**Лабораторная работа № 8**

**Тема:** использование подпрограмм и макрокоманд.

**Цель:** практическое знакомство с методикой построения и использования процедур и макрокоманд на языке Ассемблер в MASM32.

**Оборудование:** персональный компьютер.

**Программное обеспечение:** Windows 7, MASM32.

1. **КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

1. **Директива PROC. Определение процедуры**.

Процедуру можно определить как именованный блок команд, оканчивающийся оператором возврата. Для объявления процедуры используются директивы PROC и ENDP. При объявлении процедуре должно быть назначено имя, которое является одним из разрешенных идентификаторов. В каждой из рассмотренных ранее программ была только одна главная процедура под названием main, например:

main PROC

При создании любых других процедур, не являющихся стартовыми, нужно в их конце разместить команду RET. В результате процессор вернет управления команде, следующей за той, которая вызвала эту процедуру:

sample PROC

ret

sample ENDP

К особому типу процедур относится так называемая стартовая процедура программы, которой назначено имя main, поскольку она должна завершаться не командой RET, a вызовом функции ExitProcess системы Windows, которая и завершает выполнение программы:

INVOKE ExitProcess, 0

В отличие от команды процессора CALL, директива INVOKE является встроенным оператором ассемблера, позволяющим вызывать указанную процедуру и передавать ей параметры.

***Пример процедуры: суммирование трех целых чисел.*** Создадим процедуру SumOf, вычисляющую сумму трех 32-разрядных чисел. Предположим, что перед вызовом процедуры значения этих чисел мы должны поместить в регистры ЕАХ, ЕВХ и ЕСХ, а сумма возвращается в регистре ЕАХ:

SumOf PROC

add eax,

ebx add

eax,ecx

ret

SumOf ENDP

Ниже приведены несколько рекомендаций по оформлению текстов процедур. В начале каждой процедуры следует помесить:

* Описание всех функций, выполняемых процедурой.
* Список входных параметров и описание их значений. Если какой-либо из параметров имеет особый тип, его нужно также указать. Обычно входные параметры указываются после ключевого слова Передается (Receives).
* Список возвращаемых процедурой значений, указанных после ключевого слова Возвращается (Returns).
* Перечень особых требований (если таковые имеются), которые должны быть удовлетворены перед вызовом процедуры (входные условия).

***Пример оформления текста процедуры.***

SumOf PROC

Вычисляет и возвращает сумму трех 32-разрядных целых чисел.

Передается: три числа в регистрах ЕАХ, ЕВХ, ЕСХ.

Возвращается: сумма в регистре ЕАХ, а также флаги состояния (переноса, переполнения и др.)

add eax, ebx

add eax, ecx

ret

SumOf ENDP

1. **Команды CALL и RET.**

Команда CALL предназначена для передачи управления процедуре, адрес которой указывается в качестве параметра. При этом процессор начинает выполнять команду, расположенную по указанному адресу. Чтобы вернуть управление команде, расположенной сразу за CALL, в процедуре используется команда RET.

***Пример вызова и возврата из процедуры.*** Предположим, что в процедуре main по смещению 00000020h расположена команда CALL. В 32-разрядном режиме длина этой команды составляет 5 байтов. Поэтому следующая команда (в нашем случае MOV) будет расположена со смешением 00000025h:

main PROC

00000020 call MySub

00000025 mov eax, ebx

Далее, предположим, что первая команда процедуры MySub расположена со смещением 00000040h:

MySub PROC

00000040 mov eax, edx

Ret

MySub ENDP

При выполнении команды CALL в стек помещается адрес следующей за ней команды (в данном случае 00000025h), после чего в регистр EIP загружается адрес процедуры MySub,После этого процессор начинает выполнять последовательность команд процедуры MySub, пока в ней не встретится команда RET. При выполнении команды RET содержимое стека, на которое указывает регистр ESP, копируется в регистр EIP. В результате процессор после команды RET будет выполнять не следующую за ней команду, а команду, находящуюся по адресу 00000025h. А это как раз команда, расположенная следом за командой CALL, которая вызвала данную процедуру.

***Пример процедуры суммирование элементов массива целых чисел.*** Cоздадим процедуру под именем ArraySum, которой из вызывающей программы будут передаваться два параметра: указатель на массив 32-рачрядпых целых чисел и количество элементов в этом массиве. Сумму элементов массива процедура будет возвращать в регистре ЕАХ:

ArraySum PROC

Вычисляет сумму элементов массива 32-разрядным целых чисел Передается: ESI = адрес массива

; ЕСХ = количество элементов массива

; Возвращается: ЕАХ = сумма элементов массива push esi

; Сохраним значения регистров ESI и ЕСХ push ecx mov eax,0

; Обнулим значение суммы

L1 :

add eax, [esi] ; Прибавим очередной элемент массива

add esi, 4 ; Вычислим адрес следующего элемента массива

loop L1 ; Повторим цикл для всех элементов массива

pop ecx ; Восстановим значения регистров ESI и ЕСХ

pop esi

ret Вернем сумму в регистре ЕАХ

ArraySum ENDP

1. **Директива INVOKE.**

Для удобства программиста в компиляторе MASM предусмотрена специальная директива INVOKE, которая позволяет с помощью одного оператора поместить в стек аргументы и вызвать процедуру. Эта директива по сути заменяет команду CALL процессоров Intel. Она позволяет передать в процедуру несколько аргументов. Синтаксис директивы INVOKE приведен ниже:

INVOKE Имя\_процедуры [, Список\_аргументов]

Аргументы, передаваемые процедуре в директиве INVOKE, перечисляются через запятую и могут отсутствовать вовсе. Нетрудно заметить основную разницу между директивой INVOKE и командой CALL: у последней нет списка аргументов.

***Пример использования директивы.*** В приведенном ниже фрагменте кода с помощью директивы INVOKE вызывается процедура AddTwo, которой передаются два 32-разрядных целых числа: .

data

Val1 DWORD 12345h

val2 DWORD 23456h

.code INVOKE AddTwo, vall, val2

1. **Макрокоманды. Отличие подпрограмм и макрокоманд**

*Макропроцедурой (macro procedure)* называется именованный блок команд языка ассемблера. После того как макропроцедура определена в программе, ее можно многократно вызывать в разных участках кода. При вызове макропроцедуры, в код программы будут помещены содержащиеся в ней команды. Не следует путать вызов макропроцедуры с *вызовом* обычной *процедуры*, поскольку в первом случае команда CALL не используется.

Следует отметить, что термин *макропроцедура* используется в документации к компилятору Microsoft Assembler для обозначения макрокоманд, не возвращающих значения. Кроме макропроцедур, существуют также *макрофункции (macro functions)*, возвращающие значение. В среде программистов прижился термин *макрокоманда,* который по сути эквивалентен *термину макропроцедура*. Поэтому далее будет использоваться термин *макрокоманда*.

**Макроопределение**

***Размещение***. Определения макрокоманд, или макроопределения, помещаются либо непосредственно в текст исходной программы на ассемблере (как правило, в его начало), либо в отдельный текстовый файл, который включается в исходную программу на этапе компиляции с помощью директивы INCLUDE.

Как только в тексте программы встречается имя макрокоманды, оно заменяется препроцессором на соответствующий набор команд, который указан в ее макроопределении. В приведенном ниже примере макрокоманда NewLine генерирует одну ассемблерную команду, которая вызывает библиотечную процедуру CrLf:

NewLine MACRO

call CrLf

ENDM

Текст этого определения обычно помещается непосредственно перед сегментом данных. После этого в сегменте кода макрокоманда вызывается так:

.code

NewLine

При обработке программы препроцессором вызов макрокоманды NewLine будет заменен приведенной ниже командой:

call CrLf

В данном случае произошла обычная текстовая подстановка, которую можно было бы выполнить и с помощью директивы TEXTEQU. У макрокоманды, так же как и у процедуры, может быть один или несколько параметров, что делает ее гораздо более мощным средством по сравнению с директивой TEXTEQU.

**Определение макрокоманды**

Макрокоманду можно определить в любом месте исходного кода программы, воспользовавшись директивами MACRO и ENDM. Синтаксис макроопределения следующий:

*Имя* MACRO *Параметр-1, Параметр-2... Список-команд*

ENDM

**Рекомендуется** всегда выделять отступами те команды, которые помещаются между директивами MACRO и ENDM, чтобы подчеркнуть их принадлежность к макроопределению. То же самое касается и выбора имен макрокоманд. Для их выделения рекомендуется пользоваться специальным префиксом. Далее для выделения имен макрокоманд перед ними помещается префикс в виде строчной буквы "m", например, mPutchar, mWriteString и mGotoXY.

Команды, находящиеся между директивами MACRO и ENDM, до вызова макрокоманды не компилируются. Макроопределение может содержать произвольное количество параметров, которые разделяются запятыми.

*Пример макроопределения mPutchar.* Рассмотрим макрокоманду mPutchar, имеющую один входной параметр, имя которого char. Данная макрокоманда выводит символ, переданный ей в качестве параметра, на терминал с помощью вызова процедуры WriteChar:

mPutchar MACRO char

push eax

mov al,char

call WriteChar

pop eax

ENDM

**Вызов макрокоманд**

Для вызова макрокоманды нужно поместить ее имя в исходный код программы и при необходимости указать передаваемые ей значения:

*Имя\_макрокоманды Значение-1, Значение-2, ...*

*Имя\_макрокоманды* должно быть определено в исходном коде программы до ее вызова. Каждое значение является обычной текстовой строкой, которое подставляется вместо соответствующего параметра макрокоманды. Порядок передачи значений параметров должен соответствовать порядку их следования в макроопределении, однако число значений необязательно должно соответствовать числу параметров макрокоманды.

1. **Директивы условного ассемблирования**

Существует ряд специальных директив условного ассемблирования, которые можно применять внутри макроопределений для повышения их гибкости. Общий синтаксис этих директив выглядит так:

IF *Условие*

*Операторы*

[ELSE

*Операторы]*

ENDIF

Список часто используемых директив условного ассемблирования приведен в табл. 1. Если в их описании сказано, что *ассемблирование разрешено*, это означает, что в исходный текст программы будут включены все операторы, расположенные следом за рассматриваемой директивой и до первой директивы ENDIF. Следует особо отметить, что директивы, приведенные в табл. 1, обрабатываются на этапе компиляции программы, а не ее выполнения.

Таблица 1. Директивы условного ассемблирования

|  |  |
| --- | --- |
| **Директива** | **Описание** |
| I F выражение | Ассемблирование разрешено, если значение выражения истинно. В выражении могут использоваться следующие операторы отношения: LT, GT, EQ, NE, LE И GE |
| IFB <аргумент> | Ассемблирование разрешено, если аргумент имеет пустое значение. При этом имя аргумента должно быть заключено в угловые скобки |
| IFNB < аргумент> | Ассемблирование разрешено, если аргументу присвоено не пустое значение. При этом имя аргумента должно быть заключено в угловые скобки |
| IFIDN <арг1>,<арг2> | Ассемблирование разрешено, если аргументы полностью идентичны друг другу. Сравнение выполняется с учетом регистра символов |
| IFIDNI <арг1>,<арг2> | Ассемблирование разрешено, если аргументы равны друг другу. Сравнение выполняется без учета регистра символов |
| IFDIF <арг1>, <арг2> | Ассемблирование разрешено, если аргументы не идентичны друг другу. Сравнение выполняется с учетом регистра символов |
| I FDI FI < арг1 > , < арг2> | Ассемблирование разрешено, если аргументы не равны друг другу. Сравнение выполняется без учета регистра символов |
| IFDEF имя | Ассемблирование разрешено, если указанное имя определено |
| IFNDEFHM\* | Ассемблирование разрешено, если указанное имя не определено |
| ENDIF | Завершает блок директив условного ассемблирования |
| ELSE | Ассемблирование разрешено, если в предыдущей директиве условие было ложно |
| EXITM | Немедленно завершает работу макрокоманды, при этом оставшиеся операторы макроопределения не обрабатываются |

**Проверка пропущенных аргументов**

Часто бывает, что если при вызове макрокоманды не указать один или несколько аргументов, препроцессор сгенерирует некорректные ассемблерные команды. Поэтому при написании макроопределения у программиста должны быть средства проверки пропущенных аргументов. Например, если вызвать макрокоманду mHriteStr без параметров, это приведет к генерации некорректной команды загрузки смещения в регистр EDX. Ниже приведен листинг, сгенерированный компилятором ассемблера, в котором зафиксирован факт отсутствия операнда и выведено соответствующее сообщение об ошибке:

mWriteStr

1 push edx

1 mov edx,OFFSET

Macro2.asm(18) : error A2081: missing operand after unary operator (Отсутствует операнд после унарного оператора)

1 call WriteString

1 pop edx

Чтобы избежать подобного рода ошибок, связанных с отсутствием операндов при вызове макрокоманды, следует воспользоваться директивой *IFB (If Blank, или если пусто)*. Она возвращает истинное значение, только если указанный в ней аргумент содержит пустое значение. Кроме того, можно также воспользоваться директивой *IFNB (If Not Blank, или если не пусто)*. Она возвращает истинное значение, если указанный в ней аргумент содержит непустое значение.

Создадим новую версию макроопределения inWriteStr, которая при пропуске аргумента будет выводить сообщение об ошибке во время компиляции программы:

mWriteStr MACRO string

IFB <string>

ECHO ------------------------------------------------------------

ECHO \*Ошибка: пропущен параметр в макрокоманде mWriteStr

ECHO \* (Код генерироваться не будет)

ECHO ------------------------------------------------------------

EXITM

ENDIF

push edx

mov edx,OFFSET string

call WriteString

pop edx

ENDM

Директива ECHO предназначена для вывода информационных сообщений на экран во время трансляции программы. Директива EXITM предписывает препроцессору немедленно прекратить обработку макрокоманды. При этом следующие за ней директивы до конца макроопределения пропускаются.

**2. ВЫПОЛНЕНИЕ РАБОТЫ**

Написать процедуру для выполнения следующей операции:

1. Поиск наибольшего числа в массиве целых чисел;

2. Сортировка массива целых чисел;

3. Сравнения двух строк;

4. Сравнение двух массивов целых чисел (размеры массивов одинаковые)

5. Суммирование двух векторов.

6. Перевод целого числа из десятичной системы счисления в двоичную.

7. Выделение из строки символов подстроки.

8. Конкатенация двух строк символов.

9. Разложение целого числа на множители.

10. Поиск максимального аргумента процедуры.

Для тестирования разработанной процедуры написать программу на ассемблере для вызова тестируемой процедуры с различными аргументами и вывода на экран полученных результатов.

Предусмотреть все возможные проверки правильности указания операндов (наличие операндов, совпадение имен, определение имен и т. д.):

1. Написать макроопределение mMove32 с двумя параметрами – 32 разрядными операндами, размещенными в ОП. Макрокоманда должна источник поместить в приемник.
2. Написать макроопределение mMult32 с двумя параметрами для умножения двух беззнаковых 32-разрядных чисел
3. Написать макроопределение mWriteStrCol с двумя параметрами, которая выводит на экран нуль-завершенную строку указанным цветом.
4. Написать макроопределение mWriteSimMulCol c тремя параметрами для вывода на экран указанного символа заданное число раз указанным цветом.
5. Написать макроопределение mReadInt для ввода с клавиатуры целого числа без знака и вывода его на экран. Для тестирования разработанного макроопределения написать программу, использующую макрокоманду с различными аргументами и разным количеством аргументов.

**Порядок проведения лабораторной работы:**

1. Изучить краткие теоретические сведения.
2. Выполнить задания согласно варианту.
3. Оформить отчет по лабораторной работе.

**Оформление отчета по лабораторной работе:**

Отчет оформляем согласно Методическим указаниям по оформлению отчета.

**Вопросы для подготовки к зачету:**

1. Директива PROC. Определение процедуры. Структура процедуры ассемблера.
2. Возврат из процедуры.
3. Способы вызова процедур.
4. Способы передачи значений аргументов в процедуру.
5. Макрокоманды. Отличие подпрограмм и макрокоманд.
6. Определение макрокоманд
7. Директивы условного ассемблирования.
8. Проверка пропущенных аргументов макрокоманды.

**Рекомендуемая литература:**

1. Зубков С.В. Assembler для DOS, Windows и UNIX. 3-е изд., стер. – М.: ДМК Пресс ; СПб.: Питер, 2004. – 608 с. : ил. – (Серия «Для программистов»).
2. Крупник А.Б. Изучаем Ассемблер – СПб.: Питер, 2005. – 249 с.: ил.
3. Жуков А.В., Авдюхин А.А. Ассемблер. – СПб.: БХВ-Петербург, 2002. – 448 с.: ил.