегмент даних програми
DATA SEGMENT
PKEY DB " Для виходу із програми натисніть будь-яку клавішу $"
TMIN EQU 60; мінімальна температура
TMAX EQU 80; максимальна температура
ENDS
;_____________________________________
; сегмент стека програми
STACK SEGMENT
    DW   128  DUP(0)
ENDS
;_____________________________________
; сегмент коду програми
CODE SEGMENT
START:
; налаштування сегментних регістрів
    MOV AX, DATA
    MOV DS, AX
    MOV ES, AX
;_____________________________
; вивід рядка на екран
    LEA DX, PKEY
    MOV AH, 9
```



```
    INT 21H
; основний цикл керування
main:
    IN AL, 125
    CMP AL, TMIN; перевірка мінімальної температури
    JLE LOW; перехід на вмикання нагрівача
    CMP AL, TMAX; перевірка максимальної температури
    JGE HIGH; перехід на вимикання нагрівача
    JMP OK; безумовний перехід
LOW:
    MOV AL, 1
    OUT 127, AL; вмикання нагрівача
    JMP OK
HIGH:
    MOV AL, 0
    OUT 127, AL; вимикання нагрівача
OK:
; перевірка натиснення клавіші
    MOV AH, 0BH
    INT 21H
    CMP AL, 0
    JZ MAIN; зациклювання програми
;————————————————
; вихід із програми у Windows
    MOV AX, 4C00H
    INT 21H
ENDS
END START; місце входу в програму
```

2. Відкомпілювати програму за допомогою емулятора мікропроцесора 8086 «emu8086».

3. Провести тестування роботи програми.

4. Відповісти на контрольні запитання.

3. Зміст звіту

1. Список команд вводу-виводу мікропроцесора 8086 з коротким описом.
2. Програма 10.2 на асемблері з коментарями.
3. Схема контролю та керування температурою.
4. Відповіді на контрольні запитання (за завданням викладача).

4. Контрольні запитання

1. Команди вводу-виводу мікропроцесора 8086.
2. Максимальна кількість портів вводу-виводу в мікропроцесорі 8086.
3. Способи завдання адреси порту в мікропроцесорі 8086.
4. Які регістри звичайно входять до складу ПВВ?

СПИСОК ЛІТЕРАТУРИ

1. Новацький А. О. Мікропроцесорні та мікроконтролерні системи : підручник. Київ : КПІ ім. Ігоря Сікорського, Вид-во «Політехніка», 2020. Т. 1 : Мікропроцесорні системи. 361 с. URL: https://ela.kpi.ua/handle/123456789/31216 (дата звернення: 03.01.2024).

2. Новацький А. О. Мікропроцесорні та мікроконтролерні системи : підручник. Київ : КПІ ім. Ігоря Сікорського, Вид-во «Політехніка», 2021. Т. 2 : Проектування мікропроцесорних систем. 462 с. URL: https://ela.kpi.ua/handle/123456789/43051 (дата звернення: 03.01.2024).

3. Intel® 64 and IA–32 Architectures Software Developer's Manual. Santa Clara : Intel Corp., 2021. Vol. 1 : Basic Architecture. 482 p.

4. Intel® 64 and IA–32 Architectures Software Developer's Manual. Santa Clara : Intel Corp., 2021. Vol. 2 : Instruction Set Reference. 2348 p.

5. Darche P. Microprocessor 1 : Computer engineering series. London : ISTE Ltd, 2020. 209 p.

6. Darche P. Microprocessor 2 : Computer engineering series. London : ISTE Ltd, 2020. 189 p.

7. Detmer R. C. Introduction to 80x86 Assembly Language and Computer Architecture. Burlington : Jones & Bartlett Learning, 2015. 502 p.

8. Global Techno. Microprocessors and microcontrollers. Lake Mary : Global Learning Solutions, 2018. 287 p.

9. Tanenbaum A. S., Ostin T. Structured computer organization. London : Pearson, 2013. 816 p.

10. Intel – Data Center Solutions, IoT, and PC Innovation. URL: https://www.intel.com/content/www/us/en/homepage.html (date of access: 03.01.2024)

11. AMD. URL: https://www.amd.com/en.html (date of access: 03.01.2024).

Додаток А

СПИСОК КОМАНД МІКРОПРОЦЕСОРА 8086

Умовні скорочення:

i8 – 8–розрядна константа (байт);
i16 – 16–розрядна константа (слово);
i32 – 32–розрядна константа (подвійне слово);
r8 – 8–розрядний регістр (AH, AL, BH, BL, CH, CL, DH, DL);
r16 – 16–розрядний регістр (AX, BX, CX, DX, SI, DI, BP, SP);
sr – сегментний регістр (CS, DS, SS, ES);
m8 – адреса чарунки пам'яті, у якій зберігається байт;
m16 – адреса елемента пам'яті, у якому зберігається слово (адреса молодшого байта слова);
m32 – адреса елемента пам'яті, у якому зберігається подвійне слово (адреса молодшого байта подвійного слова).

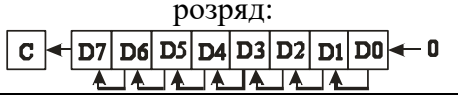
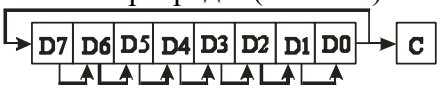
Таблиця А.1

Мнемокод команди	Тип операндів	Назва команди
1	2	3
Команди переміщення даних		
MOV op1, op2	r8 <= r8, i8, m8 m8 <= r8, i8 r16 <= r16, i16, sr, m16 sr (окрім CS) <= r16, m16 m16 <= r16, i16, sr	Команди пересилки даних: op1 <= op2
XCHG op1, op2	r8 <=> r8, m8 m8 <=> r8 r16 <=> r16, m16 m16 <=> r16	Команди перестановки даних: op1 <=> op2
CBW	AX <= AL (якщо AL>0, то AH = 00h) (якщо AL<0, то AH = FFh)	Розширення байта до слова з урахуванням знака числа
CWD	(DX, AX) <= AX (якщо AX>0, то DX = 0000h) (якщо AX<0, то DX = FFFFh)	Розширення слова до подвійного слова з урахуванням знака числа
LAHF		Завантаження регістра Flags у регістр AH
SAHF		Завантаження регістра AH у регістр Flags
Арифметичні команди		
ADD op1, op2	r8 <= r8, i8, m8 m8 <= r8, i8 r16 <= r16, i16, m16 m16 <= r16, i16	Команди додавання: op1 <= op1 + op2
SUB op1, op2	r8 <= r8, i8, m8 m8 <= r8, i8 r16 <= r16, i16, m16 m16 <= r16, i16	Команди віднімання: op1 <= op1 – op2

Продовження табл. А.1

1	2	3
CMP op1, op2	r8 <= r8, i8, m8 m8 <= r8, i8 r16 <= r16, i16, m16 m16 <= r16, i16	Команди зрівняння без збереження результату (встановлюють біти в регістрі Flags): op1 – op2
ADC op1, op2	r8 <= r8, i8, m8 m8 <= r8, i8 r16 <= r16, i16, m16 m16 <= r16, i16	Команди додавання з урахуванням біта переносу CF: op1 <= op1 + op2 + CF
SBB op1, op2	r8 <= r8, i8, m8 m8 <= r8, i8 r16 <= r16, i16, m16 m16 <= r16, i16	Команди віднімання з урахуванням біта переносу CF: op1 <= op1 – op2 – CF
INC op1	op1 = r8, r16, m8, m16	Команди інкремента: op1 <= op1 +1
DEC op1	op1 = r8, r16, m8, m16	Команди декремента: op1 <= op1 – 1
NEG op1	op1 = r8, r16, m8, m16	Команди зміни знака числа: op1 <= – op1
MUL op1	AX = AL * op1 op1 = r8, m8 (DX, AX) = AX * op1 op1 = r16, m16	Множення цілих чисел без знака
IMUL op1	AX = AL * op1 op1 = r8, m8 (DX, AX) = AX * op1 op1 = r16, m16	Множення цілих чисел із знаком
DIV op1	AL (ціле) = AX div op1 AH (залишок) = AX mod op1 op1 = r8, m8 AX (ціле) = (DX, AX) div op1 DX (залишок) = (DX, AX) mod op1 op1 = r16, m16	Ділення цілих чисел без знака
IDIV op2	AL (ціле) = AX div op1 AH (залишок) = AX mod op1 op1 = r8, m8 AX (ціле) = (DX, AX) div op1 DX (залишок) = (DX, AX) mod op1 op1 = r16, m16	Ділення цілих чисел із знаком
Команди логічних операцій		
AND op1, op2	r8 <= r8, i8, m8 m8 <= r8, i8 r16 <= r16, i16, m16 m16 <= r16, i16	Команди логічного множення: op1 <= op1 and op2
TEST op1, op2	r8 <= r8, i8, m8 m8 <= r8, i8 r16 <= r16, i16, m16 m16 <= r16, i16	Команди логічного множення без збереження результату (встановлюють біти в регістрі Flags): op1 and op2

Продовження табл. А.1

1	2	3
OR op1, op2	r8 <= r8, i8, m8 m8 <= r8, i8 r16 <= r16, i16, m16 m16 <= r16, i16	Команди логічного додавання: op1 <= op1 or op2
XOR op1, op2	r8 <= r8, i8, m8 m8 <= r8, i8 r16 <= r16, i16, m16 m16 <= r16, i16	Команди додавання по модулю два: op1 <= op1 xor op2
NOT op1	op1 = r8, r16, m8, m16	Команди логічного заперечення (інвертування біт)
SHL op1, 1	op1 = r8, r16, m8, m16	Логічний зсув уліво на один розряд: 
SHL op1, CL	op1 = r8, r16, m8, m16	Логічний зсув уліво на n розрядів (CL <= n): 
SHR op1, op2	op1 = r8, r16, m8, m16 op2 = 1 або CL	Логічний зсув управо на один або n розрядів (CL <= n): 
SAL op1, op2	op1 = r8, r16, m8, m16 op2 = 1 або CL	Арифметичний зсув уліво на один або n розрядів (CL <= n): 
SAR op1, op2	op1 = r8, r16, m8, m16 op2 = 1 або CL	Арифметичний зсув управо на один або n розрядів (CL <= n): 
ROL op1, op2	op1 = r8, r16, m8, m16 op2 = 1 або CL	Циклічний зсув уліво на один або n розрядів (CL <= n): 
ROR op1, op2	op1 = r8, r16, m8, m16 op2 = 1 або CL	Циклічний зсув управо на один або n розрядів (CL <= n): 
RCL op1, op2	op1 = r8, r16, m8, m16 op2 = 1 або CL	Циклічний зсув уліво через біт переносу на один або n розрядів (CL <=n): 
RCR op1,op2	op1 = r8, r16, m8, m16 op2 = 1 або CL	Циклічний зсув управо через біт переносу на один або n розрядів (CL <= n): 

Продовження табл. А.1

1	2	3
CLC	CF = 0	Скидання біта переносу
STC	CF = 1	Установка біта переносу
CMC		Інвертування біта переносу
Команди переходів та циклів		
JMP <мітка>		Безумовний прямий перехід
JMP op1	op1 = r16, m16	Безумовний непрямий перехід
JE <мітка>	Використовується одразу після команди зрівняння CMP	Умовний перехід на мітку, якщо op1 = op2 (біт ZF = 1)
JNE <мітка>	Використовується одразу після команди зрівняння CMP	Умовний перехід на мітку, якщо op1 < > op2 (біт ZF = 0)
JL <мітка>	Використовується зразу після команди зрівняння CMP (для чисел із знаком)	Умовний перехід на мітку, якщо op1 < op2 (біти SF < > OF)
JLE <мітка>	Використовується одразу після команди зрівняння CMP (для чисел із знаком)	Умовний перехід на мітку, якщо op1 <=op2 (біти SF < > OF або біт ZF = 1)
JG <мітка>	Використовується одразу після команди зрівняння CMP (для чисел із знаком)	Умовний перехід на мітку, якщо op1 > op2 (біти SF < > OF та біт ZF = 0)
JGE <мітка>	Використовується одразу після команди зрівняння CMP (для чисел із знаком)	Умовний перехід на мітку, якщо op1 >= op2 (біти SF < > OF)
JB <мітка>	Використовується одразу після команди зрівняння CMP (для чисел без знака)	Умовний перехід на мітку, якщо op1 < op2 (біт CF = 1)
JBE <мітка>	Використовується одразу після команди зрівняння CMP (для чисел без знака)	Умовний перехід на мітку, якщо op1 <= op2 (біти CF = 1 або ZF = 1)
JA <мітка>	Використовується одразу після команди зрівняння CMP (для чисел без знака)	Умовний перехід на мітку, якщо op1 > op2 (біти CF = 0 і ZF = 0)
JAE <мітка>	Використовується одразу після команди зрівняння CMP (для чисел без знака)	Умовний перехід на мітку, якщо op1 >= op2 (біт CF = 0)
JZ (JNZ) <мітка>	Аналіз біта нуля	Умовний перехід на мітку, якщо біт ZF = 1 (ZF = 0)
JS (JNS) <мітка>	Аналіз біта знака	Умовний перехід на мітку, якщо біт SF = 1 (SF = 0)
JC (JNC) <мітка>	Аналіз біта переносу	Умовний перехід на мітку, якщо біт CF = 1 (CF = 0)
JO (JNO) <мітка>	Аналіз біта переповнення	Умовний перехід на мітку, якщо біт OF = 1 (OF = 0)
JP (JNP) <мітка>	Аналіз біта паритету (парності)	Умовний перехід на мітку, якщо біт PF = 1 (PF = 0)
JCXZ <мітка>	Аналіз регістра CX	Умовний перехід на мітку, якщо вміст регістра CX = 0

Продовження табл. А.1

1	2	3
LOOP <мітка>	Детермінований цикл типу FOR TO	Команда організації циклу: CX = CX – 1 та перехід на мітку, якщо CX < > 0
LOOPZ (LOOPE) <мітка>	Детермінований цикл типу FOR TO з можливістю дострокового виходу з циклу	Команда організації циклу з достроковим виходом за умови: CX = CX – 1 та перехід на мітку, якщо CX <> 0 та ZF = 1
LOOPNZ (LOOPNE) <мітка>	Детермінований цикл типу FOR TO з можливістю дострокового виходу з циклу	Команда організації циклу з достроковим виходом за умови: CX = CX – 1 та перехід на мітку, якщо CX < > 0 та ZF = 0
Команди організації підпрограм		
PUSH op1	op1 = r16, sr, m16	Команда запису даних у стек: SP = SP – 2, op1 => [SS:SP]
POP op1	op1 = r16, sr , m16	Команда читання даних із стека: [SS:SP] => op1, SP = SP + 2
PUSHF		Команда запису регістра Flags до стека
POPF		Команда читання регістра Flags із стека
CALL <ім’я підпрограми>		Виклик підпрограми (адреса повернення з підпрограми записується в стек)
RET		Команда повернення з підпрограми
INT n	n = i8	Команда програмного виклику переривань із номером n
IRET		Команда повернення з підпрограми обробки переривання
Команди роботи з рядками		
CMPSB	[DS:SI] – [ES:DI]; SI = SI ± 1; DI = DI ± 1	Порівняння рядків з байт
CMPSW	[DS:SI] – [ES:DI]; SI = SI ± 2; DI = DI ± 2	Порівняння рядків зі слів
SCASB	AL – [ES:DI]; DI = DI ± 1	Сканування рядка з байт
SCASW	AX – [ES:DI]; DI = DI ± 2	Сканування рядка зі слів
MOVSB	[DS:SI] => [ES:DI]; SI = SI ± 1; DI = DI ± 1	Переміщення рядка з байт
MOVSW	[DS:SI] => [ES:DI]; SI = SI ± 2; DI = DI ± 2	Переміщення рядка зі слів
STOSB	AL => [ES:DI]; DI = DI ± 1	Зберігання рядка з байт
STOSW	AX => [ES:DI]; DI = DI ± 2	Зберігання рядка зі слів
LODSB	[DS:SI] => AL; SI = SI ± 1	Завантаження рядка з байт
LODSW	[DS:SI] => AX; SI = SI ± 2	Завантаження рядка зі слів
CLD	DF = 0	Скидання прапору напряму
STD	DF = 1	Встановлення прапору напряму

Закінчення табл. А.1

LEA r16, < ім'я змінної >	Адреса змінної => r16	Завантаження адреси змінної в регістр r16
XLAT	[BX+AL] => AL	Команда перекодування
Команди вводу–виводу та спеціальні команди керування мікропроцесором		
IN AL, n	n = i8, регістр DX (16–р. адреса)	Читання байта з порту з адресою n
IN AX, n	n = i8, регістр DX (16–р. адреса)	Читання слова з порту з адресою n
OUT n, AL	n = i8, регістр DX (16–р. адреса)	Запис байта в порт з адресою n
OUT n, AX	n = i8, регістр DX (16–р. адреса)	Запис слова в порт з адресою n
NOP		Холоста операція
HLT		Зупинка процесора до виникнення переривання
LOCK		Блокування шин до завершення виконання процесором команди, яка розташована за командою LOCK
WAIT		Перевід процесора в стан очікування

Додаток Б

ПРАВИЛА ОФОРМЛЕННЯ ЛАБОРАТОРНИХ ЗАНЯТЬ

1. Формат листів:

– А4 (без рамок), орієнтування – книжне;

– поля: ліве і праве – по 2,25 см, верхнє – 1,8 см, нижнє – 3,0 см.

2. Шрифт тексту:

– тип – Times New Roman (TNR);

– розмір – 14;

– в набраному тексті не повинно бути шрифтів іншого типу.

3. Параметри абзацу для основного тексту:

– міжрядковий інтервал – 1,0 (одинарний);

– абзацний відступ на 0,77 см від початку рядка, однаковий по всьому основному тексту;

– вирівнювання абзаців – по ширині;

– абзаци основного тексту не відділяються один від одного.

4. Параметри абзацу для заголовків розділу:

– шрифт TNR, розмір 14, напівжирний;

– вирівнювання абзаців – по центру;

– відступу першого рядка не має;

– від попереднього тексту підзаголовок відділяється інтервалом в 12 пунктів, а наступного – 6 пунктів.

5. Зразок оформлення рисунків:

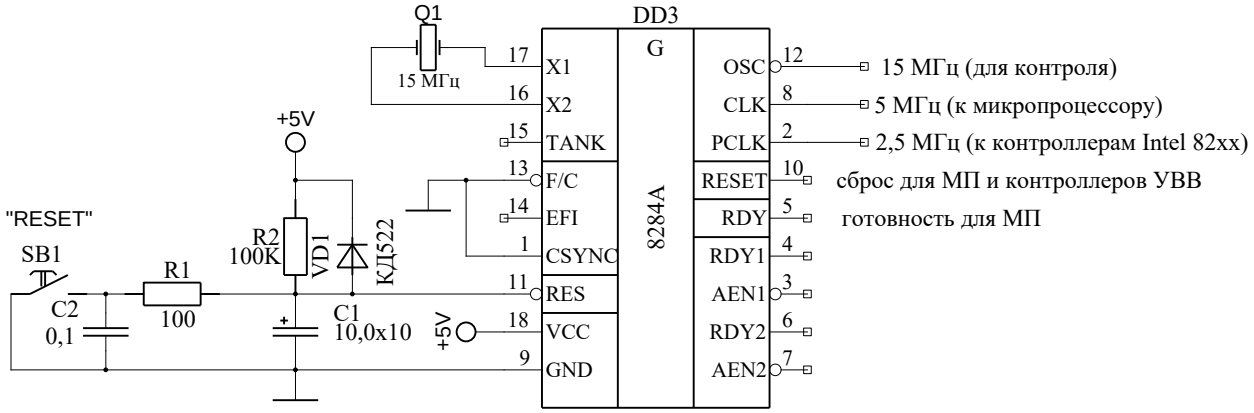


Рис. 1. УГО ИМС i8284A

Параметри абзацу для рисунків та підписом під рисунком:

– вирівнювання абзаців – по центру;

– відступу першого рядка не має;

– від попереднього тексту рисунок та підпис відділяється інтервалом в 6 пунктів, а наступного – 12 пунктів.

Навчально-методичне видання

Профатилов Володимир Іванович

МІКРОПРОЦЕСОРНІ ЗАСОБИ АВТОМАТИЗАЦІЇ

Навчально-методичні рекомендації до лабораторних занять

Електронне видання

Експертний висновок склав канд. техн. наук, доц. Костянтин Гончаров

Зареєстровано НМВ УДУНТ (№ 689 від 05.02.2024)

В авторській редакції
Комп'ютерна верстка Профатилов В. І.

Формат 60x84 1/16. Ум. друк. арк. 3,95. Обл.-вид. арк. 2,77.
Зам. № 48

Видавець: Український державний університет науки і технологій
вул. Лазаряна, 2, ауд. 2216 м. Дніпро, 49010.
Свідоцтво суб'єкта видавничої справи ДК № 7709 від 14.12.2022

Адреса видавця та дільниці оперативної поліграфії:
вул. Лазаряна, 2, м. Дніпро, 49010