НАВЧАЛЬНО-МЕТОДИЧНА ЛІТЕРАТУРА

Тернопільський національний технічний університет імені Івана Пулюя

Кафедра автоматизації технологічних процесів і виробництв

Електроніка і мікропроцесорна техніка

Методичні вказівки для лабораторної роботи № 26

Організація циклів, обробка масивів і реалізація логічних функцій в МП КР580ВМ80

Тернопіль 2016

Методичні вказівки для лабораторної роботи №26. " Організація циклів, обробка масивів і реалізація логічних функцій в МП КР580ВМ80" з курсу " Електроніка і мікропроцесорна техніка ". /Медвідь В.Р., Микулик П.М., Пісьціо В.П., Тернопіль: ТНТУ, 2016 - 11 с.

Для студентів напряму: 6.050202 "Автоматизоване управління технологічними процесами.

Методічні вказівки розглянуті і затверджені на засіданні кафедри автоматизації технологічних процесів і виробництв (протокол № 7 від 23.12.2015 року).

ЛАБОРАТОРНА РОБОТА №26

ОРГАНІЗАЦІЯ ЦИКЛІВ, ОБРОБКА МАСИВІВ І РЕАЛІЗАЦІЯ ЛОГІЧНИХ ФУНКЦІЙ В МП КР580ВМ80

Мета роботи.

1. Вивчення логічних команд, команд переходів і викликів підпрограм, команд вводу-виводу і роботи зі стеком.
2. Розгляд алгоритмів розв'язання складних задач і правил їх складання.
3. Дослідження програм для обробки масивів чисел, рішення алгебраїчних і логічних задач.

Короткі теоретичні відомості

1. Логічні команди

Логічні команди виконують побітно (порозрядно) логічні операції над вмістом акумулятора і операнду, зазначеного в команді.

В якості операнду в таких командах використовується регістр, елемент пам'яті або безпосередньо задане в команді число.

Відмінною рисою логічних команд є відсутність формування флагів перенесення АС і С.

Основними логічними операціями є: кон’юнкція (логічне «І»), диз'юнкція (логічне «АБО»), інверсія (операція «НЕ»), виключаюче «АБО».

До логічних команд іноді відносять команди порівняння, зсуву і роботи з бітами флагів.

2. Регістр флагів

Результат програми можна проаналізувати за станом регістру флагів «F».

Як і всі регістри МП КР580, регістр «F» має 8 розрядів, проте 3 розряди з них не використовуються (в них завжди одне і те ж значення, це розряди *D*1, *D*3, *D*5), а 5 розрядів цього регістру (*D*0,*D*2,*D*4,D6,D7) встановлюються по визначеному правилу у відповідності з виконанням останньої команди.

Біти ознак результату розміщаються в регістрі флагів (ознак) так, як показано на рис. 1.

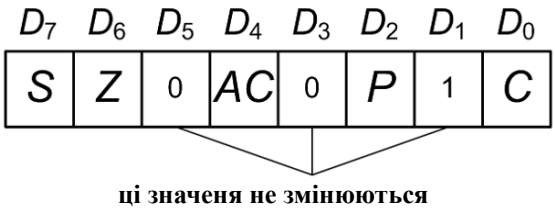


Рис. 1 Формат регістру ознак (флагів)

Флаг знаку «S» - SIGN. У нього записується «1», якщо при виконанні арифметичної або логічної команди в старшому, сьомому розряді акумулятора є «1», в іншому випадку в розряд «S» записується «0».

Приклад: Вміст регістру «А» після виконання останньої операції дорівнює:

«7Fh», «0111 1111 b» або «45h», «0100 0101 b» тоді «S» = «0»;

«80h», «1000 0000 b» або «A5h», «1010 0101 b» тоді «S» = «1».

Флаг нульового результату «Z» - ZERO. Якщо результат дорівнює нулю, в нього записується «1», в іншому випадку записується «0».

Приклад: Вміст регістру «А» після виконання останньої операції длрівнює:

«01h», «0000 0001 b» або «DFh», «1101 1111 b» тоді «Z» = «0»;

«00h», «0000 0000 b» »тоді« Z »=« 1 ».

Додатковий (допоміжний) **флаг перенесення АС** – AUXILIARY CARRY. У нього

записується 1, якщо при виконанні команд в акумуляторі виникає одиниця перенесення з третього розряду числа (перенесення між тетрадами байта).

Приклад: Вміст регістру «А» до і після виконання операції додавання з одиницею дорівнює:

до «81h», «1000 0001 b» після «82h», «1000 0010 b» тоді «AC» = «0»;

до «AFh», «1010 1111 b» після «B0h», «1011 0000 b» тоді «AC» = «1».

Флаг паритету (парності одиниць) P - PARITY. У нього записується «1», якщо при виконанні операції кількість одиниць в розрядах акумулятора буде парним, в іншому в нього записується «0».

Приклад: Вміст регістра «А» після виконання останньої операції дорівнює:

«1Fh», «0001 1111 b» або «07h», «0000 0111 b» тоді «P» = «0»;

«81h», «1000 0001 b» або «C5h», «1100 0101 b» тоді «P» = «1».

Флаг переповнення C - CARRY. У нього записується 1, якщо при виконанні останньої команди було переповнення акумулятора (перенесення з старшого розряду), в іншому випадку записується «0».

Приклад: У результаті виконання операції додавання:

«3Fh» + «0Ah» = «49h» тоді «C» = «0»;

«AAh» + «BBh» = «65h» тоді «C» = «1».

3. Команди переходів та роботи з підпрограмами

Команди переходів (умовних і безумовних), виклику підпрограм і повернення з них змінюють нормальну послідовність виконання команд і передбачають перехід за адресою, вказаною в них.

Організація умовних і безумовних переходів. В системі команд існує група команд, яка відіграє особливу роль в організації виконання програм. Це команди передачі управління.

Поки в програмі не зустрічаються команди цієї групи, лічильник команд «PC» постійно збільшує своє значення на число, що дорівнює кількості байт в поточній команді, і мікропроцесор виконує команду за командою в порядку їх розташування в пам'яті.

Порядок виконання програми може бути змінений, якщо занести в регістр лічильника команд «PC» код адреси, що відрізняється від адреси чергової команди.

Така передача управління або перехід в програмі може бути виконана за допомогою **трибайтової команди безумовного переходу «JMP adr».**

Безумовну передачу управління можна зробити також за командою «PCHL» (яка формально відноситься до групи команд пересилань), в результаті виконання якої відбудеться передача управління за адресою, що зберігається в реєстровій парі «H, L».

Крім команди безумовного переходу є вісім трибайтових команд умовного переходу. При появі команди умовного переходу, передача управління за адресою, вказаною в команді, відбувається тільки в разі виконання певної умови.

Якщо умова не задовольняється, то виконується команда, наступна за командою умовного переходу.

Умови, з якими оперують команди умовної передачі управління, визначаються станом регістра прапорів «F».

Зв'язок між станом регістру ознак і мнемонікою команди умовного переходу:

..NZ (NOT ZERO) - ненульовий результат при «Z» = 0,

..Z (ZERO) - нульовий результат при «Z» = 1,

..NC (NO CARRI) - відсутність перенесення при «C» = 0,

..C (CARRY) - перенесення при «C» = 1,

..PO (PARITY ODD) - непарний паритет при «P = 0»,

..PE (PARITY EVEN) - парний паритет при «P = 1»,

..P (PLUS) - число невід'ємне при «S = 0»,

..M (MINUS) - число від’ємне при «S = 1».

Доповнивши вищенаведену мнемоніку частиною від команди «JMP» - перехід,

отримаємо відповідні команди умовного переходу: «JNZ», «JZ», «JNC», «JC», «JPO», «JPE», «JP», «JM».

Команди роботи з підпрограмами. Ці команди, так само як і команди переходів змінюють нормальну послідовність виконання операцій в процесорі. Відмінність полягає в тому, що при виклику підпрограми (наприклад, за допомогою команди «CALL adr») використовується спеціальна область пам'яті - СТЕК.

В СТЕКу зберігається адреса повернення (адреса ділянки, на якій була викликана підпрограма), що дозволяє при поверненні з підпрограми використовувати безадресні команди (наприклад, команду «RET»).

Аналогічно, як і для команди переходів, викликати підпрограму і повернутися з неї можна за умовою.

Мнемонічні правила написання відповідних команд ідентичні вищевикладеним правилам.

4. Команди роботи зі стеком, вводу-виводу і загальні команди

Команди роботи зі стеком призначені для роботи зі спеціальною областю пам'яті, самі команди є однобайтними і мають мнемоніку «PUSH ...» - завантажити в СТЕК і «POP ...» - витягти з стека.

Як було зазначено вище, СТЕК є область пам'яті із спеціалізованим механізмом звернення (через регістр - покажчик «SP»). При кожному зверненні до стеку відбувається автоматична зміна вмісту покажчика адреси комірки пам’яті в стек «SP»: при завантаженні - зменшується, при зчитуванні - збільшується.

Команди вводу-виводу призначені для виконання операцій пов'язаних з портами мікропроцесора: команда вводу з порту N з мнемонікою «IN N» і команда виводу з порту N з мнемонікою «OUT N». **Робота з портом відбувається через акумулятор.**

Команди управління і загальні команди призначені для організації роботи системи.

Наприклад, команда рестарту «RST n», команди дозволу і заборони переривань «EI», DI», команди зупинки «HLT».

5. Алгоритмізація етапів виконання складних програм

Під алгоритмізацією розуміється зведення задачі до послідовності етапів, які виконуються один за одним так, щоб результати попередніх етапів використовувалися при виконанні наступних.

Таким чином, в результаті алгоритмізації повинен бути отриманий алгоритм вирішення задачі.

Алгоритм - це точний і зрозумілий опис послідовності дій над заданими об'єктами (вихідними даними), що приводить виконавця після кінцевого числа кроків до досягнення зазначеної мети (отримання результату) або вирішення поставленого завдання.

Алгоритм має наступні основні властивості: дискретність, визначеність, результативність і масовість.

Дискретність: в алгоритмі процес перетворення вихідних даних в результат здійснюється дискретно, по кроках. Перехід до наступного кроку можливий лише після завершення попереднього.

Визначеність: алгоритм повинен бути чітким і однозначним; значення величин одержуваних в будь-який момент часу однозначно визначаються величинами, отриманими в попередні моменти часу; в рамках алгоритму чітко визначається, яка мета повинна бути виконана наступною.

Масовість: алгоритм вирішення задачі розробляється в загальному вигляді так, що б його можна було застосувати для цілого класу задач, які відрізняються вихідними даними.

Результативність: алгоритм повинен призводити до вирішення завдання за кінцеве

число кроків; під рішенням так само розуміється наявність повідомлення про те, що при заданих даних завдання рішення не має або алгоритм не прийнятний.

5.1. Основні етапи виконання алгоритму

Ключовим елементом будь-якого алгоритму є «дія», яке відбувається на певному етапі. Одним з варіантів представлення алгоритмів є схема.

Наведемо основні «дії» і їх позначення, що мають місце при вирішенні складних задач.

До базових етапів, що визначають елементарні дії, зазвичай відносять: початок, ввід і вивід даних, обробка значень, перевірка умови і переходу, закінчення або виходу.

1. Етап початку алгоритму, або термінатор відображає вхід із зовнішнього середовища (початок схеми програми):

ПОЧАТОК (ВХІД).

2. Ввід початкових даних, вивід інформації (покажчик на дані, носій яких не визначений) відображає використання, джерело і приймач даних, допоміжну вказівну інформацію:

ДАНІ

3. Етап обробки обчислення або процесу, який слід виконати над даними:

V = Вирази

де V - змінна; Вирази - будь-яке обчислення, яке треба здійснити.

4. Етап перевірки умови (рішення) - це функція перемикаючого типу, що має один вхід і ряд альтернативних виходів, тільки один з яких може бути активований після перевірки умов:

УМОВА?

Якщо умова виконується, то здійснюється перехід до етапу з номером (Міткою) «N»; якщо умова не виконується, то перехід здійснюється до наступного по порядку етапу.

Перехід до етапу з номером N:

ЙТИ ДО «N»

5. Етап закінчення: відображає кінець виконання або вихід у зовнішнє середовище:

ВИМИКАННЯ (ВИХІД).

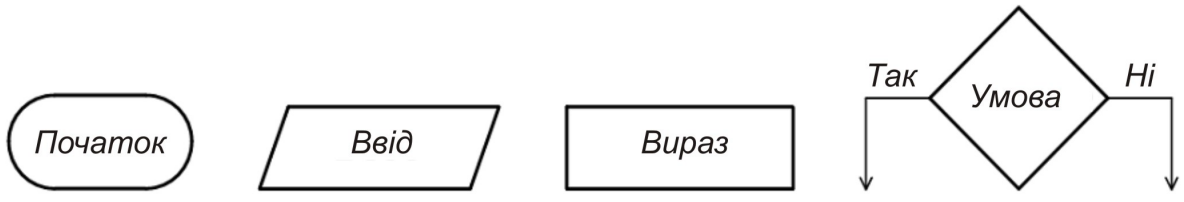
У схемі кожна з «дій» має своє умовне зображення (див. рис. 1), з'єднується з іншими за допомогою ліній (що показують напрямок руху за алгоритмом) і в разі необхідності супроводжується коментарями.

Початок і закінчення алгоритму відображається, як показано на рис. 1, а, у вигляді овалу, всередині якого вказується «початок» (вхід) або «зупинення» (вихід).

Ввід або вивід даних, джерело або їх приймач, зображується за допомогою паралелограму (див. рис. 1, б), усередині якого вказується «ввід» або «вивід», «друк» і перераховуються змінні.

Етап обробки обчислення позначаються прямокутником (див. рис. 1, в), всередині якого записується зміст дії (описується вираз).

Перевірка умови зображується ромбом, усередині якого записується формулювання. В результаті перевірки вибирається один з двох можливих шляхів обчислювального процесу (рис. 1, г): якщо умова виконується, тобто має значення «ТАК», то наступним виконується перехід за відповідним напрямком (аналогічно для переходу по гілці «НІ»).



а) б) в) г)

Рис. 1 Умовне позначення етапів алгоритму

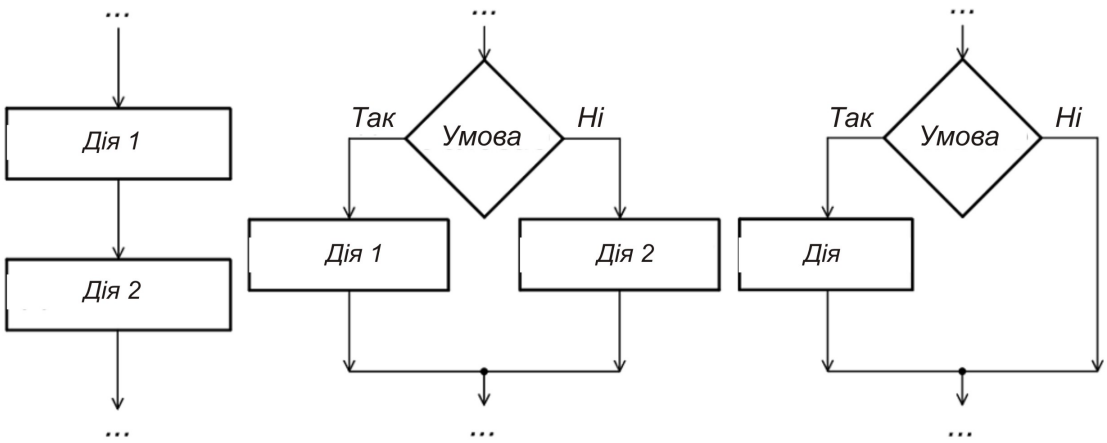
5.2. Складання алгоритмів складних завдань

При розробці алгоритмів вирішення завдань найчастіше дотримуються так званого структурного підходу: використовують метод послідовного уточнення алгоритму, застосовують допоміжні алгоритми, описують алгоритми за допомогою трьох основних структур: слідування, розгалуження, повторення.

Схема слідування застосовується при виконанні виразів, що йдуть один за одним - див. рис. 2, а.

Схема розгалуження застосовується, коли в залежності від умови потрібно виконувати або одну, або іншу дію (вираз). Просте розгалуження показано на рис. 2, б; окремий випадок розгалуження, званий іноді **обходом**, показаний на рис. 2, в.

В обох випадках напрямки руху за алгоритмом «Так» і «Ні» показані умовно (може бути і навпаки).



а) б) в)

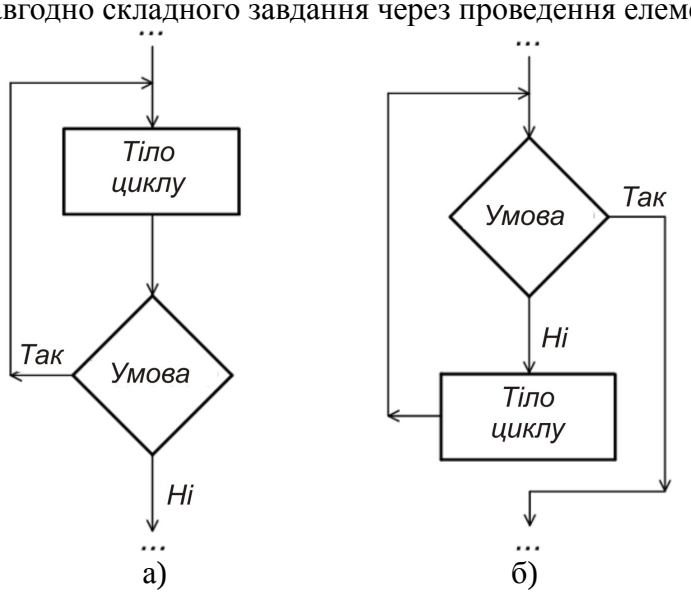
Рис. 2 Схема слідування і розгалуження

Примітка: При програмуванні задач на асемблері для організації «умов» використовують команди умовного переходу.

Схема повторення є ключовою при організації циклів алгоритму.

Такі схеми називають **циклами.** Виділяються окремо так звані цикли «До» і цикли «Поки». Різниця між цими двома схемами полягає в місці розташування умови: в циклі «До» (виконувати до умови) умова розташовується в кінці циклу (див. рис. 3, а), в циклі «Поки» умова розташовується на початку циклу (див. рис. 3, б).

Об'єднуючи вищенаведені структури, дотримуючись принципу уточнення, можна отримати вирішення як завгодно складного завдання через проведення елементарних дій.



а) б)

Рис. 4 Схеми повторення

6. Порядок роботи

Завдання 1. Дослідження програми обробки масиву.

Нехай потрібно знайти суму елементів масиву (чисел без знаку), розташованого, починаючи з адреси «0B02h».

Кількість оброблюваних елементів масиву міститься за адресою «0B01h».

Результат обробки необхідно помістити в комірку пам'яті з адресою «0B00h».

Загальний алгоритм вирішення такого завдання може бути поданий у вигляді рис. 4.

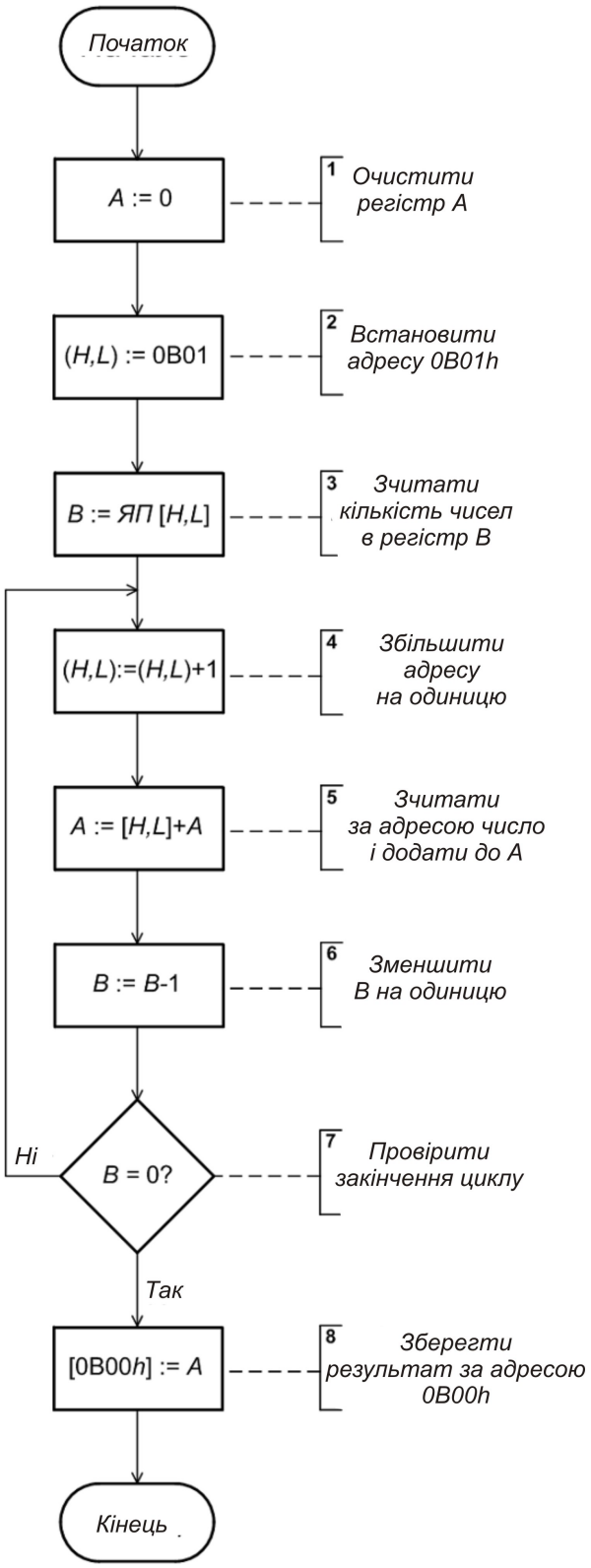


Рис. 4. Алгоритм програми 2

Спочатку потрібно виконати «початкові перетворення»: очистити регістри, використання яких передбачається (за потребою; при цьому бажано використовувати команди, що виконуються за меншу кількість тактів); встановити початкові адреси комірок пам'яті, зчитати першу комірку (кількість елементів).

У самому тілі циклу потрібно:

- збільшити адресу на одиницю (адресувати таку комірку пам'яті і перейти до чергового оброблюваного елементу);
- сумувати вміст одного з регістрів (що відповідає за накопичення суми) з черговим елементом масиву;
- зменшити кількість оброблюваних елементів на одиницю.

В кінці циклу необхідно перевірити умову закінчення: чи всі елементи оброблені (чи дорівнює лічильник елементів нулю), а якщо умова не виконана, перейти в початок циклу.

Якщо умова виконується необхідно здійснити завершальні дії: зберегти загальну суму за необхідною адресою.

Таким чином, в тілі циклу повторюються одні й ті ж дії: перехід до чергового елементу (комірки пам'яті), додавання чергового елементу до загальної суми, модифікація і перевірка умови закінчення обробки.

Зауважимо, що до «першого проходу циклу» вміст регістру, який містить загальну суму, має бути скинутий в нуль, тому що до нього в тілі циклу додається чергове число.

Тобто, при «першому проході» в регістрі міститься тільки перший елемент (сума нуля і чергового числа, яке зараз є першим), потім до цього регістру на «другому проході» додається другий елемент (раніше отримана сума і черговий елемент, який зараз буде другим), і т.д.

Програма, складена по даному алгоритму, представлена у вигляді таблиці 1.

Таблиця 1. Програма 2

№ етапу	Коментар по етапу алгоритму	Мітка	Мнемокод	Коментар до команди
1	Очистити регістр “A”		`SUB A`	Призначення команди *SUB*: Від міcту регістру “A” віднімається вміст регістру “A”, результат завантажується в “A”
2	Встановити адресу “0B01*h*”		`LXI H,0B01h`	Призначення команди *LXI*: Безпосереднє завантаження пари регістрів “*H*”,”*L*”.
3	Зчитати кількість чисел в регістр “B”		`MOV B,M`	Призначення команди *MOV*: Вміст комірки пам’яті, адреса якої знаходиться в парі регістрів “*H*”,”*L*”, пересилається в регістр “B”.
4	Збільшити адресу на “1”	Mk1:	`INX H`	Призначення команди *INX*: Вміст пари регістрів “*H*”,”*L*”, збільшується на “1”.
5	Число з “адреси” додати до вмісту “A”		`ADD M`	Призначення команди *ADD*: Вміст комірки пам’яті, адреса якої знаходиться в парі регістрів “*H*”,”*L*”, сумується з вмістом “A”, результат завантажується в “A”
6	Зменшити “B” на “1”		`DCR B`	Призначення команди *DCR*: Вміст регістру “B” зменшується на “1”.
7	Перевірити закінчення циклу		`JNZ Mk1`	Призначення команди *JNZ*: Якщо останній результат не рівний “0”, то перехід за міткою “*Mk1*”
8	Зберегти результат за адресою “0B00*h*”		`STA 0B00h`	Призначення команди *STA*: Вміст “A” пересилається за адресою в комірку пам’яті, що вказана в команді
-			`HLT`	Призначення команди *HLT*: Зупинка

В рамках виконання даного завдання потрібно:

а) скласти програму по визначенню суми елементів масиву (аналогічно табл. 1), враховуючи зміни для кожного варіанту завдання у вигляді табл. 2;

б) підготувати програму до запису в емулятор, переводячи її в машинний код. При цьому необхідно скласти таблицю, ідентичну табл. 2, яку так само подати в звіті по роботі);

в) занести створену програму в ОЗП емулятора, досліджувати її (при цьому елементи масиву вибрати на свій розсуд, але їх кількість повинна бути не менша 41), проаналізувати результат, зробити висновок про роботу, окремо розглянути питання функціонування системи і зміни вмісту регістра стану (флагів), склавши і заповнивши табл. 3 для по-командного режиму роботи емулятора.

Таблиця 2

Варіанти завдань для Програми 2

Призначення адреси пам'яті	Значення адреси комірок пам'яті відповідно до варіантів, *h*														
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Кількість оброблюваних чисел	0B11	0B21	0B31	0B41	0B51	0B61	0B71	0B81	0B91	0BA1	0BB1	0BC1	0BD1	0BE1	0BF1
Початок розташування масиву чисел	0B12	0B22	0B32	0B42	0B52	0B62	0B72	0B82	0B92	0BA2	0BB2	0BC2	0BD2	0BE2	0BF2
Місце розташування результату	0B10	0B20	0B30	0B40	0B50	0B60	0B70	0B80	0B90	0BA0	0BB0	0BC0	0BD0	0BE0	0BF0

Таблиця 3

Результат виконання Програми 2 по командах

Адреса, *h*	**Мнемокод**	**Кількість байт в команді**	**Машинний код, *h***	**Біти регістра флагів**								**Пояснення по факту зміни бітів регістра флагів і використання цього в програмі**
				***D*7**	***D*6**	***D*5**	***D*4**	***D*3**	***D*2**	***D*1**	***D*0**	
				S	***Z***	**0**	***AC***	**0**	***P***	**1**	**C**	
0800	*SUB* A	1	97									
…	…	…	…	…								

У табл. 3 допустимо пропускати повторювані елементи (при проходженні циклу).

Контрольні запитання

1. Які флаги не змінюють логічні команди?
2. Які групи команд умовно відносять до логічних?
3. Наведіть побітовий зміст регістру ознак мікропроцесора.
4. Яким чином використовуються флаги (ознаки)?
5. Які команди переходів Ви знаєте?
5. Скільки байт міститься в командах переходів і роботи з підпрограмами?
6. Яким чином відбувається повернення з підпрограми?
7. Як може використовуватися СТЕК в мікропроцесорних системах?
8. Дайте визначення алгоритму.
9. Якими властивостями володіє алгоритм?
10. Покажіть основні етапи алгоритмів у вигляді схем і прикладів на Асемблері.

Список рекомендованої літератури

1. Самофалов К. Г., Викторов О. В. Микропроцессоры. – Б-ка инеженера – 2-е изд., перераб. и доп. - К: Техника, 1989.-312 с.
2. Шевкопляс Б.В. Микропроцессорные структуры. Инженерные решения: Справочник. 2-е изд., перераб. и доп. -М.: Радио и связь, 1990, -512с.
3. Угрюмов Е.П. Цифровая схемотехника. Спб.: BHV, 2001. 528 с.
4. Злобин В. К-, Григорьев В. Л. 58 Программирование арифметических операций в микропроцессорах: Учеб. пособие для технических вузов.— М.: Высш. шк., 1991. —303 с.: ил.