ПРАКТИЧЕСКОЕ ЗАНЯТИЕ_5

МАКРОСРЕДСТВА АССЕМБЛЕРА

ОПЕРАЦИИ ВВОДА/ВЫВОДА В АССЕМБЛЕРЕ

Цель работы:

1. Изучение структуры и операторов макроопределения.
2. Написание макроопределения ввода/вывода.

МАКРООПРЕДЕЛЕНИЯ

Макроопределения представляют собой последовательность операторов на языке ассемблера (команд и псевдооператоров), которые могут несколько раз появиться в программе.

Основные правила подготовки макроопределения:

1. Макроопределение должно выполнять только одну функцию.
2. Макроопределение должно обеспечить изоляцию данных. Это означает, что все регистры, используемые в Макроопределении, перед их использованием должны быть сохранены в стеке, а перед выходом из Макроопределения должны быть восстановлены из стека.
3. Каждое макроопределение должно иметь одну точку входа и одну точку выхода.
4. Для эффективности программирования Макроопределения должны быть простыми и легко понимаемыми.
5. Для быстродействия Макроопределения должны занимать минимум памяти.
6. Если правила 4 и 5 конфликтуют, предпочтение отдается правилу 4.

Вызывается макроопределение с помощью макрокоманды. Только в этом случае оно выполняется. При ассемблировании макрокоманда замещается макрорасширением. Макрорасширение это макроопределение, в котором формальные параметры заменены фактическими.

СОСТАВ МАКРООПРЕДЕЛЕНИЙ

Каждое макроопределение имеет три части:

1. ***Заголовок*** - псевдооператор ***MACRO***, в поле метки которого указано *имя* макроопределения, а в поле операнда - необязательный ***список формальных параметров***. В списке формальных параметров указываются переменные - входные параметры, которые можно изменять при каждом вызове макроопределения.

2. ***Тело*** - последовательность операторов Ассемблера (команд и псевдооператоров), которые задают действия, выполняемые макроопределением.

3. ***Концевик*** - псевдооператор ***ENDM***, который отмечает конец макроопределения.

Синтаксис макроопределения следующий:

```
Имя   MACRO [список формальных параметров]  ; начало макроопределения
            Тело макроопределения, использующее формальные параметры
      ENDM                                  ;ключевое слово – конец макроопределения
```

Имя используется для вызова макроопределения в макрокоманде с фактическими параметрами.

Имя [список фактических параметров]

Пример 1: макроопределение для сложения значений размером в слово:ADD_WORDS

```
MACRO TERM1, TERM2, SUM
        MOV AX, TERM1
        ADD AX, TERM2
        MOV  SUM, AX
ENDM
```

Для Ассемблера безразлично, что будет указано в качестве параметров макроопределения; имена регистров, ячейки памяти или непосредственные значения (конечно, непосредственное значение нельзя использовать в качестве операнда **SUM).**

Пример 2: макрокоманды вызова:

```
ADD_WORDS  CX,  DX,  BX     ; сложить содержимое регистров CX и DX,
                            ;  результатпоместить в регистр BX
ADD_WORDS 134, 357, CX      ; сложить 134 и 357, результат в регистр CX
ADD_WORDS  X,  Y,  RES      ; сложить значения переменных X и Y,
                            ; результатпоместить в переменную RES
```

Макрокоманды используются в тексте программы. Каждая макрокоманда в тексте программы заменяется **макрорасширенем**.

Вместо первой макрокоманды:

```
+       MOV     AX, CX
+       ADD     AX, DX
+       MOV     BX, AX
```

Вместо второй макрокоманды:

```
+       MOV     AX, 134
+       ADD     AX, 357
+       MOV     CX, AX
```

Вместо третьей макрокоманды:

```
+       MOV     AX, X
+       ADD     AX, Y
+       MOV     RES, AX
```

В листинге программы макрорасширения помечаются знаком +.

ПСЕВДООПЕРАТОРЫ МАКРОАССЕМБЛЕРА

ПСЕВДООПЕРАТОР LOCAL

Если Ваше макроопределение содержит помеченные команды или псевдооператоры, то Вы должны указать Ассемблеру, чтобы он изменял метки при каждом расширении макроопределения. В противном случае Вы столкнетесь с сообщениями об ошибках **Symbol is Multi-Defined** (символ многократно определен). Псевдооператор **LOCAL** сообщает Ассемблеру, какие метки должны изменяться при каждом расширении.

Пример 3: макроопределение ***MSUM*** выполняет сложение элементов массива ***MAS*** длиной ***LEN.*** Значение суммы помещается в переменную ***MASS.*** Указание метки ***NEXT*** в операторе ***LOCAL*** позволяет нам пользоваться этим макроопределением в программе более одного раза:

```
MSUM MACRO MAS, LEN, MASS
LOCAL  NEXT
        PUSH    CX              ;сохранить текущее значение CX
        MOV     CX,LEN          ;поместить счетчик в CX
        MOV AX,0                ;обнулить регистр для суммы
        MOV SI,0                ;обнулить индексный регистр
NEXT:   ADD AX,MAS[SI]          ;добавить очередной элемент
         INC SI                 ;увеличить значение индексного регистра
         LOOP  NEXT             ;повторять, пока счетчик не обратится в 0
```

```
        MOV MASS, AX
        POP   CX              ;восстановить исходное значение CX
ENDM
```

Оператор **LOCAL** следует непосредственно за оператором MACRO.

ЗАДАНИЕ МАКРООПРЕДЕЛЕНИЙ В ИСХОДНЫХ ПРОГРАММАХ

Существует два способа использования макроопределений: их можно задавать в начале программы или считывать в программу из отдельного файла с библиотекой макроопределений.

Структура программы с макроопределениями

```
ADD_WORDS MACRO TERM1, TERM2, SUM
        MOV     AX, TERM1
        ADD     AX, TERM2
        MOV     SUM, AX
ADD_WORDS ENDM
```

```
.stack 100
.data
        x1 db 13
        x2 db 11
        ys db ?
        a1 db 44
        b1 db 33
        ab db ?
.code
        start:
                mov bx,@data
                mov ds,bx
              ADD_WORDS x1,x2, ys
              ADD_WORDS a1,b1, ab
        end start
```

Примечание: Задание макроопределений непосредственно в тексте программы удобно, но имеет недостаток - ими можно воспользоваться только в этой программе. Чтобы они были доступны и другим программам, их надо помещать в библиотеку макроопределений.

Задание 1. Измените программу, разработанную на практическом занятии №2, заменив повторяющиеся действия макросами вывода строки на экран и установки курсора в заданную позицию.

Примеры макросов:

Макрос вывод на экран в заданную позицию

Примечание. Для установки курсора в заданную позицию используется прерывание **10H**.
Макрос очистку экрана с помощью функции **AH=6.**

```
Set_cursor MACRO row, col
        Push ax
        Push bx
        Push cx             ;Все регистры в стек
        Push dx
        mov  ax, 0600h      ;(al=0) очистить весь экран
        mov  bh, 07         ; атрибут нормальный ч/б
        mov  cx, 0000       ; координаты от 00,00
        mov  dx, 184fh      ; до 24,79 (весь экран)
        int  10h

        mov   ah, 02        ; Установка курсора
        mov   bh, 00        ; страница
        mov   dh, row       ; номер строки в DH
        mov   dl, col       ; номер столбца в DL
        int   10h

        pop ax
        pop bx
        pop cx
        pop dx              ;Все регистры из стека
       Set_cursor ENDM
```

Макрос вывода заданной строки string на экран

```
mWriteStr macro string
    push ax             ; Сохранение регистров, используемых в макросе, в стек
    push dx

    mov ah, 09h         ; 09h - функция вывода строки на экран
    mov dx, offset string
    int 21h

    pop dx              ; Перенос сохранённых значений обратно в регистры
    pop ax
endm mWriteStr
```

Задание 2. Измените программу, разработанную в лабораторной работе №3 «Выполнение арифметических операций над числами без знака и со знаком», дополнив ее макросами «Ввода целого числа в регистр AX в 10-ричной системе счисления», и «Вывода целого числа в регистр AX в 10-ричной системе счисления», приведенными ниже.

Макрос ввода целого числа в регистр AX в 10-ричной системе счисления

```
mReadAX macro buffer, size
local input, startOfConvert, endOfConvert
    push bx  ;Сохранение регистров, используемых в макросе, в стек
    push cx
    push dx

input:
    mov [buffer], size          ;Задаём размер буфера
    mov dx, offset [buffer]
    mov ah, 0Ah                 ;0Ah - функция чтения строки из консоли
    int 21h

    mov ah, 02h             ;02h - функция вывода символа на экран
    mov dl, 0Dh
    int 21h                 ;Переводим каретку на новою строку

    mov ah, 02h             ;02h - функция вывода символа на экран
    mov dl, 0Ah
    int 21h                 ;Переносим курсор на новою строку

    xor ah, ah
    cmp ah, [buffer][1]  ;Проверка на пустую строку
    jz input             ;Если строка пустая - переходим обратно к вводу

    xor cx, cx
    mov cl, [buffer][1]  ;Инициализируем переменную счетчика

    xor ax, ax
    xor bx, bx
    xor dx, dx
    mov bx, offset [buffer][2] ;bx= начало строки
;(строка начинается со второго байта)
    cmp [buffer][2], '-'        ;Проверяем, отрицательное ли число
    jne startOfConvert          ;Если отрицательное - пропускаем минус
    inc bx
    dec cl

startOfConvert:
    mov dx, 10
    mul dx          ;Умножаем на10 перед сложением с младшим разрядом
    cmp ax, 8000h ;Если число выходит за границы, то
    jae input     ;  возвращаемся на ввод числа

    mov dl, [bx]            ;Получаем следующий символ
    sub dl, '0'             ;Переводим его в числовой формат

    add ax, dx              ;Прибавляем к конечному результату
    cmp ax, 8000h           ;Если число выходит за границы, то
    jae input               ;  врзвращаемся на ввод числа

    inc bx                  ;Переходим к следующему символу
    loop startOfConvert

    cmp [buffer][2], '-'  ;Ещё раз проверяем знак
    jne endOfConvert      ;Если знак отрицательный, то
    neg ax                ;  инвертируем число

endOfConvert:
    pop dx                  ;Перенос сохранённых значений обратно в регистры
    pop cx
    pop bx
endm mReadAX
```

Макрос вывода на экран содержания регистра ax в 10-ричной системе счисления

```
mWriteAX macro
local convert, write
    push ax       ; Сохранение регистров, используемых в макросе, в стек
    push bx
    push cx
    push dx
    push di

    mov cx, 10    ; cx - основание системы счисления
    xor di, di    ; di - количество цифр в числе

    or ax, ax     ; Проверяем, равно ли число в ax нулю и устанавливаем флаги
    jns convert
; Переход к конвертированию, если число в ax положительное
    push ax

    mov dx, '-'
    mov ah, 02h   ; 02h - функция вывода символа на экран
    int 21h       ; Вывод символа "-"

    pop ax
    neg ax        ; Инвертируем отрицательное число

convert:
    xor dx, dx

    div cx        ; После деления dl = остатку от деления ax на cx
    add dl, '0'   ; Перевод в символьный формат
    inc di        ; Увеличиваем количество цифр в числе на 1

    push dx       ; Складываем в стек

    or ax, ax     ; Проверяем, равно ли число в ax нулю и устанавливаем флаги
    jnz convert   ; Переход к конвертированию, если число в ax не равно нулю

write:            ; Вывод значения из стека на экран
    pop dx        ; dl = очередной символ

    mov ah, 02h
    int 21h       ; Вывод очередного символа
    dec di        ; Повторяем, пока di <> 0
    jnz write

    pop di        ; Перенос сохранённых значений обратно в регистры
    pop dx
    pop cx
    pop bx
    pop ax
endm mWriteAX
```