МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторних робіт
"Використання мови асемблера для розробки застосувань"
з дисципліни
"Архітектура обчислювальних систем"

для студентів спеціальностей
122 – Комп'ютерні науки та інформаційні технології,
124 – Системний аналіз, 186 – Видавництво та поліграфія

Затверджено
Редакційно-видавничою
радою університету,
протокол № 2 від 23.06.17

Харків
НТУ «ХПІ»
2018

Методичні вказівки до лабораторних робіт «Використання мови асемблера для розробки застосувань» з дисципліни "Архітектура обчислювальних систем" для студентів спеціальностей 122 – Комп'ютерні науки та інформаційні технології, 124 – Системний аналіз, 186 – Видавництво та поліграфія / Уклад. Ю. М. Кожин, О. М. Малих, В. П. Прокопенков. – Харків.: НТУ "ХПІ", 2018. – 64 с.

Укладачі: Ю.М. Кожин,
О.М. Малих,
В.П. Прокопенков

Рецензент М.І. Безменов

Кафедра системного аналізу та інформаційно-аналітичних технологій

Вступ

Сучасне програмне забезпечення (ПЗ) стає все більш складним і об'ємним. Проектування складних програмних продуктів вимагає використання систем програмування з достатніми засобами автоматизації і контролю якості розробки.

При розроблені програми, до якої висуваються високі вимоги з швидкодії, можна використовувати мову низького рівня – асемблер. Мова асемблера дозволяє використати усі апаратні можливості процесора з обробки даних і забезпечує побудову програмного модуля з мінімальним кількстью команд і максимальною швидкодією.

У даних методичних вказівках розглядаються остовні команди процессора X86 для роботи з масивами даних, обчислення значень за формулою з використанням цілих та дійсніх чисел, організації підпрограм.

Дані методичні вказівки укладені з метою допомогти студентам оволодіти технологією використання мови низького рівня при розроблені застосувань у середовище програмування *Visual Studio.*

1. КОМАНДИ ПЕРЕДАЧІ ДАНИХ І АДРЕСАЦІЯ ДО ПАМ'ЯТІ

Мета роботи : освоєння прийомів пересилки даних між елементами пам'яті з використанням команд команди передачі даних по байтах або по словах, а також освоєння режимів адресації до пам'яті обчислювача.

1.1. Команди передачі даних

Зазвичай команди пересилки даних найчастіше використовуються у наборі команд будь–якою ЕОМ. Система команд процесора X86 має великий набір команд передачі даних між регістрами та пам'яттю обчислювача.

1.1.1. Команда передачі даних MOV

Команда *MOV* – основна команда пересилки даних, яка записує байт, слово або подвійне слово даних з пам'яті у регістр, з регістра у пам'ять або з регістра у регістр. Команда *MOV* може також занести число, визначене програмістом у вигляді константи, в регістр або у пам'ять обчислювача.

Загальна форма команди *MOV* має вигляд:

MOV приймач, джерело

Як джерело і приймач можуть бути використані операнди, подані у табл. 1.1.

Таблиця 1.1 – Джерела і приймачі даних у командах пересилки

Приймач	Джерело			
	Reg	*Sr*	*Const*	*Memory*
Reg	+	+	+	+
Memory	+		+	
Sr	+			

У табл. 1.1 використані такі позначення:

Reg – байтові регістри *AH, AL, BH, BL, CH, CL, DH, CL,* 16 – розрядні регістри *AX, BX, CX, DX, SP, BP, SI, DI,* 32 – розрядні регістри *EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI*;

Sr – сегментні регістри *SS, ES, DS, FS, GS* ;

Memory – елемент пам'яті;

Const – безпосередньо задана константа.

У команді пересилки число байт джерела і приймача повинно збігатися.

Крім того, у команді неприпустимо як джерело і приймач мати елементи пам'яті. Для пересилки даних з одного елементу пам'яті у інший потрібно дві команди: одна записує з оперативної пам'яті дані у регістр центрального процесора, друга переписує вміст регістра в інший елемент пам'яті.

Наприклад, пересилка даних зі змінної з ім'ям *COUNT* у змінну *Digit* (розмір даних 4 байти) може бути здійснена за допомогою команд:

MOV EAX , COUNT

MOV Digit , EAX

1.1.2. Команда обміну даних XCHG

Команда обміну даними *XCHG* міняє місцями значення двох операндів: двох регістрів або регістра і елементу пам'яті.

Команда обміну даними має вигляд:

XCHG приймач, джерело

Можливі джерела і приймачі аналогічні команді MOV.

Наприклад, у команді

XCHG Digit, EAX

значення, що зберігається у змінній *Digit*, записується в *EAX*, а значення у регістрі *EAX* у змінну *Digit.*

1.1.3. Завантаження адреси

Для виконання операції з непрямою адресацією потрібна адреса елементу пам'яті, в якій розміщується операнд. Команда завантаження адреси *LEA* дозволяє отримати таку адресу.

Синтаксис команди:

LEA Reg, джерело,

де *Reg* – один з 32-розрядних регістрів центрального процесора.

Команда *LEA* дуже схожа на команду *MOV*. При цьому приймачем може бути регістр, а джерелом – елемент пам'яті з будь-яким способом адресації. Замість пересилки даних, команда *LEA* завантажує в регістр адресу даних.

Наприклад, команда

LEA ESI, EXWORD

записує в регістр *ESI* адресу розміщення змінної *EXWORD* у пам'яті обчислювача.

Отримати адресу розміщення даних у пам'яті обчислювача можна за допомогою оператора *OFFSET* в команді *MOV* з безпосереднім операндом. Наприклад, команда

MOV EBX, OFFSET EXWORD

завантажує в регістр *EBX* адресу розміщення змінної *EXWORD* у пам'яті обчислювача.

1.1.4. Пересилка прапорів процесора

Набір команд процесора X86 має команди *LAHF* і *SAHF* для роботи з регістром прапорів. Команда *LAHF* бере 8 молодших біт регістра прапорів (ці прапори співпадають з прапорами мікропроцесора 8080) і записує їх у регістр *AH*. Команда *SAHF* молодший байт регістра прапорів завантажує з регістра *AH*. Обидві команди не мають операндів.

1.1.5. Заміна байт

Команда *XLAT* перетворить коди символів відповідно до таблиці. Таблиця об'ємом 256 байт містить нові коди символів. Адреса таблиці повинна знаходитися у регістрі *EBX*. Команда *XLAT* замінює значення в регістрі *AL* на код символу з номером, який був записаний *AL*. Команду *XLAT* можна використати при кодуванні і декодуванні текстових даних.

1.1.6. Команди роботи із стеком

До команд передачі даних відносяться також команди завантаження у стек і передачі даних із стека. При роботі із стеком використовується покажчик стека *ESP*, який зменшується при записі у стек і збільшується при вивантаженні даних із стека. Розмір стека замовляється директивою

STACK N,

де *N* – кількість байт (число має бути парним).

Завантаження даних у стек здійснюється командою

PUSH джерело

Як джерело може виступати 16 - або 32-розрядний регістр (у тому числі і сегментний) або елемент пам'яті.

Пересилка даних із стека здійснюється командою

POP приймач

Приймачем даних також може виступати 16 - або 32-розрядний регістр (у тому числі і сегментний) або елемент пам'яті.

1.1.7. Команди введення і виведення

Для виконання операцій введення від зовнішніх пристроїв і виведення на зовнішні пристрої процесор має команди *IN* та *OUT* відповідно.

Команда *IN* пересилає дані від зовнішнього пристрою в регістр *AL*. Ця команда може вказати адресу пристрою двома різними способами. Якщо адреса пристрою знаходиться в межах 0 – 255, він може міститися в команді як безпосереднє значення. Для завдання адреси зовнішнього пристрою в межах 0 – 65535 використується регістр *DX*.

Аналогічно працює команда *OUT*, за винятком того, що вона передає значення регістра *AL* зовнішньому пристрою. Адреси у команді *OUT* вказуються так само, як і в команді *IN*.

Синтаксис команд введення-виведення має вигляд:

IN const

IN DX

OUT const

OUT DX,

де const ціле значення в межах від 0 до 255.

1.2. Адресація до пам'яті обчислювача

Для доступу до оперативної пам'яті і вибірки даних з неї можна використати наступні методи адресації: неявний, прямий і непрямий.

1.2.1. Неявна адресація

Неявна адресація припускає відсутність адресної частини (операнда) в коді команди. При цьому сама операція визначає джерела та/або приймачі даних. Прикладом таких команд є команди *XLAT* або *LAHF*, які не мають операндів.

1.2.2. Пряма адресація

Пряма адресація дозволяє задати у коді команди адреси джерела та/або приймача інформації. Наприклад:

MOV AX, DIGIT

має пряму адресу операндів – регістру *AX* та змінної *DIGIT*.

1.2.3. Непряма адресація

Непряма адресація припускає наявність у коді команди вираження, значення якого задає адресу розміщення даних. Незважаючи на збільшення довжини коду команди і збільшення часу звернення до даних цей

спосіб адресації є найгнучкішим. Процесор забезпечує наступні режими непрямої адресації до пам'яті:

- регістрова;
- індексна або базова зі зміщенням;
- індексно-базова;
- індексно-базова зі зміщенням;
- індексна або базова зі зміщенням і масштабуванням;
- індексно-базова з масштабуванням;
- індексно-базова зі зміщенням і масштабуванням.

Непряма регістрова адресація використовується для звернення до елементу пам'яті, адреса якої зберігається в регістрі. Адресним регістром можуть виступати будь-які 32–розрядні регістри (регістри *EBP* і *ESP* за умовчанням використовують сегмент стека, а інші регістри – сегмент даних). Команда

MOV [*ESI*],*AX*

завантажує значення акумулятора у область пам'яті за адресою, яка міститься у регістрі *ESI*. Квадратні дужки вказують на використання непрямій адресації.

Індексна або базова зі зміщенням забезпечує використання константи і значення регістра для визначення адреси пам'яті. Відмінність індексної адресації від базової полягає в іншій інтерпретації вмісту адресних регістрів. У першому випадку вміст регістра розглядається як індекс або число зі знаком (робота з масивами), у другому – як база або число без знака (зміщення відносно початку сегмента). При обчисленні адреси розміщення змінної процесор складає дві компоненти: константу і значення регістру (зміщення і базу або базу і індекс). Отримане значення використовується для звернення до пам'яті.

Індексно-базова адресація забезпечує формування адреси з двох частин, які зберігаються у регістрах. Один з них містить базу, а другий індекс (зміщення). На рис. 1.1 показано формування повної адреси при використанні індексно–базової адресації у команді пересилки даних.

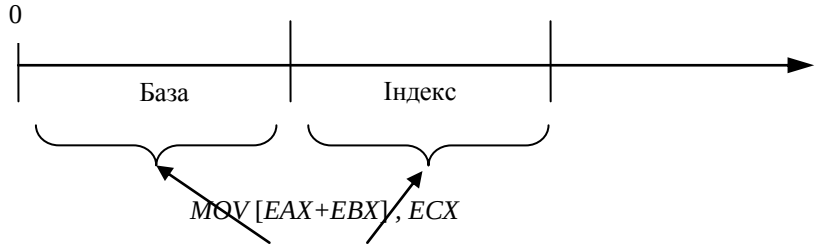

Рисунок 1.1 – Формування адреси за допомогою індексно–базової адресації

База розглядається як ціле число без знака, а індекс – як число зі знаком або без знака.

Індексно–базова адресація зазвичай використовується для доступу до елементів масиву в підпрограмах, коли адреса масиву передається через параметр (адреса початку масиву передається через стек).

Індексно-базова адресація зі зміщенням (рис. 1.2) використовується при адресації до даних, що мають складну внутрішню структуру. Наприклад, для звернення до багатовимірних масивів або до полів масиву структур. Адреса, що формується процесором, складається з трьох компонентів: константи (задається у команді), бази та індексу. Для завдання адреси використовуються ті ж регістри, що і в індексно-базовій адресації.

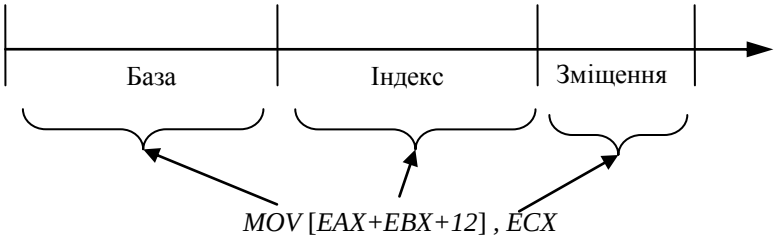

Рисунок 1.2 – Формування адреси за допомогою індексно–базової адресації зі зміщенням

Значення регістру *EAX* задає зміщення відносно початку сегмента до масиву даних, базовий регістр *EBX* – зміщення елементу в масиві (індекс), а константа 12 – зміщення до необхідного поля відносно початку розміщення елементу.

Процесор дозволяє використовувати масштабування індексного регістра в режимах з непрямою адресацією (рис. 1.3.). При масштабуванні

величина індексного регістру множиться на 2, 4 або 8 і отримане значення використовується для формування адреси.

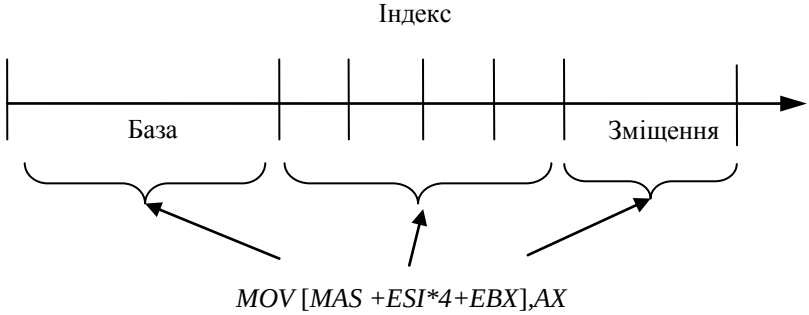

Рисунок 1.3– Режим адресації з масштабуванням

Як індексний регістр може бути використаний будь-який з 32-розрядних регістрів процесора.

Режими адресації з масштабуванням зручні для роботи з масивами числових даних.

1.3. Приклад використання команд передачі даних

Розглянемо приклад розроблення програми для шифрування рядка символів шляхом перестановки символів.

Черговий символ рядка записується у позицію результуючого рядка на місце відповідно до масиву перестановок.

Початковий рядок символів зберігається у змінної *Simbol*, рядок з результатом виконання перестановки символів у змінної *Shifr*. Масив *mesto* задає позицію кожного символу вихідного рядка в результуючому рядку.

Нижче наведений текст програми для перестановки перших чотирьох символів. У програмі для доступу до початкового та результуючого рядків використовується індексна адресація зі зміщенням, а для доступу до масиву перестановок – індексна зі зміщенням і масштабуванням.

```
.586
.model flat
.stack 100h
.data
```

```
Simbol   DB " Мама мила раму "
Shifr    DB   2 DUP(7 DUP (" "))
mesto    dd    2, 3, 0, 1
.code
ExitProcess PROTO STDCALL :DWORD
start:
mov ebx,0
mov al , Simbol [ebx]
mov edx , mesto [ebx*4]
mov al , Simbol [ebx]
mov Shifr [edx] , al
inc ebx
mov al,Simbol [ebx]
mov edx,mesto [ebx*4]
mov al,Simbol [ebx]
mov Shifr [edx] , al
inc ebx
mov al , Simbol [ebx]
mov edx , mesto [ebx*4]
mov al , Simbol [ebx]
mov Shifr [edx] , al
inc ebx
mov al , Simbol [ebx]
mov edx , mesto [ebx*4]
mov al , Simbol [ebx]
mov Shifr [ebx] , al
Invoke ExitProcess,0
 End Start
```

1.4. Порядок виконання роботи

1. Розробити структура даних програми (табл.1.2).
2. Написати програму що реалізовує побудову рядка символів із заданого набору символів відповідно до варіанта з використанням прямої, базової, індексно–базової адресації.
3. Перевірити роботу програми.
4. Оформити звіт

Таблиця 1.2 – Варіанти завдань

Варіант	Набор символів	Результат
1	йкиочпцані	міцний
2	аятимпрцка	матриця
3	цнеипдіхлкещро	перехід
4	фякцаьрштитн	картина
5	ьюблерымоклдт	мольберт
6	тнлюкафрдіше	тарілка
7	нцшяєиеьрёфйо	шифоньєр
8	ейщбсярчкшта	батарея
9	антбисякшетзьр	скатертина
10	амвколиіэрдтху	клавіатура

Контрольні запитання

1. Для чого використовується команда *MOV*?
2. Скільки операндів вимагає команда *MOV* і якими мають бути розміри операндів для виконання операції пересилки?
3. Скільки команд знадобиться для передачі даних з одного елементу пам’яті у інший?
4. На які прапори впливає команда пересилки даних?
5. У чому особливість команди обміну даними *XCHG*?
6. За допомогою якої команди можна отримати адресу розміщення даних у пам’яті обчислювача?
7. Як здійснити доступ до прапорів центрального процесора?
8. Як виконується команда *XLAT* перекодування даних?
9. Що таке неявна адресація?
10. У якому випадку використовується пряма адресація?
11. У чому переваги і недоліки прямої адресації?
12. Що таке непряма адресація?
13. Які регістри можуть брати участь у формуванні адреси у непрямій адресації?
14. У чому відмінність базової адресації від індексної?
15. Чи може використовуватися операція віднімання при формуванні адреси з непрямою адресацією?
16. Який масштаб можна вибрати в адресації з масштабуванням?

2. ОБЧИСЛЕННЯ ВИРАЗІВ ЦІЛОЧИСЕЛЬНОЇ АРИФМЕТИКИ

Мета роботи: вивчити систему арифметичних команд, отримати навички використання арифметичних операцій для обробки числових даних.

2.1. Арифметичні команди

У набір арифметичних команд включені основні операції обробки числових даних: складання, віднімання, множення і ділення.

2.1.1. Команда складання

Команда *ADD* виконує складання вказаних операндів, представлених у двійковому додатковому коді. Синтаксис команди має вигляд:

ADD приймач, джерело

Допустимі поєднання джерел і приймачів приведені в табл. 2.1

Таблиця 2.1 – Пеєднання джерел і приймачів

Приймач	Джерело		
	Reg	*Const*	*Memory*
Reg	+	+	+
Memory	+	+	

Reg – регістри *AH,AL,BH,BL,CH,CL,DH,CL* (байтові регістри), *AX, BX, CX, DX, BP, SI, DI* (16-розрядні регістри), *EAX, EBX, ECX, EDX, EBP, ESI, EDI* (32-розрядні регістри);

Memory – елемент пам'яті;

Const – безпосередньо задана константа.

Процесор поміщає результат на місце першого операнда після того, як складе обидва операнди. Другий операнд не змінюється.

При виконанні арифметичних команд операнди повинні мати однаковий розмір.

Команда коригує регістри прапорів відповідно до результату складання.

Команда складання з перенесенням *ADC* аналогічна команді *ADD*, за винятком того, що в суму включається розряд перенесення. Така команда використовується для організації багатобайтового складання (складання довгих числових даних, для обробки яких розміри регістрів недостатні).

2.1.2. Команда віднімання

Команди *SUB* і *SBB* виконують віднімання двох операндів.

Синтаксис команд:

SUB приймач, джерело

SBB приймач, джерело

Прапори при відніманні встановлюються аналогічно командам складання, за винятком прапора перенесення. Прапор перенесення формується як одиниця позики при обробленні старших розрядів операндів.

Команда віднімання з позикою *SBB* вирішує віднімання числових даних підвищеної точності. При виконанні команди *SBB* значення позики, отриманої при відніманні молодших розрядів, віднімається від результату при обробленні старших розрядів.

Можливі операнди команди віднімання аналогічні операндам команди складання (табл. 2.1).

2.1.3. Порівняння двох цілих чисел

Команда порівняння *CMP* порівнює два числа, віднімаючи одне з другого (як і команда *SUB*). Вона не записує результат, але встановлює прапори відповідно до результату.

Синтаксис команди *CMP*:

CMP приймач, джерело.

2.1.4. Арифметичні команди з одним операндом

Команда *NEG* забезпечує зміну знаку числа. Операндом можуть виступати значення у регістрі або елемент пам'яті. При виконанні команди завжди встановлюється перенесення *C*=1. Якщо операнд дорівнює 0 прапор *C* встановлюється в 0.

Якщо значення операнда дорівнює нижній межі (80H, 8000H або 80000000H), перетворення не виконується (значенню не відповідає позитивне число). При цьому значення прапора *O* встановлюється в 1.

Команда збільшення *INC* додає 1, а команда зменшення *DEC* віднімає 1 з операнда. Ці команди також можуть реалізувати лічильник циклу. Команди не змінюють значення прапора перенесення. Це забезпечує можливість організації циклів при обробленні багатобайтових числових даних.

Ці три команди впливають на біти регістра прапорів так само, як і арифметичні команди.

Синтаксис команд:

INC приймач

DEC приймач

NEG приймач

Приймачем може виступати операнд аналогічно команді складання.

2.1.5. Команди множення

Існує два різновиди команд множення – зі знаком *IMUL* і без знака *MUL*.

Обидві команди множення працюють з байтами, із словами і подвійними словами.

При множенні 8-розрядих даних, один з операндів повинен знаходитися в регістрі *AL*, другий задається в адресній частині команди. Результат має довжину 16 розрядів і розміщується в регістрі *AX*.

Щоб помножити 16-розрядні числа, один з операндів треба помістити в регістр *AX*, другий – задається в команди. Результат має довжину 32 біт і поміщається в пару регістрів – у регістрі *DX* містяться старші 16 біт результату, а в регістрі *AX* – молодші 16 розрядів.

Для множення 32-розрядних чисел перший співмножник міститься в регістрі *EAX*, другий – вказується в команді. Результат множення записується в *EAX* молодші 32 розряди, в *EDX* – старші 32 розряди.

Синтаксис команд має вигляд

MUL Reg

MUL [*Memory*]

IMUL Reg

IMUL [*Memory*] ,

де *Reg* – регістр процесора, *Memory* – елемент пам'яті, який містить другий співмножник.

Команда *MUL* при множенні чисел без знаку встановлює прапори переповнення та перенесення, якщо старша половина результату не нульова. Команда *IMUL* встановлює ці прапори, якщо старша частина результату не є розширенням знака молодшої частини результату.

Починаючи з процесора X386, команда множення має розширений синтаксис. У коді команди можна вказувати два або три операнди.

MUL Reg,Reg

MUL Reg,[*Memory*]

MUL Reg,Reg,const

MUL Reg, [*Memory*] *,const*

IMUL Reg,Reg

IMUL Reg, [*Memory*]

IMUL Reg,Reg,const

IMUL Reg, [*Memory*] *,const*

Перший операнд є регістром, куди записується результат множення. Вміст цього регістра використовується як перший співмножник. Другий операнд може бути або регістр, або елемент пам'яті. Розміри обох операндів мають бути однакові (байт, слово або подвійне слово). Якщо вказується третій операнд, то це може бути константа, яка множиться до результату множення перших операндів.

Використання команд множення з розширеним синтаксисом має особливості. Якщо результат множення не може розміститися у приймачі, процесор установлює прапор переповнювання в 1, а приймач містить молодші розряди результату множення.

2.1.6. Команда ділення

Існує дві форми ділення: одна – для двійкових чисел без знака *DIV*, а друга *IDIV* – для чисел зі знаком у додатковому коді. Будь-яка форма ділення може працювати з байтами, словами і подвійними словами.

Команда ділення *DIV* виконує ділення без знака і дає як частку, так і залишок. Ділене є операндом подвійної довжини. Байтові команди ділять 16-бітове ділене на 8-бітовий дільник. У результаті ділення формується два числа. Частка записується в регістр *AL*, а залишок – у регістр *AH*.

Команда ділення, працююча із словами, потребує, щоб ділене знаходилось у парі регістрів *DX* і *AX*. Причому регістр *DX* містить старшу частину діленого, а регістр *AX* – молодшу. Команда ділення слів поміщає частку в регістр *AX*, а залишок – у регістр *DX*.

Ділення 64-бітового числа на 32-бітове значення виконується над регістрами *EDX* і *EAX*. При цьому регістр *EDX* містить старшу частину, а регістр *EAX* – молодшу. Після ділення регістр *EAX* зберігає частку, а регістр *EDX* – залишок.

Синтаксис команд має вигляд :

DIV Reg

DIV [*Memory*]

IDIV Reg

IDIV [*Memory*] ,

де *Reg* – 8-, 16- або 32-розрядний регістр, *Memory* – елемент пам'яті.

Жоден з прапорів процесора не визначений після команди ділення.

Під час ділення може виникнути помилка значущості. Якщо частка більша, ніж може бути поміщено в регістр результату, мікропроцесор не

може сформувати правильний результат. У цьому випадку він виконує переривання рівня 0.

Ділення цілих чисел зі знаком *IDIV* відрізняється від команди *DIV* тільки тим, що воно враховує знаки обох операндів. Знак залишку завжди той же, що і у діленого. Значення частки округляється у напрямі нуля.

2.1.7. Команди вирівнювання довжини даних

При виконанні арифметичних операцій операнди мають бути однієї довжини. Для вирівнювання довжини числових даних зі знаком використовуються команди *CBW, CWD, CDQ* та *CWDE*.

Команда *CBW* значення регістра *AL* записує у слово в регістрі *AX* з урахуванням знака (старшого розряду регістра *AL*).

Команда *CWD* перетворює слово в регістрі *AX* у подвійне слово, яке зберігається в регістрах *DX* і *AX* (у *DX* – старші, в *AX*– молодші розряди).

Команда *CWDE* перетворює слово в регістрі *AX* у подвійне слово, яке зберігається в регістрі *EAX*.

Команда *CDQ* перетворює подвійне слово в регістрі *EAX* в подвійне слово, яке зберігається в регістрах *EAX* та *EDX* (в *EDX* – старші, в *EAX*– молодші розряди).

2.2. Програмування арифметичних виразів

При розроблені програми з цілочисельними обчисленнями є ряд особливостей :

- у вираженні можуть брати участь операнди різної довжини;
- потрібен аналіз переповнювання при виконанні операцій;
- операція ділення формує тільки цілу частину і в загальному випадку не дає точного результату.

Перед початком обчислення формулу розрахунку бажано перетворити так, щоб операція ділення була останньою. В цьому випадку формується точніший кінцевий результат. Так, наприклад, якщо задані формули

$$\frac{a}{b}+\frac{c}{d} \text{ та } \frac{ad+cb}{bd} \ ,$$

то в першому випадку потрібно три операції, у другому – п’ять. Проте, в другому випадку можна отримати точніший результат.

Якщо a = 8, b = 3, c = 10, d = 3, то результат обчислення за першою формулою в цілих числах дорівнюватиме 5, а за другою – 6.

При обчисленні по виразах можна скласти схему обліку розмірів операндів. Наприклад, якщо є формула

$$\frac{a+b-c}{d}+\frac{e\times f}{h},$$

у якій беруть участь операнди, що мають розміри у байтах:

a – 1 байт (*DB*);
b – 2 байти (*DW*);
c – 4 байти (*DD*);
d – 4 байти (*DD*);
e – 2 байти (*DW*);
f – 1 байт (*DB*);
h – 2 байти (*DW*) ,

то оскільки в операціях додавання і віднімання операнди мають бути однакової довжини, в операції множення результат має довжину у два рази більше ніж початкові дані, а ділення вимагає розмір діленого у два рази більше ніж дільника, можна скласти поступову схему розмірів при обчисленні проміжних значень (рис.2.1).

Перетворення розміру операндів може бути здійснене заздалегідь до виконання обчислення або перед виконанням операції обробки даних. Для перетворення розміру даних можна використати команди *CBW, CWD, CDQ*.

Наприклад, для перетворення даних розміром 1 байт до 2 байт можна використати наступні команди:

```
MOV AL , a
CBW
```

Облік переповнювання здійснюється після виконання операцій додавання і віднімання. Якщо виникає переповнювання розрядної сітки, подальші обчислення можна припиняти, оскільки результат невірний.

Для перевірки переповнювання можна використати команду *JO*, яка здійснює перехід, якщо прапор переповнювання встановлений в 1, інакше виконується наступна команда. Команду *JO* потрібно вставляти відразу після виконання арифметичної операції.

Наприклад:

```
ADD EAX, ECX
JO MERR
SUB EAX , 5
```

...
MERR :

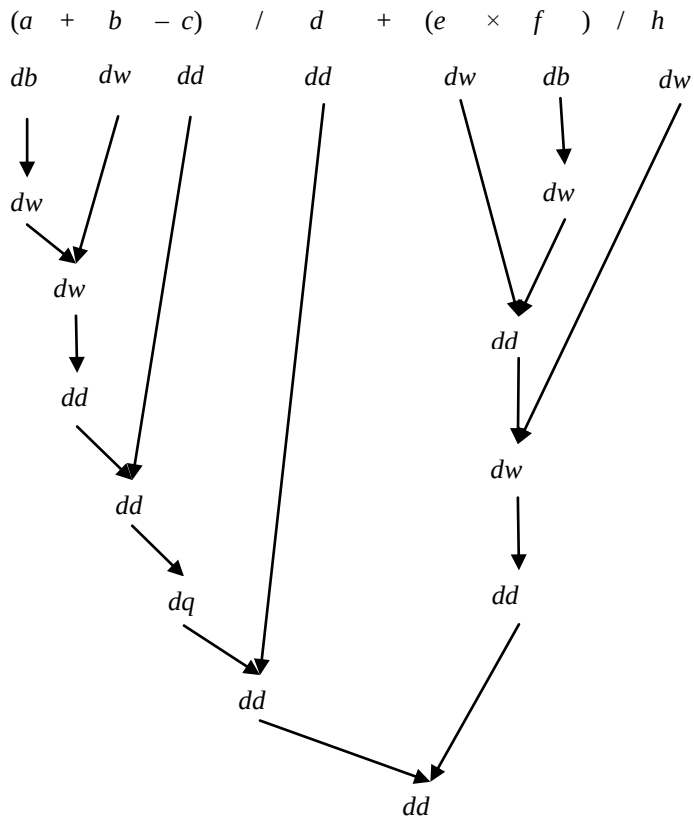

Рисунок 2.1 – Схема розмірів при обчисленні виразу

Перед операцією ділення необхідно перевірити на рівність нулю значення дільника, оскільки при діленні на 0 центральний процесор формує переривання виконання програми. Для перевірки нульового значення можна використати команду *CMP*. Наприклад:

CMP d , 0
JE M2
DIV d
...

M2:

2.3. Програма для обчислення арифметичного виразу

Розглянемо розробку програми для для обчислення арифметичного виразу

$$r = \frac{a+b-c}{d} + \frac{ef}{h},$$

у якому змінні *a* і *f* мають розмір 1 байт, змінні *b*, *e* і *h* – 2, а змінні *c*, *d* і *r* – 4 байти.

З метою підвищення точності обчислення перетворимо формулу до такого вигляду:

$$\frac{a+b-c}{d} + \frac{ef}{h} = \frac{(a+b-c)h + efd}{dh}$$

З урахуванням необхідних операцій перетворення розмірності даних програма може виглядати таким чином:

```
.DATA
        a    DB   3
        b    DW   4
        c    DD   5
        d    DD   6
        e    DW   7;
        f    DB   8
        h    DW   9
        r    DD   0
.CODE
        MOV AL , f
        CBW
        IMUL  e
        MOV   word ptr r     , AX
        MOV   word ptr r+2 , DX
        MOV   EAX , r
        IMUL   d
        MOV   EBX , EAX
        MOV   ECX , EDX
        MOV    AL , a
        CBW
```

```
        ADD   AX , b
        JO   m1
        CWDE
        SUB   EAX , c1
        JO m1
        XCHG  EAX , EDI
        MOV   AX , h
        CWDE
        XCHG   EAX , EDI
        IMUL   EDI
        ADD    EAX , EBX
        ADC   EDX , ECX
        CMP d , 0
        JE m1
        IDIV   d
        XCHG   EAX , EDI
        MOV AX , h
        CMP AX , 0
        JE m1
        CWDE
        XCHG EAX , EDI
        IDIV   EDI
        MOV r , EAX
    Invoke ExitProcess , 0 ; закінчення застосування
m1:
; переповнення
Invoke ExitProcess , 1  ; закінчення застосування з кодом помилки 1
End Start
```

Для перетворення результату множення $e \times f$ до розміру подвійного слова у програмі використана додаткова змінна *r*, в яку спочатку записуються молодші розряди добутку, а потім – старші.

Мітка *m*1 служить для переходу до закінчення обчислення у разі виявлення переповнювання при додаванні або відніманні, а також при спробі ділення на 0.

2.4. Порядок виконання роботи

1. Розробити структуру даних програми згідно з варіантом (табл. 2.2). Написати програму що реалізує обчислення арифметичного

виразу відповідно до варіанта. При виконанні обчислень здійснити контроль переповнювання цілих чисел та ділення на 0.

2. Розробити три тести для перевірки роботи програми: перший тест для перевірки працездатності, другий тест – для перевірки ділення на 0, третій – для перевірки виявлення переповнювання при виконанні додавання і віднімання.

3. Перевірити роботу програми.

4. Оформити звіт

Таблиця 2.2 –Варіанти завдань

N	Формула	Розмір змінних						
		X	*Y*	*Z*	*W*	*U*	*J*	*K*
1	((X–Y)•Z–W/U+J) / K	dw	dw	dw	db	dw	db	dw
2	((X•Z–W•U/(J+K)) / Y	db	db	dw	db	db	dw	dw
3	(X–Y•Z–W/(U+J))+K	db	dw	db	db	dw	db	dw
4	(X•Y–Z+(W–U)•J) / K	db	dw	dw	db	dw	db	db
5	((X–Y) /Z–(W•U+J))–K	dw	dw	dw	db	db	db	dw
6	((X+Y) / (Z–W/U–J))•K	db	dw	dw	db	dw	db	dw
7	(X•Y+Z•W–U / J) / K	db	dw	dw	db	db	db	db
8	(X+Y)•(Z–W / U+J)–K	db	db	dw	db	dw	db	dw
9	X / Y–Z / W+U•J–K	dw	dw	db	db	dw	db	db
10	(X–Y•Z–W / U+J) / K	db	db	dw	db	dw	db	dw

Контрольні запитання

1. Які команди операції цілочисельної арифметики включені в набір команд центрального процесора?

2. Які вимоги ставляться до операндів операцій додавання і віднімання?

3. У чому відмінність виконання команд *ADD* і *ADC*? У якому випадку обидві команди дають той самий результат?

4. Які команди необхідно використати для додавання (віднімання) двох чисел, якщо їх розмір перевищує розрядність процесора?

5. Які прапори встановлюються при виконанні операції додавання?

6. Які регістри використовуються для зберігання першого співмножника операції множення?

7. У яких регістрах формується добуток двох цілих чисел?

8. Які прапори встановлюються у процесорі після виконання операції множення?

9. У чому відмінність команд *MUL* і *IMUL*?

10. Як здійснюється ділення цілих чисел?

11. Які прапори встановлюються у процесорі після виконання ділення цілих чисел?

12. За допомогою якої команди можна отримати залишок від ділення двох чисел?

13. Для чого служить команда *CWD*?

14. Як аналізується результат виконання операції додавання і віднімання на набуття правильного значення?

15. Чи потрібен аналіз на переповнювання при виконанні операції множення?

3. ЛОГІЧНІ КОМАНДИ І КОМАНДИ ПЕРЕДАЧІ УПРАВЛІННЯ

Мета роботи : отримати навички використання логічних команд для побітової обробки даних, вивчити набір команд організації циклів і передач управління за допомогою умовних та безумовних переходів.

3.1. Логічні команди

3.1.1. Синтаксис логічних команд

У систему команд процесора X86 включені команди, які виконують основні логічні операції: *AND* (логічне і), *OR* (логічне або), *XOR* (сума по модулю 2), *NOT* (ні). Усі операції є побітовими, тобто виконуються над окремими розрядами операндів.

Синтаксис логічних команд *AND, OR* і *XOR* має вигляд:

AND приймач, джерело

OR приймач, джерело

XOR приймач, джерело

Команда інверсії розрядів має вигляд:

NOT приймач

Операнди, що використовуються як джерело і приймач, аналогічні операндам команди *MOV*. При цьому приймач виступає першим операндом логічної операції. Розміри операндів при виконанні операцій мають бути однаковими.

Остання логічна команда – *TEST* (перевірка). Ця команда виконує операцію "логічне і", але при цьому не зберігається результат операції. Команда *TEST* установлює прапори відповідно до результату.

3.1.2. Установка прапорів

Коли мікропроцесор X86 виконує логічну команду, він встановлює прапори відповідно до результату. Оскільки операція не арифметична, прапори перенесення і переповнювання завжди встановлюються рівними 0. Прапор додаткового перенесення після логічних операцій залишається невизначеним, тоді як інші прапори (знак, нуль, парність) правильно встановлюються за результатом операції. Команда *NOT* не змінює жодного прапора.

3.1.3. Використання логічних команд

Логічні команди використовуються для аналізу і обробки окремих розрядів або групи розрядів у байті або слові. Для виділення бітів вико-

ристовується команда *AND* і маска. Маска – константа, що містить одиничні значення в розрядах, які необхідно виділити. Зазвичай маска задається з використанням двійкової системи числення. Наприклад, якщо необхідно перевірити, які значення встановлені в 1, 3 і 5-му розрядах, можна використати наступні команди:

```
.DATA
Flag    DW ?
MaskaR1 DW 0000000000000010 b
MaskaR3 DW 0000000000001000 b
MaskaR5 DW 0000000000100000 b
R1      DW  0
R3      DW  0
R5      DW  0
.CODE
Mov ax , Flag
AND ax , MaskaR1
Mov R1 , ax
Mov ax , Flag
AND ax , MaskaR3
Mov R3 , ax
Mov ax , Flag
AND ax , MaskaR5
Mov R5 , ax
```

Змінні *R*1, *R*3, *R*5 дорівнюватимуть 0, якщо відповідні розряди мали значення, рівне 0. Якщо значення розрядів, що перевіряються, мали одиничне значення, то змінні приймуть значення: *R*1 = 2, *R*3 = 8, *R*5 = 32.

Для виділення групи розрядів маска міститиме одиничне значення біт, у відповідних місцях. Наприклад, для виділення перших двох і останніх двох розрядів можна використовувати маску:

```
MaskaR15_14_1_0  DW 1100000000000011 b
```

Команду *AND* можна використовувати для установлення розрядів результату в нульове значення. В цьому випадку маска повинна містити значення 0 для тих розрядів, які повинні мати нульове значення, і 1 – для тих розрядів, які не повинні змінюватися. Наприклад, для установлення значення 0 в 3 і 7-му розрядах можна використовувати такі команди:

```
.DATA
```

```
Flag     DW ?
MaskaR1 DW 1111111101110111 b
R1          DW   0
.CODE
Mov ax , Flag
AND ax ,  MaskaR1
Mov R1 , ax
```

Команду *OR* можна використовувати для установлення розрядів результату в одиничне значення. В цьому випадку маска повинна містити значення 0 для тих розрядів, які не повинні змінитися, і 1 – для тих розрядів, які мають бути встановлені в одиничне значення. Наприклад, для установлення 3 і 7-го розрядів можна використати такі команди:

```
.DATA
Flag     DW ?
MaskaR1 DW 00000000010001000 b
R1          DW   0
.CODE
Mov ax , Flag
OR  ax ,  MaskaR1
Mov R1 , ax
```

Команда *XOR* дозволяє інвертувати розряди. Маска для команди *XOR* встановлюється так, щоб на місці біта, що інвертується, була 1, а інші дорівнюватимуть 0.

Наприклад, для зміни 1 і 4-го розрядів можна використовувати такі команди:

```
.DATA
Flag     DW ?
MaskaR1 DW 000000000000010010 b
R1          DW   0
.CODE
Mov ax , Flag
XOR ax , MaskaR1
Mov R1 , ax
```

Часто команду *XOR* використовують для запису нульового значення в регістр процесора. Наприклад, дві команди:

```
XOR ax , ax
Mov ax,0
```

призводять до одного результату – в регістрі *AX* формується нульове значення. Проте, перша команда виконується швидше і займає менше пам'яті.

3.2. Команди зсуву

Команда зсуву переміщає усі біти в даних вправо або вліво.

Усі команди зсуву визначають лічильник – кількість біт, на яку потрібно виконати зсув. Його найбільш поширене значення – одиниця. Проте команда може задати довільний лічильник зрушень, заносячи його значення в регістр *CL* перед зсувом. Число в регістрі *CL* може бути будь-яким від 0 до 255. Значення 0 не викликає зсуву.

Інша загальна риса команд зсуву – установлення прапора перенесення. Команди зсуву поміщають останній витіснений з операнду біт у прапор перенесення.

3.2.1. Зсув вліво і вправо

Система команд процесора X86 має логічні і арифметичні команди зсуву даних вліво і управо. Зсув вліво є зсув двійкової кодової комбінації у бік старших розрядів, а зсув управо – у бік молодших розрядів.

У разі логічного зсуву біт, що звільнився, заповнюється нульовим значенням (рис. 3.1).

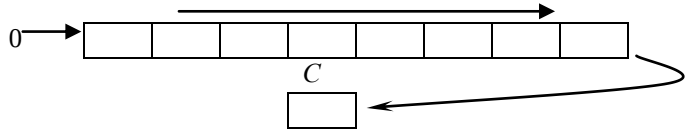

Рисунок 3.1 – Зсув вправо

При арифметичному зсуві вправо звільнений біт має значення, яке він мав до виконання операції (рис. 3.2).

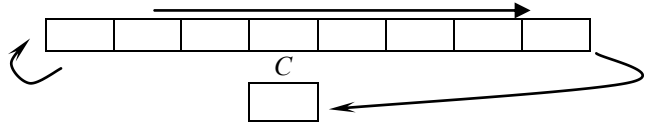

Рисунок 3.2 – Арифметичний зсув управо

Логічні зсуви подані наступними командами:

SLL приймач, 1 ;

SLL приймач, *CL* – логічний зсув вліво;

SLR приймач, 1 ;

SLR приймач, *CL* – логічний зсув управо;

SAL приймач, 1 ;

SAL приймач, *CL* – арифметичний зсув вліво;

SAR приймач, 1 ;

SAR приймач, *CL* – арифметичний зсув управо.

Арифметичній зсув управо на один розряд для цілочисельних даних рівносильний діленню на два, а зсув вліво – множенню на два.

3.2.2. Циклічний зсув вліво та управо

Циклічні зсуві записують біт, який з'являється при зсуві, в інший кінець (рис.3.3).

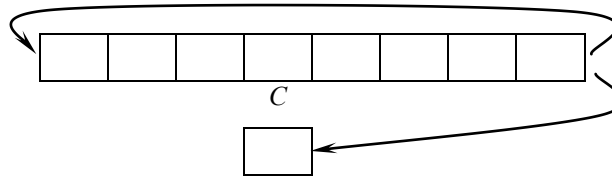

Рисунок 3.3 – Циклічний зсув управо

Команди зсуву мають синтаксис:

ROL приймач, 1 – циклічний зсув вліво;

ROL приймач, *CL* – циклічний зсув вліво на декілька розрядів;

ROR приймач, 1 – циклічний зсув управо;

ROR приймач, *CL* – циклічний зсув управо на декілька розрядів.

Циклічні зрушення вліво і вправо використовують попереднє значення прапора переносу для формування значення вільного розряду (рис.3.4).

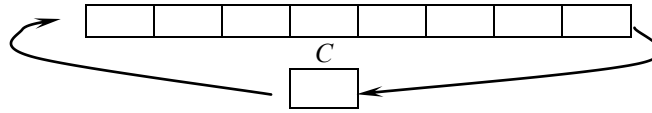

Рисунок 3.4 – Циклічний зсув управо з переносом

Команди зсуву з переносом мають синтаксис:

RCL приймач, 1 – циклічний зсув вліво на один розряд;

RCL приймач, *CL* – циклічний зсув вліво на декілька розрядів;

RCR приймач, 1 – циклічний зсув управо на один розряд;

RCR приймач, *CL* – циклічний зсув управо на декілька розрядів.

3.3. Управління прапорами центрального процесора

У системі команд є набір операцій для встановлення прапорів центрального процесора в одиничний або нульовий стан. У табл. 3.1. подані команди управління прапором перенесення і прапором напряму обробки ланцюгових команд.

Таблиця 3.1 – Управління прапорами центрального процесора

Команда	Операція
STC	Встановити прапор переносу
CLC	Очистити прапор переносу
CMC	Інвертувати прапор переносу
CLD	Очистити флаг напрямку
STD	Встановити прапор напрямку

3.4. Команди передачі управління

Система команд процесора має ряд команд безумовного і умовного управління послідовністю виконання команд програми. Ці команди називаються командами переходів або передачі управління.

Групу команд передачі управління складають:

- команди безумовних переходів;
- команди умовних переходів;
- команди виклику підпрограм;
- команди повернення з підпрограм;
- команди управління викликом;
- команди переривання.

Всі команди передачі управління не змінюють значень розрядів регістра прапорів.

3.4.1. Команди безумовних переходів.

Загальний вигляд команди безумовного переходу

JMP МІТКА ,

де МІТКА – ім’я мітки команди, на яку здійснюється перехід. Мітка у програмі задається у вигляді імені, за яким стоїть двокрапка.

Команда непрямої передачі управління

JMP регістр/ім’я змінної

здійснює перехід до команди, адреса якої знаходиться в 32-розрядному регістрі або змінній.

При цьому в регістрі або у змінній розташовується значення адреси команди, на яку передається управління. Адреса переходу в такому випадку може обчислюватися при виконанні програми, а потім завантажується в регістр або змінну. Для визначення адреси передачі управління можуть бути використані команда *LEA* або оператор *OFFSET* команди *MOV*.

3.4.2. Команди умовних переходів

Усі команди умовної передачі управління мають єдиний формат запису і перевіряють ознаки регістра прапорів, встановлені попередніми командами. Синтаксис команд умовного переходу має вигляд:

Команда умовного переходу МІТКА ,

де МІТКА – ім'я метки команди, на яку потибно здіснити перехід.

Якщо умова команди виконується, то здійснюється перехід до команди, яка має відповідну мітку.

Наприклад, у ділянці програми

```
  CMP EAX , 5
  JE M5
  MOV EBX, 0
…
  JMP M6
M5:
  MOV EBX , 1
…
M6:
```

перевіряється, чи рівно значення акумулятора п'яти. Якщо так, то в регістр *EBX* записується 0, інакше – 1.

У табл.3.2 наведені команди умовних переходів за результатами виконання попередньою командою порівняння *CMP*.

Таблиця 3.2 – Команди умовних переходів за співвідношеннями між числами

Умова	Команда	
	числа без знаку	числа зі знаком
=	*JE / JZ*	*JE / JZ*
≠	*JNZ / JNE*	*JNZ / JNE*

Продовження табл. 3.2

Умова	Команда	
	числа без знаку	числа зі знаком
>	*JA*	*JG*
≥	*JAE*	*JGE*
<	*JB*	*JL*
≤	*JNB*	*JNL*

Команди умовних переходів за значеннями окремих розрядів регістру прапорів подані в табл.3.3

Таблиця 3.3 – Команди умовних переходів за значеннями прапорів

Значення прапора	Команда	Перейти, якщо …
OF=0	*JNO*	немає переповнювання
OF=1	*JO*	є переповнювання
S=0	*JNS*	число позитивне
S=1	*JS*	число негативне
Z=0	*JNZ*	не нуль
Z=1	*JZ*	дорівнює нулю
P=O	*JNP*	паритет непарний
P=1	*JP*	паритет парний
C=0	*JNC*	немає перенесення
C=1	*JC*	є перенесення

3.4.3. Команди організації циклів

У програмах можна виділити два типи циклів – цикли на задану кількість повторень і ітеративні цикли (число повторень не задане).

Цикли на задану кількість повторень можна формувати за допомогою команд *LOOP* і *JCXZ*. Обидві команди використовують як лічильник циклу регістр *CX* процесора.

Команди організації циклів на задану кількість повторень мають синтаксис:

LOOP МІТКА

JCXZ МІТКА

Команда *LOOP* здійснює віднімання від регістру *ECX* одиниці. Потім перевіряє значення в *ECX* і, якщо воно не дорівнює 0, забезпечує перехід до команди із заданою міткою, інакше – до наступної за порядком команди. Максимальна кількість повторень циклу досягається завантаженням 0 у регістр *ECX*, оскільки віднімання 1 здійснюється до пере-

вірки значення. Для організації циклів з великим числом повторень можна організувати цикл у циклі (число повторень розкладається на множники) або виконати розбиття на два послідовні цикли (число повторень розкладається на доданки).

Додатково існує два різновиди команди – *LOOPE* (*LOOPZ*) і *LOOPNE* (*LOOPNZ*), які окрім перевірки регістру *ECX* перед повторенням циклу перевіряють прапор *Z* регістру прапорів. Для команди *LOOPZ*, якщо *Z*=0, цикл припиняється. Аналогічно команда LOOPNE забезпечує вихід, якщо *Z*=1. Таким чином, команди мають дві умови для закінчення циклу *ECX*=0 і *Z*=0 або *Z*=1, що дозволяє побудувати цикл з передчасним виходом.

Наприклад, для пошуку символу ‘A’ в рядку символів *STR*1 можна використати таку послідовність команд:

```
        MOV ECX ,128
        MOV ESI, – 1
        MOV AL,'A'
M1:
        INC ESI
        CMP [STR1+ESI] , AL
        LOOPNE M1
        CMP ECX,0
        JE M2
```

Сама команда *LOOP* значень у регістрі прапорів не змінює.

Слід мати на увазі, що команда *LOOP* забезпечує короткі передачі управління. Тому об’єм тіла циклу не повинен перевищувати 127 байт (загальний об’єм команд у циклі). Для організації циклів з великим об’ємом можна використати в тілі циклу команди безумовного переходу на продовження циклу з подальшим поверненням в основне тіло циклу.

```
        MOV ECX ,128      ; встановити параметр циклу
M1:
        …  ; тіло циклу
        Jmp M2
M3:
    Jmp M1
        M2: LOOP M3
```

Команда *JCXZ* перевіряє, але не змінює значення регістра *ECX*. Якщо значення *ECX* = 0 здійснюється перехід на задану мітку. Команда зазвичай використовується для перевірки умови перед тілом циклу

(передумова). Зміна значення регістра *ECX* повинна здійснюватися в тілі циклу додатковими командами.

Наприклад, завдання пошуку символу в рядку може бути здійснене наступною послідовністю команд :

```
        MOV ECX , LEN
        MOV ESI,–1
        MOV AL,'A'
M1:
        JCXZ M2
        INC ESI
        CMP [STR1+ESI] , AL
    JE M2
    DEC ECX
        JMP M1
  M2:
```

ESI містить зміщення до позиції символу в рядку, якщо регістр *ECX* зберігає значення, відмінне від 0.

Ітеративні цикли можна будувати, використовуючи команди умовного переходу.

3.5. Шифрування тексту з використанням логічних операцій та операцій зсуву

Розглянемо програму шифрування тексту. Кожен символ тексту шифрується окремо за таким правилом:

1. Підраховується значення *K* – кількість біт (значення зберігається в регістрі ESI), які мають значення 1 в коді символу.
2. Якщо ця кількість не рівна 0 або 8, кодова комбінація зрушується на *K* біт вліво, якщо число К парне, і управо, якщо воно не парне.
3. Якщо число біт, установлених у символі, рівне 0 або рівне 8, здійснюється інверсія усіх розрядів.
4. Після виконання зрушень або інверсії розрядів для кожного символу здійснюється логічна операція "сума за модулем 2" з кодом 01100110.

Нижче наведений текст програми для шифрування рядка символів.

```
.MODEL FLAT
.STACK 100H
.DATA
```

```
    STRING DB "QWERTYUIOP"
    MASKA DB 01100110B
    LENS DD MASKA–STRING
.CODE
START:
  MOV ECX , LENS
  MOV EBX , 0
  M1:
    MOV AL , STRING [ EBX ]
          MOV EDX , ECX
          MOV ECX , 8
          XOR ESI , ESI
         M2:
                ROR AL , 1
                JNC M3
                INC ESI
          M3:
                LOOP M2
          CMP ESI , 0
          JE M4
          CMP ESI , 8
          JNE M 5
         M4 :
                NOT AL
                JMP M7
          M5:
          MOV ECX , ESI
            TEST CL , 00000001B
                JZ M6
                ROR AL , CL
                JMP M7
                M6:
                ROL AL , CL
          M7:
            XOR AL , MASKA
                MOV STRING [ EBX ] ,AL
                INC EBX
                MOV ECX , EDX
```

*LOOP M*1

END START

3.6. Порядок виконання роботи

1. Розробити алгоритм і структуру даних відповідно до завдання.
2. Розробити програму за розробленим алгоритмом.
3. Розробити тест для перевірки роботи програми.
4. Перевірити роботу програми з використанням тесту.
5. Оформити звіт.

Варіанти завдань

1) Розробити програму пошуку кодової комбінації в рядку 1024 біт. Кодова комбінація задана у вигляді рядка завдовжки не більше 16 біт.

Наприклад: рядок для пошуку

0110010101101110...

...00101000100110010010010010110111100100100

Кодова комбінація для пошуку 010011

Результат отримати у вигляді номера розряду, з якого розпочинається кодова комбінація в рядку.

2) Розробити програму роботи з множинами, які задані у вигляді рядка біт. Кожен розряд визначає, чи є присутнім елемент у множині. Елементи у множині задані у вигляді чисел у діапазоні від 0 до 1024.

3) Розробити програму для визначення, чи ділиться задане число на ступінь двійки і, якщо ділиться, визначити максимальну ступінь. Число задане у вигляді подвійного слова (4 байти) зі знаком.

4) Розробити програму шифрування тексту шляхом підрахунку кількості 0 у коді символу і зрушення цього коду на кількість 0 вліво та накладення маски 10101010 за допомогою операції «сума за модулем 2».

5) Розробити програму побудови кодової комбінації завдовжки до 128 біт з перевернутим розташуванням біт: старший розряд на перше місце, другий – на попереднє місце і т.д.

6) Розробити програму підрахунку кількості 1 у кодах символів рядка. Додати до рядка код (байт) з кількістю 1, щоб загальне число одиниць ділилося на 7.

7) Розробити програму шифрування тексту з використанням операції «сума за модулем 2» з кодом завдовжки 5 біт. Номери розрядів, установлених в 1, задані у вигляді послідовності чисел від 0 до 4.

8) Розробити програму пошуку символу в рядку з кількістю встановлених в 1 біт рівним *N* (*N* не перевищує 8).

9) Розробити програму пошуку двох однакових послідовностей біт завдовжки не менше 4 біт у рядку біт (рядок біт заданий у вигляді рядка символів).

10) Розробити програму шифрування тексту шляхом перестановки тетрад (4 розряди) для кожного символу і накладенням маски (маска завдовжки 4 біт) операцією «сума за модулем 2».

Контрольні запитання

1. Які логічні операції включені в набір команд процесора?
2. Як формується результат виконання команди XOR, якщо операнди мають розмір два байти?
3. Як встановити необхідні розряди змінної в нульове значення?
4. Які прапори змінюються у процесорі при виконанні логічних операцій?
5. У чому відмінність арифметичного зсуву від логічного?
6. На яку максимальну кількість розрядів можна зрушувати значення змінної?
7. Яка команда забезпечує арифметичне зрушення вліво на 3 розряди?
8. Як формується прапор C при виконанні логічного зрушення на декілька розрядів?
9. У чому відмінність логічного зрушення від циклічного?
10. Чим відрізняється виконання команд ROR і RCR?
11. За допомогою якої команди можна встановити значення прапора C рівним 1?
12. На які групи розділяються команди передачі управління?
13. Як оголошується мітка у програмі на асемблері?
14. Що розуміється під терміном "непряма передача управління"?
15. Яка команда здійснює передачу управління, якщо попередня команда дала результат у формі позитивного числа?
16. За допомогою яких команд можна побудувати цикл у програмі на асемблері?
17. У чому відмінність команди LOOPE від LOOP?
18. Яку максимальну кількість циклів повторення може забезпечити команда LOOP?

4. ПРОГРАМУВАННЯ АРИФМЕТИЧНОГО СПІВПРОЦЕСОРА

Мета роботи: отримати навички роботи з дійсними числами за допомогою арифметичного співпроцесора, вивчити систему команд арифметики дійсних чисел.

4.1. Арифметичний співпроцесор

Співпроцесор – це пристрій, що розширює можливості центрального процесора. Співпроцесор дозволяє виконувати операції з числами, показаними в дійсному форматі.

4.1.1. Робота співпроцесора

Співпроцесор виконує свої числові операції паралельно з основним процесором X86. При цьому процесор X86 може виконувати наступну команду, поки співпроцесор виконує поточну команду. Оскільки два процесори можуть працювати паралельно, в деяких випадках потрібна синхронізація роботи між ними. Команда *FWAIT* зупиняє обробку команд у процесорі X86 до тих пір, поки не завершиться поточна команда співпроцесора.

4.1.2. Типи даних співпроцесора

Співпроцесор може обробляти дані розширеного набору типів, необхідних для виконання операцій над дійсними числами. При цьому співпроцесор дозволяє виконувати операції і над цілими числами. Проте, обробка цілих чисел у співпроцесорі виконується дещо довше, ніж у центральному процесорі.

У табл. 4.1 наведені допустимі типи даних співпроцесора.

Таблиця 4.1 – Типи даних співпроцесора

Тип даних	Об’ява	Діапазон подання
Коротке ціле	*DW*	$2^{-15} \div 2^{15} - 1$
Ціле	*DD*	$2^{-31} \div 2^{31} - 1$
Довге ціле	*DQ*	$2^{-63} \div 2^{63} - 1$
Дійсне число	*DD*	$\approx\pm 10^{-38} \div 10^{38}$
Дійсне число подвійної точності	*DQ*	$\approx\pm 10^{-307} \div 10^{307}$
Розширений формат	*DT*	$\approx\pm 10^{-4932} \div 10^{4932}$
Двійково-десяткове число	*DT*	18 десяткових цифр

Значення змінних у дійсному форматі записуються у формі з точкою або у формі з експонентою. Для запису числового значення завжди використовується десяткова система числення.

Наприклад, дві дійсні змінні *Y* і *W* розміром чотири і вісім байт відповідно можуть бути оголошені як:

Y DD 34.89

W DQ 98E–20

Перші дві цифри упакованого десяткового числа мають бути 80, якщо число негативне, і 00, якщо воно позитивне. Далі йдуть цифри числа і ознака шістнадцятирічного числа. Наприклад, змінну *X* з початковим значенням –1234 можна оголосити як:

X DT 80000000000000001234*H*

4.1.3. Формати даних співпроцесора x87

Існує три формати чисел з плаваючою точкою, підтримуваних співпроцесором x87. На рис.4.1 показана структура чисел дійсного числа одинарної точності.

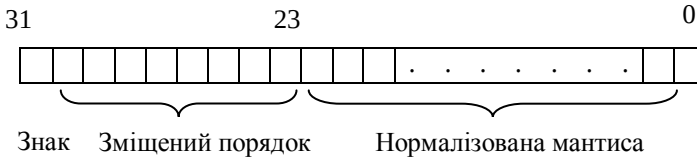

Рисунок 4.1 – Структура дійсного числа одинарної точності

Коротке дійсне число займає 32 біта. Його мантиса містить 23 біта (7–6 десяткових цифр). Значення порядку (8 розрядів) лежить у діапазоні від – 127 до 127. Біт, що залишився, визначає знак усього числа.

Довгий дійсний формат числа займає 64 біта, або вісім байтів пам’яті. Число подвоєної точності має 52-бітову мантису (15–16 десяткових цифр). Порядок займає 11 біт.

Розширений формат дійсного числа є найбільш точним і має великий діапазон. Воно подається 80 бітами. Його мантиса має довжину 64 біта. Це дає точність, еквівалентну приблизно 19 десятковим цифрам.

4.1.4. Програмна модель співпроцесора x87

На рис. 4.2 показана програмна модель співпроцесора.

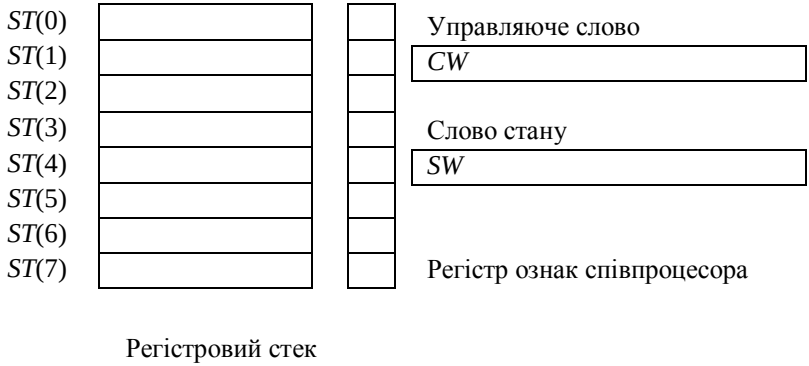

Рисунок 4.2 – Програмна модель співпроцесора

Співпроцесор x87 має регістри спеціального призначення і вісім регістрів для операндів.

Регістровий стек співпроцесора складається з восьми регістрів, кожен з яких має ширину 80 біт. Співпроцесор обробляє усі дані в тимчасовому дійсному форматі.

Регістр ознак дозволяє співпроцесору відстежувати, які з регістрів стека використовуються, а які – вільні. Будь-яка спроба запису даних у стек на вже зайняту позицію призводить до виникнення особливої ситуації – недійсної операції.

Програма може заносити дані у стек співпроцесора за допомогою команди завантаження. Якщо число у пам'яті записане не в тимчасовому дійсному форматі, співпроцесор перетворить його у 80-бітове подання під час виконання команди завантаження.

Аналогічно команди запису витягають значення з регістрового стека і поміщають їх у пам'ять. При необхідності перед збереженням даних у пам'яті здійснюється перетворення формату даних до відповідного типу автоматично.

Управляючий регістр дозволяє програмісту задати режим роботи співпроцесора:

- Вибрати один з двох різних типів нескінченності. За умовчанням установлюється проектне замикання, при якому співпроцесор трактує позитивну і негативну нескінченність як єдину нескінченність без знака. Інший спосіб – афінне замикання, в яке входить і позитивна, і негативна нескінченність.

- Установлювати один з чотирьох способів округлення:
  - до найближчого числа;
  - до меншого числа;
  - до більшого числа;
  - відкидання розрядів.

За умовчанням установлюється режим округлення до найближчого числа.

Слово стану x87 містить стан співпроцесора. Розташування окремих бітів слова стану показане на рис.4.3.

15	14	13	10	9	8	5	4	3	2	1	0
	C3	TOP	C2	C1	C0	PE	UE	OE	ZE	DE	IE

Рисунок 4.3 – Слово стану співпроцесора

Біти *PE, UE, OE, ZE, DE* і *IE* відображають помилки при виконанні поточної операції співпроцесора:

- *PE* – помилка точності;
- *UE* –антипереповнення;
- *OE* – переповнювання;
- *ZE* – ділення на 0;
- *DE* – денормалізація;
- *IE* – неприпустима операція.

Розряди *C*3–*C*0 установлюються при виконанні операції порівняння двох дійсних чисел.

Три розряди *TOP* – містять фізичний номер регістру співпроцесора, який виступає вершиною стека.

4.1.5. Команди співпроцесора

Команди співпроцесора повинні розглядатися як розширення набору команд центрального процесора.

У командах співпроцесора, якщо операнд є елементом пам’яті, можуть використовуватися ті ж режими адресації, що і в командах центрального процесора.

Основні команди співпроцесора наведені в табл. 4.2.

Таблиця 4.2 – Основні команди співпроцесора

Команда	Операція
Завантаження числових даних	
FLD	*st*(0) ← дійсне число (DD, DQ, DT)
FILD	*st*(0) ← ціле число (DW, DD, DQ)
FBLD	*st*(0) ← двійково–десяткове число (DT)
Завантаження констант	
FLDZ	*st*(0) ← 0.0
FLD1	*st*(0) ← 1.0
FLDPI	st(0) ← π
FLDL2T	st(0) ← $\log_2(10)$
FLDL2E	st(0) ← $\log_2(e)$
FLDLG2	st(0) ← $\log_{10}(2)$
FLDLN2	st(0) ← $\log_e(2)$
Запис значення регістрів співпроцесора у пам'ять обчислювача	
FST	(DD/DQ) ← st(0) збереження дійсного числа
FBST	(DT) ← st(0) збереження двійково-десяткового числа
FIST	(DW/DD/ DQ) ← st(0) збереження цілого числа
FSTP	(DD/DQ) ← st(0) збереження дійсного числа з видаленням значення із стека
FBSTP	(DT) ← st(0) збереження двійково-десяткового числа з видаленням значення із стека
FISTP	DW/DD/DQ ← st(0) збереження цілого числа з видаленням значення із стека
Порівняння даних	
FCOM	st(1) – st(0) порівняння дійсних чисел
FCOMPP	порівняння дійсних чисел st(1) та st(0) з видаленням із стека
FTST	st(0) = = 0.0 порівняння з 0
FICOM	st(0) – (DW/DD) порівняння з цілим числом
FICOMP	st(0) – (DW/DD) порівняння з цілим числом з видаленням значення st(0) із стека
Завантаження числових даних	
FLD	*st*(0) ← дійсне число (DD / DQ / DT)
FILD	*st*(0) ← ціле число (DW/DD/ DQ)
FBLD	*st*(0) ← двійково–десяткове число (DT)
Завантаження констант	
FLDZ	*st*(0) ← 0.0
FLD1	*st*(0) ← 1.0
FLDPI	st(0) ← π
FLDL2T	st(0) ← $\log_2(10)$
FLDL2E	st(0) ← $\log_2(e)$
FLDLG2	st(0) ← $\log_{10}(2)$

Продовження табл. 4.2

Команда	Операція
FLDLN2	st(0) ← $\log_e(2)$
Запис значення регістрів співпроцесора у пам'ять обчислювача	
FST	(DD / DQ) ← st(0) збереження дійсного числа
FBST	(mem80) ← st(0) збереження двійково-десяткового числа
FIST	(DW/ DD / DQ) ← st(0) збереження цілого числа
FSTP	(DD / DQ) ← st(0) збереження дійсного числа з видаленням значення із стека
FBSTP	(mem80) ← st(0) збереження двійково-десяткового числа з видаленням значення із стека
FISTP	(DW/DD/DQ)← st(0) збереження цілого числа з видаленням значення із стека
Порівняння даних	
FCOM	st(1) – st(0) порівняння дійсних чисел
FCOMPP	порівняння дійсних чисел st(1) та st(0) з видаленням із стека
FTST	st(0) == 0.0 порівняння з 0
FICOM	st(0) – (DW/ DD) порівняння з цілим числом
FICOMP	st(0) – (DW/ DD) порівняння з цілим числом з видаленням значення st(0) із стека
Арифметичні команди	
FADD	st(0) ← st(1) + st(0) додавання дійсних чисел
FSUB	st(0) ← st(1) – st(0) віднімання дійсних чисел
FSUBR	st(0) ← st(0) – st(1) віднімання у зворотному порядку дійсних чисел
FIADD	st(0) ← st(0) + (DW/ DD) додавання цілого числа
FISUB	st(0) ← st(0) – (DW/ DD) віднімання цілого числа
FMUL	st(0) ← st(0)×(st(i)/DD/DQ/DT) множення дійсних чисел
FIMUL	st(0) ← st(0) × (DW/ DD) множення на ціле число
FDIV	st(0) ← st(1) / (st(0)/ DD / DQ / DT)) ділення дійсних чисел
FIDIV	st(0) ← st(0) / (DW/ DD) ділення на ціле число
FDIVR	st(0) ← st(1) / st(0) ділення дійсних чисел у зворотному порядку

Продовження табл. 4.2

Команда	Операція
FIDIVR	st(0) ←(DW/ DD) / st(0) ділення цілого числа на st(0)
FABS	st(0) ← |st(0)| абсолютне значення
FCHS	st(0) ← – st(0) зміна знака
FSQRT	st(0) ←√st(0) квадратний корінь
FSCALE	st(0) ←st(0)×$2^{ST(1)}$ масштабування
FRNDINT	st(0) ← округлення до цілого st(0)
Команда	Операція
Тригонометричні операції	
FCOS	st(0) ← COS(st(0))
FSIN	st(0) ← SIN(st(0))
FSINCOS	st(1) ← SIN(st(0)) , st(0) ← COS(st(0))
FPTAN	st(1) ← TAN(st(0)) st(0) ← 1. частковий тангенс
FPATAN	st(0) ← ATAN(st(1)/st(0)) арктангенс
Трансцендентні операції	
FYL2X	st(0) ← st(1) × $\log_2$(st(0))
F2XM1	st(0) ← $2^{st(0)} - 1$
Службові	
FINIT	ініціалізація співпроцесора
FSTSW	збереження слова стану співпроцесора
FFREE	очищення регістру
FNOP	порожня операція
FWAIT	очікування закінчення роботи співпроцесора

4.1.6. Завантаження даних в співпроцесор

Для виконання операцій дані завантажуються в регістри співпроцесора. Завантаження даних здійснюється завжди на вершину регістрового стека. У системі команд є окремі команди для завантаження дійсних, цілих і *BCD* чисел:

FLD ім'я – завантаження дійсного числа (4, 8 або 10 байт);

FILD ім'я – завантаження цілого числа (2, 4 або 8 байт);

FBLD ім'я – завантаження двійково-десяткового числа (10 байт).

Перед записом даних інформація, що зберігається у пам'яті обчислювача, перетвориться в тимчасовий дійсний формат. При цьому контроль формату подання не здійснюється. Тому послідовність

.DATA

```
    A DD 3
    B DD 3.0
.CODE
    FLD A
    FLD B
```

приведе до того, що команда *FLD B* завантажить у регістр правильне значення дійсного числа 3.0, а команда *FLD A* збереже значення дійсного числа, двійкове представлення якого збігається з двійковим представленням цілого числа 3.

У системі команд співпроцесора є декілька команд завантаження констант. Усі команди завантаження констант не мають операндів і здійснюють завантаження на вершину стека.

FLDZ	завантаження дійсного значення 0.0
*FLD*1	завантаження дійсного значення 1.0
FLDPI	завантаження числа $\pi = 3.1415926535897932$
*FLDL*2*T*	завантаження $\log_2(10) = 3.3219280948873623$
*FLDL*2*E*	завантаження $\log_2(e) = 1.4426950408889634$
*FLDLG*2	завантаження $\log_{10}(2) = 0.30102999566398119$
*FLDLN*2	завантаження $\ln(2) = 0.69314718055994530$

Якщо для завантаження потрібна інша константа, слід оголосити змінну відповідного типу, присвоїти їй необхідне значення і завантажити це значення за допомогою команди *FLD* (або *FILD* і *FBLD*).

4.1.7. Збереження значень регістрів співпроцесора

Для збереження значення одного з регістрів співпроцесора у пам’яті обчислювача використовуються команди *FST, FBST, FIST*.

Команда *FST* дозволяє зберегти значення в дійсному форматі, команда *FIST* – у форматі цілого числа, а *FBST* – у форматі двійково-десяткового числа.

Команди *FST, FBST, FIST* мають один аргумент – ім’я змінної, в якій зберігається значення регістру *ST*(0).

```
FST ім’я
FBST ім’я
FIST ім’я
```

Перед збереженням значення співпроцесор здійснює округлення даних і перетворення до необхідного формату. Якщо значення регістру співпроцесора перевищує максимально допустиме для типу змінної, то зберігається максимально можливе значення.

Команда *FST* дозволяє записувати значення регістру *ST*(0) в інший регістр співпроцесора. Для цього в команді вказується регістр, в який здійснюється запис значення *ST*(0).

FST ST(2)

Різновид команд збереження *FSTP, FBSTP, FISTP* виконує аналогічні дії командам *FST, FBST, FIST*, за винятком того, що ці команди після запису даних видаляють інформацію з вершини стека співпроцесора. Це дозволяє не очищати регістри окремою командою.

4.1.8. Арифметичні операції

Співпроцесор підтримує операції складання, віднімання, множення і ділення дійсних чисел. Ці операції виконуються командами *FADD, FSUB, FMUL* і *FDIV* відповідно.

Команди *FADD, FSUB, FMUL* і *FDIV* можуть не мати аргументів, мати один або два аргументи.

Якщо команда не має аргументів, операція виконується над *ST*(1) і *ST*(0). При цьому *ST*(1) – лівий операнд, а *ST*(0) – правий операнд операції. Після закінчення операції обидва операнди видаляються із стека, а на вершину стека співпроцесора записується результат.

Якщо заданий один аргумент, операція виконується над вершиною стека і операндом, заданим у команді.

Якщо задані два аргументи, то це мають бути регістри співпроцесора. При цьому один з операндів є регістром *ST*(0), а другий – будь-яким регістром співпроцесора.

Команди віднімання і ділення мають різновид: *FSUBR, FDIVR*. Особливістю цих команд є те, що операції віднімання і ділення виконуються над операндами у зворотному порядку.

До арифметичних операцій можна віднести також операції зміни знака, обчислення абсолютного значення і знаходження арифметичного квадратного кореня позитивного числа. Для виконання цих операцій використовуються команди *FCHS, FABS* і *FSQRT* відповідно. Команди не мають операндів і обробляють дані, які знаходяться в регістрі *ST*(0).

4.1.9. Порівняння дійсних чисел

При виконанні команди порівняння дійсних чисел *FCOM* і *FCOMP* лівий операнд операції знаходиться в регістрі співпроцесора *ST*(0). Другий вказується в коді команди. Це може бути змінна або регістр співпроцесора. Якщо операнди в командах *FCOM* і *FCOMP* не вказані, здійснюється порівняння двох значень, які зберігаються у *ST*(0) і *ST*(1). Ре-

зультат порівняння зберігається в розрядах *C*3, *C*2 і *C*0 регістру стану співпроцесора. Безпосередній аналіз цих розрядів центральним процесором неможливий. Тому необхідно перенести значення розрядів *C*3, *C*2 і *C*0 у регістр прапорів, після чого виконати операції умовного переходу.

Наприклад, для порівняння двох дійсних чисел *X* та *Y* одинарної точності можна використати наступну послідовність команд.

```
.DATA
    X DD 3.4
    Y DD –2.6
.CODE
    FLD X
    FCOM Y
    FSTSW AX
    SAHF
    JP No_CMP
    JC X_LT_Y
    JZ X_EQ_Y
        X_GT_Y :        ; X більше Y
…
    JMP  NEXT
        X_EQ_Y:         ;X рівно Y
…
    JMP NEXT
        X_LT_Y:      ;X менше Y
…
    JMP NEXT
No_CMP:
…
NEXT:
```

Команда *FCOMP* після виконання порівняння забезпечує видалення першого операнда з *ST*(0).

Команда *FCOMPP* не має аргументів і здійснює порівняння значень *ST*(0) і *ST*(1). Після порівняння значення *ST*(0) і *ST*(1) видаляються з регістрового стека.

4.1.10. Тригонометричні операції

Система команд співпроцесора має набір команд для обчислення значень тригонометричних функцій sin(x), cos(x) та tg(x). Аргумент

тригонометричної функції задається в радіанах і повинен знаходитися в діапазоні від -2^{63} до $+2^{63}$.

Для обчислення sin(x) та cos(x) використовуються команди *FSIN* і *FCOS*. Команди не мають операндів. Значення аргументу знаходиться на вершині стека (регістр *ST*(0)). Команда *FSINCOS* дозволяє вичислити одночасно sin(x) та cos(x). При цьому значення sin(x) зберігається в регістрі *ST*(1), а cos(x) – у *ST*(0).

Для обчислення tg(x) використовується команда *FPTAN*, яка знаходить частковий тангенс – значення двох катетів прямокутного трикутника. Відношення цих катетів і задає значення тангенса кута. Для обчислення тангенса кута можна використати послідовність команд:

```
.DATA
X DD 1.2
.CODE
FLD X
FPTAN
FDIV ,
```

а для обчислення котангенса:

```
.DATA
X DD 1.2
.CODE
FLD X
FPTAN
FDIVR
```

Зворотні тригонометричні функції подані тільки функцією знаходження арктангенса. Обчислення арктангенса здійснюється командою *FPATAN*. Команда не має аргументів. Особливістю команди *FPATAN* є те, що значення тангенса кута задається у вигляді значень двох катетів *X* і *Y*. Відношення *X* / *Y* обумовлює значення тангенса. Значення *X* повинне знаходитися в регістрі *ST*(1), а *Y* – у *ST*(0). Після виконання команди *FPATAN* значення *ST*(0) і *ST*(1) видаляються із стека, а на вершину стека записується значення арктангенса.

Наприклад, для обчислення кута, тангенс якого дорівнює 0.456, можна використати таку послідовність:

```
.DATA
X DD 0.456
.CODE
FLD X  ;
```

FLD1 ;
FPATAN

4.1.11. Логарифмічні функції та функції зведення у ступінь

Операція зведення у ступінь представлена командою *F2XM*1, яка обчислює вираз $2^X - 1$. Значення *X* має бути в діапазоні – 1 ÷ +1 та знаходитися в *ST*(0). Наприклад, для обчислення $2^{0.35}$ можна використати наступну послідовність команд:

.DATA
X DD 0.35
.CODE
FLD X
*F2XM*1
*FLD*1 ; загрузка 1.0
FADD

Для зведення числа 2 до ступеня більшої ніж 1 можна скористатися формулою $2^X = 2^{[X]} \times 2^{X-[X]}$, де [*X*]– ціла частина числа *X*, а *X*–[*X*] – дробова частина. Зведення 2 в ціле значення можна здійснити за допомогою команди *FSCALE*.

Якщо потрібне обчислення довільного числа в позитивний ступінь, можна скористатися формулою

$$X^Y = 2^{Y \times log_2 X}$$

Команда *FYL2X* обчислює значення виразу $Y \times \log_2(X)$. При цьому значення *Y* повинне знаходитися в *ST*(1), а значення *X* – у *ST*(0). Значення змінної *X* має бути позитивним.

Для обчислення логарифма довільної основи необхідно скористатися основною логарифмічною тотожністю

$$\log_b(X) = \log_2(X) / \log_2(b)$$

4.1 Розробка програми для обчислення з використанням дійсної арифметики

Розглянемо розробку програми для для обчислення арифметичного виразу

$$\frac{\ln\left(\mathrm{tg}(\pi - x + x^3)\right)}{\cos(x^2 - x + A)},$$

де *A* – константа, *x* – значення змінної.

Перед розробкою програми необхідно перетворити арифметичний вираз так, щоб усі операції з цього виразу могли бути виконані за допомогою команд співпроцесора.

Обчислення натурального логарифму не підтримується співпроцесором. Тому замінимо натуральний логарифм на логарифм по основі 2. Крім того, замінимо операцію зведення у ступінь на множення. Отримаємо вираз

$$\frac{1\times\log_2\left(\mathrm{tg}(\pi - x + x\times x\times x\ )\right)}{\log_2 \mathrm{e}\times\cos(x\times x - x + A)}.$$

При обчисленні потрібна перевірка на допустиме значення. Так, у заданому виразі слід перевірити значення тангенса в чисельнику перед обчисленням двійкового логарифма. Значення тангенса не може бути негативним числом. Значення косинуса ув знаменнику не може бути рівним 0.

Нижче наведена програма, що дозволяє вичислити значення цього виразу.

```
.586
.MODEL FLAT
.STACK 100H
.DATA
X     DD    1.1
A     DQ    0.9834
ZERO DQ    0.0
.CODE
START :
    FINIT
            FLD1
            FLDPI
            FSUB X
            FLD X
            FMUL X
            FMUL X
            FADD
            FCOM ZERO
            FSTSW AX
            SAHF
            JP EXIT
            JC EXIT
```

```
        JZ EXIT
        FPTAN
        FDIV
        FYL2X
        FLD X
        FMUL X
        FSUB X
        FADD A
        FCOM ZERO
        FSTSW AX
        SAHF
        JP EXIT
        JC NEXT
        JZ EXIT
NEXT:
        FCOS
        FDIV
        FLDL2E
        FDIV
        EXIT:
        RET
END START
```

Особливістю розробки програми безпосередньо за арифметичним виразом є те, що необхідно відстежувати, в яких регістрах співпроцесора зберігаються початкові та/або проміжні дані. Зручною формою для програмування співпроцесора є подання арифметичного виразу в польському записі. Для арифметичного виразу заданого раніше польська форма матиме вигляд:

1 Pi *X* – *X X X* × × + tg log *X X* × *X* – *A* + cos / log e /

Перетворення польської форми подання арифметичного виразу у програму можна здійснити таким чином:

Вираз у польському записі обробляється зліва направо;

Якщо у виразі поточний елемент – ім'я змінної або константа, у програму вставляється команда *FLD* з ім'ям змінної;

Якщо у виразі поточний елемент – операція, вставляється команда, що виконує цю операцію без операндів.

Нижче наведена програма обчислення заданого виразу

```
.586
```

```
.MODEL FLAT
.STACK 100H
.DATA
X      DD     1.1
A      DQ     0.9834
ZERO DQ     0.0
.CODE
START :
   FINIT
         FLD1
         FLDPI
         FSUB X
         FLD X
         FLD X
         FLD X
         FMUL
         FMUL
         FADD
         FCOM ZERO
         FSTSW AX
         SAHF
         JP EXIT
         JC EXIT
         JZ EXIT
         FPTAN
         FDIV
         FYL2X
         FLD X
         FLD X
         FMUL
         FLD X
         FSUB
         FLD A
         FADD
         FCOS
         FCOM ZERO
         FSTSW AX
         SAHF
```

```
        JP EXIT
        JC NEXT
        JZ EXIT
NEXT:
        FCOS
        FDIV
        FLDL2E
        FDIV
        EXIT:
        RET
        END START
```

Використання польської форми подання арифметичного виразу може призводити до більшої за об'ємом програми, але при цьому можна не відстежувати вміст регістрів співпроцесора.

4.2. Порядок виконання роботи

1. Вивчити програмну модель співпроцесора.

2. Вивчити типи даних, оброблювані співпроцесором і правила їх запису на мові асемблер.

3. Вивчити систему команд і особливості виконання команд співпроцесора.

4. Розробити програму для обчислення за формулами відповідно до варіанта (табл. 4.3). Дані у програмі мають типи: a – дійсне, b – довге дійсне, c – ціле, d – довге ціле, e – двійково-десяткове, f – дійсне, X – дійсне, Y – довге дійсне, Z – ціле.

5. Перевірити програму і записати результат обчислення.

6. Перевірити роботу регістрового стека у процесі виконання програми.

7. Перевірити реакцію процесора і співпроцесора на виконання неприпустимої операції (ділення на 0, обчислення кореня з негативного числа і тому подібне).

8. Оформити звіт.

Контрольні запитання

1. Для чого служить співпроцесор?

2. Як виконує команди співпроцесор відносно до виконання команд центральним процесором?

3. Для чого служить команда *FWAIT*?

4. Які типи даних обробляє співпроцесор?

5. Який діапазон дійсних даних допустимий для обробки у співпроцесорі?

6. Для чого служить регістровий стек співпроцесора?

7. Що зберігає регістр ознак співпроцесора?

8. Для чого служить управляюче слово співпроцесора?

9. Яка інформація записується в слово стану співпроцесора?

10. Які команди використовуються для запису в регістри співпроцесора і збереження даних з регістрів співпроцесора?

11. Які арифметичні команди підтримуються співпроцесором?

12. Що таке команди із зворотним порядком виконання?

13. За допомогою яких команд здійснюється порівняння дійсних чисел?

14. Які тригонометричні операції допустимі у співпроцесорі?

15. Що розуміється під обчисленням часткового тангенса?

16. Для чого служить команда *FINIT*?

Таблиця 4.3 – Варіанти завдань

Варіант	Формула	Варіант	Формула
1	Z=(a–в)×(c+e)/d X=a×|(c–a)|+f ^{2} Y= a× (sin(b) – 3×tg(f))	6	Z=(a+в)/((c –e) × d) X=|a×(c–a)| –f $^{-2}$ Y= a – (cos(b) – 4×tg(f))
2	Z=(a+в)/((c –e) × d) X=|a×(c–a)| –f $^{-2}$ Y= a× (cos(b) – 4)/ctg(f))	7	Z=(a–в) × ((c + e) × d) X=a×| (c–a) –f $^{-2}$ | Y= a× (cos(b) – 4×ctg(f))
3	Z=(a×в)/((c –e) – d) X=|a×(c–a)| –f $^{-2}$ Y= a – (cos(b) / 4+tg(f))	8	Z=(a/в)/((c –e) + d) X=|a^{-2} ×(c–a)| –f Y= a+ (cos(b) – 4×tg(f))
4	Z=(a/в)+((c –e) × d) X=|a×(c–a)| –f $^{-2}$ Y= a+ (cos(b) + 4/ctg(f))	9	Z=(a+в)/((c/e) × d) X=|a×(c–a)$^{-2}$ | +f Y=a – (cos(b) – 4×ctg(f))
5	Z=(a+в)/((c –e) + d) X=|a×(c–a)| –f $^{-2}$ Y= a+ (cos(b) – 7+tg(f))	10	Z=(a+в)/((c –e) / d) X=|a×(c–a)| –f $^{-2}$ Y= a+ (cos(b) – 4×tg(f))

5. ОРГАНІЗАЦІЯ ПІДПРОГРАМ

Мета роботи: вивчення особливостей створення підпрограм, організації передачі параметрів підпрограмам і повернення результатів, виклик підпрограми і способи повернення управління з підпрограми.

5.1. Підпрограми

Підпрограма - пойменована або іншим чином ідентифікована частина комп'ютерної програми. Підпрограма може багаторазово викликатися з різних частин програми з наступним поверненням до виконання команди, що слідує за викликом. Підпрограми дозволяють будувати програми за модульним принципом. Підпрограми також дозволяють зробити програми набагато більш компактними, оскільки окрему підпрограму можна викликати в багатьох місцях (рис. 5.1).

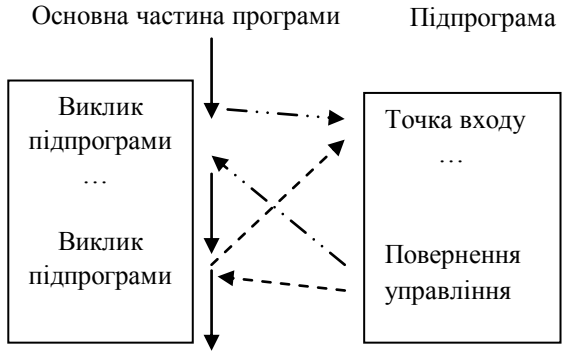

Рисунок 5.1 – Робота програми з використанням підпрограми

Коли підпрограма закінчує роботу, виконується команда *RET*, яка витягує з стека адресу, занесену туди відповідною операцією *CALL*, і заносить її в лічильник команд *IP*. Це призводить до того, що програма відновить виконання з команди, що йде за командою *CALL*.

5.1.1. Оголошення підпрограми

Підпрограми оформляються наступним чином:

```
Ім'я_підпрограми PROC
        Команди (тіло підпрограми)
```

Ім’я_підпрограми ENDP ,

де Ім’я_підпрограми – рядок символів, що задає ім’я підпрограми. Ім’я підпрограми використовується для виклику підпрограми командою *CALL*.

Директива PROC відзначає початок, а директива ENDP – кінець тексту підпрограми.

Виконання підпрограми починається з першої команди після директиви PROC.

Набір команд підпрограми повинен включати команду RET, яка забезпечує повернення управління з підпрограми. Підпрограми можна оголошувати в будь-якому порядку і в будь-якому місці в сегменті CODE.

5.1.2. Виклик підпрограми і повернення з підпрограми

Для передачі управління підпрограмі використовується команда *CALL*, яка заносить адресу наступної інструкції у стек і завантажує в регістр IP адресу відповідної підпрограми, здійснюючи перехід на підпрограму.

Команда *CALL* має синтаксис:

CALL ім’я_підпрограми

для виклику підпрограми по імені або для непрямого виклику

CALL [регістр] ,

де регістр – один з 32-бітових регістрів, який містить адресу першої команди підпрограми.

Для повернення управління в місце виклику підпрограми використовується команда *RET*.

5.1.3. Передача параметрів

Існує три загальноприйнятих способи передачі параметрів:

у регістрах процесора;

через глобальні змінні;

через область у стеку.

Передача параметрів в регістрах дуже проста. Для цього потрібно просто помістити значення у відповідні регістри і викликати підпрограму.

Передача даних через глобальні змінні використовується, якщо обсяг даних великий. Оскільки доступ до глобальних змінних можливий з

будь-якого місця програми, перед викликом підпрограми в глобальних змінних можна розмістити відповідні значення і мати доступ до них при виконанні підпрограми.

Передача параметрів до стека передбачає попереднє завантаження значень у стек програми перед викликом підпрограми. Кількість параметрів, порядок їх завантаження у стек, а також число байт для кожного параметра визначається на етапі розробки підпрограми.

Зважаючи на особливості роботи стека програми, розмір параметрів повинен бути кратний 2 (або 4). Для передачі параметрів розміром в один байт передається два байти, один з яких використовується для розміщення значення параметра.

Наприклад, для передачі трьох параметрів, перший з яких – ціле число, другий – дійсне число одинарної точності, а третій – одиночний символ, може бути використана наступна послідовність команд:

```
.DATA
Kol     DW  56
Zena    DD  5.8
Tip     DB  'M'
.CODE
Push kol
Push Zena
Mov al , Tip
Push   ax
```

Для доступу в підпрограмі до параметрів, переданих через стек, зазвичай використовують регістр *EBP*. Для цього першими командами підпрограми повинні бути команди:

```
push ebp
mov ebp,esp
```

Тоді зміщення до значень параметрів щодо поточного значення регістру *EBP* будуть: 8 байт для параметра *Kol*, 10 байт для параметра *Zena* і 14 байт для параметра *Tip* (рис. 5.2).

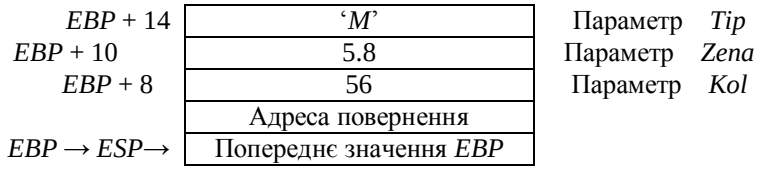

Рисунок 5.2 – Доступ до параметрів підпрограми

5.1.4. Локальні змінні підпрограми

Локальні змінні підпрограми зазвичай розміщують у стеку. Для цього підраховують загальну кількість байт для розміщення всіх локальних змінних, округлюють це число до парного значення в більший бік і використовують команду *SUB ESP*, const (const – об'єм виділеної пам'яті).

Наприклад, якщо необхідне використання трьох локальних змінних: ціле число розміром 2 байти, одиночний символ і рядок символів довжиною 23 символи, то загальний обсяг пам'яті складе 26 байт. У цьому випадку команда

SUB ESP, 26

резервує у стеку 26 байт для розміщення локальних змінних.

Доступ до локальних змінних здійснюється за допомогою зміщення щодо значення регістра *EBP*. Зміщення до відповідної змінної визначається розробником програми (рис. 5.3).

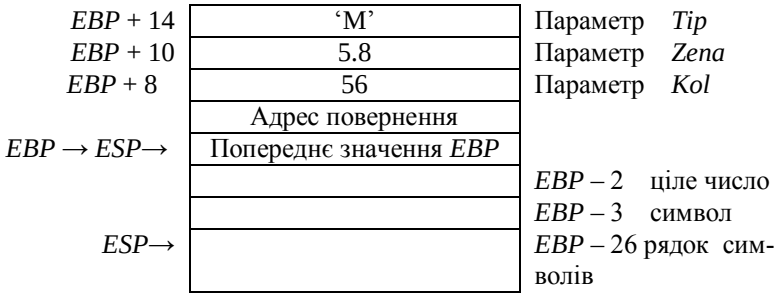

Рисунок 5.3 – Доступ до локальних змінних підпрограми

5.1.5. Повернення значень

Повернення результату роботи підпрограми може бути здійснене:

в регістрах процесора;

в глобальних змінних;

через покажчики (адреси розміщення даних) в області параметрів підпрограми.

Якщо обсяг даних, що повертаються підпрограмою невеликий, результат можна розміщувати в регістрах процесора. Зазвичай для повернення одиночного значення використовується регістр *EAX*.

Якщо обсяг даних для повернення результату перевищує можливості регістрів процесора, можна використовувати глобальні змінні або покажчики.

5.1.6. Локальні мітки підпрограми

У командах умовного переходу у підпрограмі можна використовувати локальні мітки. Мітка називається локальною, якщо її ім'я починається з двох символів @@. Наприклад:

@@Begin :

Особливістю локальних міток є те, що область видимості локальної мітки обмежена тілом підпрограми або областю команд між нелокальними мітками (рис.5.4.). Це дозволяє мати в різних підпрограмах мітки з однаковими іменами.

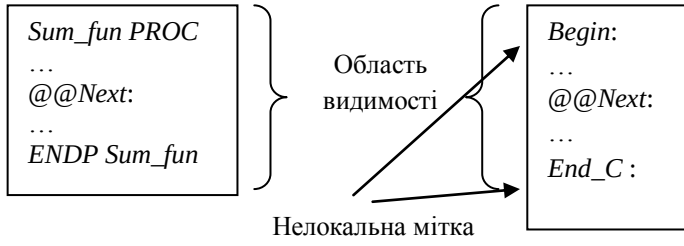

Рисунок 5.4 – Область видимості локальних міток

5.1.1. Збереження регістрів

При вході в кожну підпрограму вміст регістрів, які будуть використовуватися в підпрограмі для проміжних розрахунків, слід зберегти і відновити перед поверненням з підпрограми. Зазвичай для збереження регістрів використовується стек програми.

5.2. Приклад розробки підпрограми

Розглянемо розробку підпрограми для перетворення рядка символів у ціле число. Для перетворення рядка в число можна використовувати схему Горнера. Обробка рядка здійснюється з боку старшої цифри.

1. Встановити значення числа N = 0.
2. Отримати старший символ рядка.
3. Помножити число N на 10.
4. Додати число, відповідне поточній цифрі.
5. Якщо в рядку є ще цифри, отримати наступну цифру і перейти до п.3.
6. Зупинка.

Рядок символів з поданням числа може мати провідні символи пробілу, знак числа, цифри і кінцеві пробіли. Знак числа може бути відсутнім. У цьому випадку передбачається, що число позитивне.

Підпрограмі передаються два аргументи: рядок символів і довжина (кількість символів у рядку).

Результат перетворення рядка записується в регістр *EAX*.

Якщо при перетворенні були виявлені помилки (в рядку символів є символи, відмінні від цифр, або значення числа більше поданого в чотирьох байтах), прапор переносу *C* установити в 1. При відсутності помилок перетворення прапор перенесення *C* дорівнює 0.

Рядок символів передається в підпрограму як адреса її розміщення в пам'яті обчислювача.

На рис. 5.5 подано стан стеку програми перед викликом підпрограми.

EBP + 12	*S* (адреса)	Адреса рядка
EBP + 8	*N*	Кількість символів
EBP + 4	Адреса повернення	
EBP → *ESP*→	Попереднє значення *EBP*	

Рисунок 5.5 – Передача параметрів у підпрограму перетворення рядка

Нижче наведено текст підпрограми *A_To_I*.

```
A_To_I   proc
    Push EBP
```

```
    Mov EBP , ESP
    Push EBX
    Push ECX
    Push EDX
    Push ESI
    Push EDI
    Mov ESI ,  [EBP + 12]
    Mov ECX ,[EBP + 8]
@@_Del_blank:    ; пропуск провідних пробілів
    Mov AL,[ESI]
    Inc ESI
    Cmp AL,' '
  Loope @@_Del_blank
    Dec ESI
    Inc ECX
    Mov EDI , 0
    Cmp AL , '-'
    Jne @@_Plus
    Mov EDI , 1
    Inc ESI
    Dec ECX
  Jmp @@_Next
@@_Plus:
    Cmp AL,'+'
  Jne  @@_Next
    Inc ESI
    Dec ECX
@@_Next:
    Xor EAX , EAX
    Xor EBX , EBX
@@_Digit:
     Mov BL,[ESI]
     Cmp BL , '0'    ; аналіз символу на цифру
     Jb @@End_digit
     Cmp BL , '9'
     Ja @@End_digit
     Push EDI
     Mov EDI , 10
```

```
        Imul EAX,EDI
        Pop EDI
        JO @@Err1
        SUB BL,'0'
        Add EAX, EBX
        JO @@Err1
        Inc ESI
        Loop @@_Digit
 @@End_digit:
       CMP ECX,0
        JE @@_SIGN
        Cmp BL , ' '
        JNE @@Err1
@@_SIGN:
        Cmp EDI , 0
        Je @@m1
        Neg EAX
@@m1:
     Clc
@@m2:
     Pop EDI
Pop ESI
Pop EDX
Pop ECX
Pop EBX
     Pop EBP
     Ret
@@Err1:
     Stc
     Jmp @@m2
A_To_I endp
```

У підпрограмі не реалізована перевірка на порожній рядок і на рядок, який містить тільки пробіли.

Першими командами зберігається е стеку значення регістре EBP і встановлюється його нове значення, так щоб забезпечити доступ до параметрів підпрограми. За ними йдуть команди збереження регістрів процесора, які будуть використовуватися е підпрограмі для проміжних розрахунків.

Регістр ECX використовується для зберігання числа символів у рядку, а регістр ESI – для доступу до одиночних символів при аналізі рядка. У регістрі EDI зберігається знак числа (0 для позитивних чисел і 1 – для негативних). Якщо знак числа відсутній, передбачається, що число позитивне. Регістр BL використовується для обробки чергової цифри рядка. При нормальному завершенні перетворення прапор перенесення C установлюється в 0 за допомогою команди CLC.

5.3. Порядок виконання роботи

1. Розробити структуру даних програми відповідно до завдання.

2. Розробити порядок і вибрати спосіб передачі параметрів у підпрограму.

3. Вибрати спосіб повернення результату з підпрограми.

4. Розробити алгоритм розв'язання задачі.

5. Розробити програму для розв'язання завдання з використанням підпрограми.

6. Провести тестування програми.

7. Оформити звіт.

Варіанти завдань.

1. Обчислення N-го числа Фібоначчі.

2. Обчислення найбільшого загального дільника для двох цілих чисел.

3. Обчислення найменшого спільного кратного для двох цілих чисел.

4. Обчислення середнього геометричного для масиву чисел.

5. Обчислення кореня квадратного цілого числа ітеративним методом.

6. Перевірка цілого числа на просте значення.

7. Розкладання цілого числа на прості множники.

8. Перетворення рядка символів з національними літерами в рядок з використанням англійських букв.

9. Перетворення числа в рядок символів.

10. Перевірка на щасливе число.

Контрольні запитання

1. Що розуміється під терміном «підпрограма»?

2. У чому переваги і недоліки розробки програм з використанням підпрограм?

3. Як оголошується підпрограма?

4. В якій частині програми повинна розміщуватися тіло підпрограми?

5. За допомогою якої команди здійснюється виклик підпрограми?

6. Яка інформація повинна бути збережена в програмі для забезпечення повернення управління з підпрограми?

7. Що розуміється під непрямим викликом підпрограми?

8. Для чого використовується команда RET?

9. Які існують підходи для організації передачі даних у підпрограму?

СПИСОК ЛІТЕРАТУРИ

1. Юров В.И. Assembler : Специальный справочник. 2-е изд./В.И. Юров. – СПб : Питер, 2004. – 412 с.:ил.

2. Злобин В.К. Программирование арифметических операций в микропроцессорах./ В.К. Злобин, В.Л. Григорьев. - М.: Высшая школа, 1991.

3. Злобін Г.Г. Архітектура та апаратне забезпечення ПЕОМ : навч.посіб./ Г.Г. Злобін. – К.: Каравела, 2006.– 304 с.

4. Баженов П.С. Інформатика. Комп'ютерна техніка. Комп'ютерні технології : підручник./П.С. Баженов. – К.: Каравела, 2005.

ЗМІСТ

Вступ……3

1. Команди передачі даних і адресація до пам'яті……4

2. Обчислення виразів цілочисельної арифметики……13

3. Логічні команди і команди передачі управління……... 24

4. Програмування арифметичного співпроцесора……. 37

5. Організація підпрограм…….54

Навчальне видання

Методичні вказівки
до лабораторних робіт
" Використання мови асемблера для розробки застосувань "
з дисципліні "Архітектура обчислювальних систем"
для студентів спеціальностей
113 – Прикладна математика, 122 – комп'ютерні науки та інформаційні технології, 124 – системний аналіз 186 – видавництво та поліграфія

Укладачі:

КОЖИН Юрій Миколайович
МАЛИХ Олег Миколайович
ПРОКОПЕНКОВ Володимир Пилипович

Відповідальний за випуск *О.С. Куценко*
Роботу до друку рекомендував *М.І. Безменов*
Редактор *О.І. Шпільова*

План 2017 , поз. 136
Підп. до друку. Формат 60х84 1/16. Папір офсетний.
Riso–друк. Гарнітура Таймс. Ум. друк. арк. 3,8.
Наклад 25 прим. Зам. № Ціна договірна.

Видавець
Видавничий центр НТУ "ХПІ" м. Харків, 61002, вул. Кирпичова, 2
Свідоцтво про державну реєстрацію ДК №4064 від 21.08.2017 р