Лабораторна робота № 8

Тема: Модульне програмування. Підпрограми.

Мета: Навчитися використовувати підпрограми у програмах на асемблері.

Короткі теоретичні відомості

При програмуванні достатньо серйозної задачі, у програмі з'являються ділянки коду, які повторюються. При цьому втрачається наочність і компактність програми. Асемблер має засоби для вирішення проблеми дублювання ділянок коду, зокрема підпрограми (процедури) та макрозасоби. Добре написані і відлагоджені ділянки програм можна використовувати повторно.

Сучасні програми розробляються за структурно-модульним принципом: є одна головна програма і кілька підпрограм, які викликаються із головної програми за необхідністю. Основною одиницею структурованої програми є модуль. Терміни модуль, підпрограма, процедура, функція для асемблера є синонімами.

Процедура – це основна функціональна одиниця декомпозиції задачі. Процедура являє собою групу команд для вирішення конкретної підзадачі і має засоби для одержання керування із точки виклику задачі більш високого рівня і повернення керування в цю точку. Інакше кажучи, процедуру можна визначити як певним чином оформлену послідовність команд, яка при необхідності може бути викликана у будь-якому місці програми. При цьому має місце питання збереження контексту (стану) програми в точці виклику.

1. Процедури в асемблері

Для оформлення послідовності команд асемблера у вигляді процедури використовуються дві директиви: PROC і ENDP. Синтаксис опису процедури такий:

```
NAME PROC [мова] [тип] [USES регістри]      ; Заголовок
     ...                                    ; процедури
     ;команди та директиви                  ; Тіло
     ;асемблера                             ; процедури
     ...
[name] ENDP                                 ; Кінець процедури
```

В заголовку процедури обов'язковим є лише задання імені процедури та директиви PROC. Інші параметри опціональні. Параметр `[мова]` важлива, якщо передбачається взаємодія асемблерної процедури з програмами, написаними на мовах високого рівня. Цей параметр визначає порядок передачі параметрів у процедуру і спосіб очищення стеку від параметрів після повернення із процедури. Можливі значення параметра FORTRAN, C, SYSCALL, STDCALL, PASCAL та деякі інші. Заслуговує на увагу параметр `[тип]`. Цей

атрибут може приймати значення NEAR або FAR і відповідно усі процедури можуть бути двох типів: близькі і далекі. Між ними є суттєва різниця: близька процедура може бути викликана тільки із свого сегменту (в стеку зберігається лише значення лічильника команд IP), а далека процедура може викликатись із будь-якого сегменту (в стеку зберігається вміст регістрової пари CS:IP). За умовчанням атрибут ТИП приймає значення NEAR. Директива USES описує перелік регістрів, які будуть змінюватися процедурою. При цьому компілятор автоматично генерує команди збереження в стек заданих регістрів на початку процедури і команди відновлення збережених даних у регістри в кінці процедури.

Процедура може розміщуватися в будь-якому місці програми, але так, щоб на неї керування не потрапляло випадково. Є такі варіанти розміщення процедури у програмі:

- на початку програми (до точки входу в програму);
- в кінці програми (після команди, яка повертає керування операційній системі);
- всередині іншої процедури або основної програми (в цьому випадку слід передбачити обхід процедури за допомогою команди безумовного переходу JMP);
- в іншому модулі (текстовому, об'єктному або бібліотеці).

В системі команд процесора є дві команди для організації роботи з процедурами CALL і RET.

- команда CALL здійснює виклик процедури (підпрограми). Синтаксис команди:
  `CALL [тип] NAME`
  Подібно до команди JMP команда CALL передає керування за адресою із символічним іменем NAME, але при цьому в стеку зберігається адреса повернення (тобто адреса команди, яка є наступною після команди CALL). Параметр `[тип]` визначає тип переходу: NEAR або FAR.
- Команда RET зчитує адресу повернення із стеку і завантажує її в регістр IP (або CS:IP), що повертає керування команді, яка є наступною після команди CALL.

Синтаксис команди:

```
ret [число]
```

Необов'язковий параметр `[число]` позначає кількість байтів з параметрами, які видаляються із стеку при поверненні із процедури.

Приклад оформлення частини коду у вигляді підпрограми.

Лістинг 5.1.

```
.data
mes1 db "Hello World!", '$'           ; перше повідомлення
mes2 db "Good bye!", '$'              ; друге повідомлення
```

```
.code

print proc                          ; процедура виведення рядка
      mov ah,9
      int 21h
      ret
print endp

.begin
      lea dx, mes1          ; адреса першого повідомлення
      call print
      lea dx, mes2          ; адреса другого повідомлення
      call print
.end
```

2. Способи передачі параметрів

Підпрограми можуть виконувати певні дії з різними початковими даними (параметрами). Для цього в процедуру передаються параметри. Є кілька способів передачі параметрів:

- через регістри;
- через глобальні змінні;
- через стек;
- в блоці параметрів через вказівник;
- у потоці коду.

Найбільш поширеними способами є передача параметрів через регістри і через стек.

Передача параметрів через регістри. Для передачі до шести параметрів використовують регістри загального призначення в такому порядку: AX, DX, SI, DI, BX, CX. Слід уникати використання регістра BP для передачі параметрів. Регістр SP використовувати заборонено! Це вказівник стеку.

У вищенаведеному прикладі через регістр DX передається адреса повідомлення, яке виводиться на екран за допомогою програмного переривання.

Передача параметрів через стек. Параметри, які потрібно передати в процедуру, заносяться в стек командами PUSH згідно обраної домовленості про виклики. Так, наприклад, домовленість PASCAL передбачає передачу параметрів у прямому порядку, а домовленість C – у зворотному порядку (рис. 5.1, а). Стандартний пролог процедури зберігає значення регістра BP в стеку (PUSH BP), а положення вершини стеку фіксується в регістрі BP командою MOV BP, SP (рис. 5.1, б) Стан стеку після виконання прологу процедури відображає рис. 5.1, в. На час виконання процедури регістр BP виконує роль тимчасового вказівника на блок параметрів у стеку (стековий фрейм). Доступ до параметрів процедури здійснюється через вміст регістра BP: [BP+4], [BP+6], [BP+8] і т.д.

(рис. 5.1, г). У 32-розрядних програмах в стек заносяться 4-байтні числа, тому параметри знаходяться за адресами [BP+8], [BP+12], [BP+16]… відповідно.

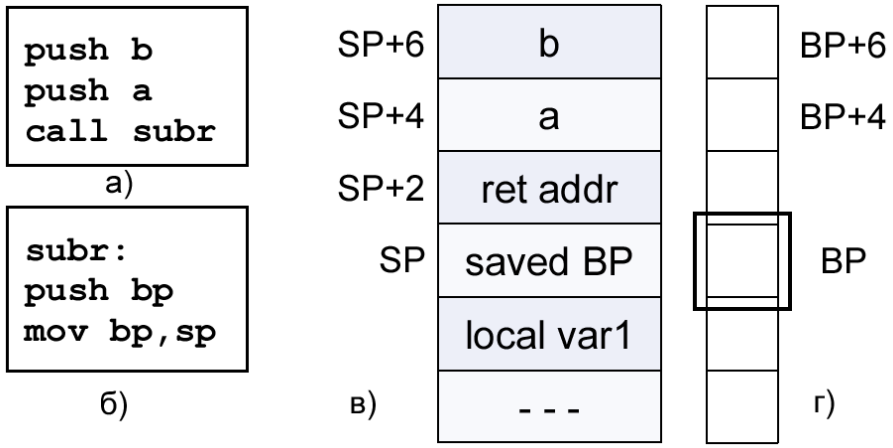

Рис. 5.1. Передача параметрів через стек: а) виклик процедури; б) стандартний пролог процедури; в) стан стеку після виконання прологу; г) регістр BP як вказівник на параметри.

Після завершення процедури значення потрібно відновити регістра BP (POP BP) і звільнити стек від переданих процедурі параметрів. Домовленість Pascal передбачає, що параметри видаляються у самій процедурі командою RET N, де N – кількість байтів, зайнятих параметрами, які потрібно видалити зі стеку. Домовленість C передбачає, що стек звільняється від параметрів викликаючою програмою відразу після повернення із процедури, тобто наступною після CALL командою. Найчастіше це команда ADD SP, 2*N, де N – кількість параметрів[1]. Наприклад:

```
.code

print proc                          ; процедура виведення рядка
      push bp                       ; зберігаємо BP
      mov bp, sp                    ; фіксуємо початок стекового кадру
      mov dx, [bp+4]                ; перший параметр
      mov ah,9
      int 21h
      pop bp                        ; відновлюємо BP
      ret
print endp

.begin
      lea ax, mes1                  ; адреса першого повідомлення
      push ax                       ; параметр в стек (адреса mes1)
      call print                    ; print (mes1)
      add sp, 2                     ; вийняти 1 параметр (pop ax)
      lea ax, mes2                  ; адреса другого повідомлення
      push ax                       ; параметр в стек (адреса mes2)
      call print                    ; print (mes2)
      add sp, 2                     ; вийняти 1 параметр
.end
```

[1] У 32-розрядному програмуванні ADD SP, 4*N.

3. Зовнішні процедури

Процедури називаються зовнішніми, якщо їх об'єктні модулі знаходяться в окремих файлах (бібліотеках). При цьому перевірені і відлагоджені процедури не компілюються повторно, а тільки підключаються на етапі компонування. Крім цього набір процедур, які найчастіше використовуються, можна оформити у вигляді об'єктного файлу (бібліотеки). Перевагою такого підходу є те, що на етапі компонування підключаються лише ті процедури, які викликаються головною програмою.

Для того, щоб компонувальник міг знайти і підключити необхідні процедури, їх потрібно певним чином описати. Файл із процедурами (назвемо його бібліотекою) повинен оголосити імена своїх процедур «видимими» для інших модулів. Видимість забезпечується директивою PUBLIC.

```
public name1, name2 ...
```

У файлі головної програми ці імена потрібно оголосити, як зовнішні, інакше на етапі компіляції одержимо помилку, пов'язану із відсутністю необхідних модулів у тексті програми. Зовнішні імена оголошуються директивою EXTRN[2].

```
extrn name1:type [, name2:type] ...
```

Якщо NAME – ім'я процедури, то її тип може бути NEAR або FAR. Якщо NAME – ім'я змінної, то її типом може бути BYTE, WORD, DWORD і т.д.

Першим компілюється файл з бібліотечними процедурами для отримання об'єктного модуля. Наприклад:

```
tasm mylib.asm
```

Після цього компілюють головний файл програми і одержують його об'єктний модуль. Нарешті, компонують обидва об'єктних модулі у виконуваний файл. Наприклад:

`tlink main.obj+mylib.obj` або просто `tlink main+mylib`

Розглянемо приклад, у якому показано сегмент коду головної програми і бібліотеку.

main.asm

```
extrn pr_sym:near
extrn kb_in:near
.code
.begin
      mov dl, 'a'
      call pr_sym
      mov dl, 'b'
      call pr_sym
      call kb_in
.end
```

mylib.asm

```
public pr_sym, kb_in
.code
pr_sym proc
      mov ah, 2
      int 21h
      ret
pr_sym endp
kb_in proc
      mov ah, 1
      int 21h
      ret
kb_in endp
end
```

[2] Зверніть увагу на написання директиви EXTRN для TASM (на відміну від EXTERN для MASM).

Порядок виконання роботи

1. Написати програму з використанням підпрограми (див. варіанти завдань). Передачу параметрів здійснити через регістри процесора.

2. Змінити програму так, щоб передача параметрів здійснювалась через стек.

3*. Винести підпрограму в окремий файл. Скомпілювати окремо головну програму і підпрограму, а потім скомпонувати в єдиний завантажувальний модуль.

Варіанти завдань

1. Програма перетворює у заданому рядку всі літери на великі. Підпрограма приймає код літери і повертає код великої літери.

2. Програма перетворює у заданому рядку всі літери на малі. Підпрограма приймає код літери і повертає код малої літери.

3. Програма рахує кількість слів у заданому рядку. Підпрограма приймає номер символа (позицію) у рядку, починаючи з цієї позиції шукає кінець слова і повертає позицію розділювача після кінця слова.

4. Програма рахує кількість пропусків між словами у заданому рядку. Підпрограма приймає позицію в рядку, починаючи з якої шукає найближчий пропуск між словами і повертає позицію знайденого пропуску.

5. Програма видаляє із заданого рядка перше слово. Підпрограма шукає в рядку початок другого слова і повертає позицію першої літери другого слова.

6. Програма видаляє із заданого рядка останнє слово. Підпрограма шукає в рядку початок останнього слова і повертає позицію його першої літери.

7. Програма записує кожне слово в заданому рядку з великої літери. Підпрограма приймає поточну позицію у рядку, знаходить початок наступного слова і повертає позицію першої літери знайденого слова.

8. Програма виводить на екран позицію заданої літери у рядку. Підпрограма приймає код літери і початкову позицію, шукає найближчу літеру в рядку і повертає її позицію.

9. Програма вводить з клавіатури одну літеру, підраховує кількість таких літер в рядку і виводить її на екран. При введенні цифри 0 програма завершує роботу. Підпрограма приймає код літери і підраховує кількість літер із заданим кодом в рядку.

10. Програма приймає з клавіатури довжину слова (одна цифра, 1…9) і підраховує у рядку кількість слів із заданою довжиною. Підпрограма приймає позицію початку слова, визначає і повертає довжину слова.

11. Програма підраховує у заданому рядку кількість слів, написаних з великої літери. Підпрограма приймає поточну позицію, шукає в рядку початок наступного слова, написаного з великої літери, повертає позицію великої літери.

12. Програма знаходить довжину гіпотенузи прямокутного трикутника за заданими катетами. Підпрограма приймає ціле число, обчислює і повертає його корінь (ціле число).

Контрольні питання

1. Як описується у програмі на асемблері процедура?

2. Як викликається підпрограма на асемблері?

3. Як здійснюється повернення з підпрограми і куди?

4. Які ви знаєте способи розташування підпрограми у програмі?

5. Яким чином можна передати у процедуру параметри?

6. Які регістри процесора можна використовувати для передачі параметрів, а які не можна? Чому?

7. Яким чином здійснюється передача параметрів через стек? Що таке стеків фрейм?

8. Що таке домовленості про виклики? Які вони бувають?

9. Як узгодити виклики підпрограм з інших програмних модулів?

10. Як об’єднати кілька об’єктних модулів програми в один завантажувальний модуль?