Лабораторная работа №2. Использование условных и циклических конструкций

Синтаксис объявления меток

Метка представляет собой символическое имя, вместо которого компилятор подставляет адрес. В программе на ассемблере можно присвоить имя любому адресу в коде или данных. Обычно метки используются для организации переходов, циклов или каких-то манипуляций с данными. По сути имена переменных, объявленных с помощью директив объявления данных, тоже являются метками. Но с ними компилятор дополнительно связывает размер переменной. Метка объявляется очень просто: достаточно в начале строки написать имя и поставить двоеточие. Например:

```
m1: mov ax,4C00h
    int 21h
```

Теперь вместо имени m1 компилятор везде будет подставлять адрес команды `mov ax,4C00h`. Можно объявлять метку на пустой строке перед командой:

```
exit_app:
    mov ax,4C00h
    int 21h
```

Имя метки может состоять из латинских букв, цифр и символов подчёркивания, но должно начинаться с буквы. Имя метки должно быть уникальным. В качестве имени метки нельзя использовать директивы и ключевые слова компилятора, названия команд и регистров (в этом случае FASM покажет сообщение об ошибке). FASM различает регистр символов в именах меток. Можно также объявлять несколько меток на один адрес. Например:

```
no_error:
exit_app:
m1: mov ax,4C00h
```

Команда LOOP

Для организации цикла предназначена команда LOOP. У этой команды один операнд — имя метки, на которую осуществляется переход. В качестве счётчика цикла используется регистр CX. Команда LOOP выполняет декремент CX, а затем проверяет его значение. Если содержимое CX не равно нулю, то осуществляется переход на метку, иначе управление переходит к следующей после LOOP команде.

Содержимое CX интерпретируется командой как число без знака. В CX нужно помещать число, равное требуемому количеству повторений цикла. Понятно, что максимально может быть 65535 повторений. Ещё одно ограничение связано с дальность перехода. Метка должна находиться в диапазоне -127…+128 байт от команды LOOP (если это не так, FASM сообщит об ошибке).

Пример

В качестве примера приведём программу, которая будет печатать все буквы английского алфавита. ASCII-коды этих символов расположены последовательно, поэтому можно выводить их в цикле. Для вывода символа на экран используется функция DOS 02h (выводимый байт должен находиться в регистре DL).

```
use16                   ;Генерировать 16-битный код
org 100h                ;Программа начинается с адреса 100h
```

```
    mov ah,02h          ;Для вызова функции DOS 02h - вывод символа
    mov dl,'A'          ;Первый выводимый символ
    mov cx,26           ;Счётчик повторений цикла
metka:
    int 21h             ;Обращение к функции DOS
    inc dl              ;Следующий символ
    loop metka          ;Команда цикла

    mov ah,09h          ;Функция DOS 09h - вывод строки
    mov dx,press        ;В DX адрес строки
    int 21h             ;Обращение к функции DOS

    mov ah,08h          ;Функция DOS 08h - ввод символа без эха
    int 21h             ;Обращение к функции DOS

    mov ax,4C00h        ;\
    int 21h             ;/ Завершение программы
;-------------------------------------------------------
press:
    db 13,10,'Press any key...$'
```

Команды «int 21h» и «inc dl» (строки 8 и 9) будут выполняться в цикле 26 раз. Для того, чтобы программа не закрылась сразу, используется функция DOS 08h — ввод символа с клавиатуры. Перед этим выводится предложение нажать любую кнопку. Для примера адрес строки объявлен с помощью метки. Символы с кодами 13 и 10 обозначают переход на следующую строку (символ 13(0Dh) называется CR — Carriage Return — возврат каретки, а символ 10(0Ah) LF — Line Feed — перевод строки.

Безусловные переходы

Безусловный переход — это переход, который выполняется всегда. Безусловный переход осуществляется с помощью команды JMP. У этой команды один операнд, который может быть непосредственным адресом (меткой), регистром или ячейкой памяти, содержащей адрес. Примеры безусловных переходов:

```
jmp metka    ;Переход на метку
jmp bx       ;Переход по адресу в BX
jmp word[bx] ;Переход по адресу, содержащемуся в памяти по адресу в BX
```

Условные переходы

Условный переход осуществляется, если выполняется определённое условие, заданное флагами процессора (кроме одной команды, которая проверяет CX на равенство нулю). Если условие не выполняется, то управление переходит к следующей команде.

Существует много команд для различных условных переходов. Также для некоторых команд есть синонимы (например, JZ и JE — это одно и то же). Для наглядности все команды условных переходов приведены в таблице:

Команда	Переход, если	Условие перехода
JZ/JE	нуль или равно	ZF=1
JNZ/JNE	не нуль или не равно	ZF=0
JC/JNAE/JB	есть переполнение/не выше и не равно/ниже	CF=1
JNC/JAE/JNB	нет переполнения/выше или равно/не ниже	CF=0
JP	число единичных бит чётное	PF=1
JNP	число единичных бит нечётное	PF=0
JS	знак равен 1	SF=1
JNS	знак равен 0	SF=0
JO	есть переполнение	OF=1
JNO	нет переполнения	OF=0
JA/JNBE	выше/не ниже и не равно	CF=0 и ZF=0
JNA/JBE	не выше/ниже или равно	CF=1 или ZF=1
JG/JNLE	больше/не меньше и не равно	ZF=0 и SF=OF
JGE/JNL	больше или равно/не меньше	SF=OF
JL/JNGE	меньше/не больше и не равно	SF≠OF
JLE/JNG	меньше или равно/не больше	ZF=1 или SF≠OF
JCXZ	содержимое CX равно нулю	CX=0

У всех этих команд один операнд — имя метки для перехода. Обратите внимание, что некоторые команды применяются для беззнаковых чисел, а другие — для чисел со знаком. Сравнения «выше» и «ниже» относятся к беззнаковым числам, а «больше» и «меньше» — к числам со знаком. Для беззнаковых чисел признаком переполнения будет флаг CF, а соответствующими командами перехода JC и JNC. Для чисел со знаком о переполнении можно судить по состоянию флага OF, поэтому им соответствуют команды перехода JO и JNO. Команды переходов не изменяют значения флагов.

Команды CMP и TEST

Часто для формирования условий переходов используются команды CMP и TEST. Команда CMP предназначена для сравнения чисел. Она выполняется аналогично команде SUB: из первого операнда вычитается второй, но результат не записывается на место первого операнда, изменяются только значения флагов. Например:

```
cmp al,5      ;Сравнение AL и 5
jl c1         ;Переход, если AL < 5 (числа со знаком)
cmp al,5      ;Сравнение AL и 5
jb c1         ;Переход, если AL < 5 (числа без знака)
```

Команда TEST работает аналогично команде AND (если оба бита равны 1, то результат равен 1, иначе результат равен 0), но также как и в CMP результат не сохраняется, изменяются только флаги. С помощью этой команды можно проверить состояние различных битов операнда. Например:

```
test bl,00000100b ;Проверить состояние 2-го бита BL
jz c2             ;Переход, если 2-й бит равен 0
```

Пример

```
use16
org 100h
    jmp start  ;Безусловный переход на метку start

;-----------------------------------;
menu db '1 - print name',13,10    ;
     db '2 - print surname',13,10 ;
     db '0 - exit',13,10,'$'      ;
                          ;
name db 13,10,'Ivan',13,10,'$'             ;
surname db 13,10,'Ivanov',13,10,'$'             ;
                          ;
select    db 13,10,"Select> $"            ;
;-----------------------------------;

start:
    mov ah, 9    ;\
    mov dx, menu  ; > Вывод меню
    int 21h      ;/

select_menu:
    mov ah, 9       ;
    mov dx, select  ;Вывод строки ‘Select’
    int 21h

    mov ah, 1       ;Функция DOS 01h - ввод символа
    int 21h         ;Введенный символ помещается в AL
    cmp al, '1'     ;Сравнение введенного символа с '1'
    je    c1        ;Переход, если равно
    cmp al, '2'     ;Сравнение введенного символа с '2'
    je    c2        ;Переход, если равно
    cmp al, '0'     ;Сравнение введенного символа с '0'
    je    exit      ;Переход, если равно
    jmp select_menu ;Безусловный переход
c1:
    mov ah, 9
    mov dx, name
    int 21h         ;Вывод имени
    jmp start       ;Переход к началу программы

c2:
    mov ah, 9
    mov dx, surname
    int 21h         ;Вывод фамилии
    jmp start       ;Переход к началу программы

exit:
    int 20h         ;Выход из программы
```

```
mov ah, 8      ;Задержка экрана
int 21h
int 20h
```

УКАЗАНИЯ К РАБОТЕ

Объявите одномерный целочисленный массив (без ввода с клавиатуры). Элементы массива размером в байт. Результаты представьте в отладчике Turbo Debugger (td.exe).

Пример:

Найти максимальный элемент массива.

```
use16
org 100h
    mov al, [a]     ;Первый элемент в регистр AL
    mov [max],al    ;Из регистра – в переменную max

    mov bx, a       ;В BX – адрес начала массива
    mov cx, 5       ;В CX – количество элементов
    cycle:
     mov al, [bx]   ;В AL помещаем элемент с адресом BX
     cmp al, [max]  ;Сравниваем его с max
     jle nextstep   ;Если меньше или равно – переходим на nextstep
        mov [max], al    ;иначе - заменяем max
     nextstep:
        inc bl      ;Переходим к следующему элементу
    loop cycle      ;Если CX <> 0 – переходим к метке cycle

    mov al, [max]   ;По выходу из цикла кладем maxв AL

mov ah, 8           ;задержка экрана
int 21h
int 20h

a db 1,2,9,8,5      ;Объявляем массив a
max rb 1            ;Резервируем память под переменную max
```

ВАРИАНТЫ

1. Найти минимальный четный элемент массива;
2. Подсчитать количество четных элементов массива и определить индекс последнего четного.
3. Найти за один проход два максимума в массиве;
4. Определить максимум и минимум массива;
5. Определить количество четных элементов в массиве и индекс первого четного;
6. Найти максимальный отрицательный четный элемент в массиве;
7. Найти отдельно сумму четных и сумму нечетных элементов;
8. Определить число и сумму элементов, расположенных внутри отрезка [a, b] (a, b заданы);
9. Найти сумму нечетных элементов массива;
10. Найти минимальный нечетный элемент;
11. Определить индексы первого и последнего отрицательных элементов;

12. Обменять первый четный с последним нечетным элементом в массиве;
13. Найти минимальный четный элемент в массиве;
14. Найти сумму положительных и отрицательных элементов;
15. Найти первый положительный элемент;
16. Обменять максимальный элемент с первым нулевым;
17. Найти максимальный элемент, делящийся на 3;
18. Обменять максимальный отрицательный и минимальный положительный;
19. Найти первый и последний нулевой элементы;
20. Найти минимальный элемент массива;
21. Найти максимальный элемент, который делится на 5;
22. Найти максимальный элемент массива, который находится на четной позиции;
23. Найти сумму отдельно элементов с четными и с нечетными индексами;
24. Обменять местами первый и второй отрицательные элементы;
25. Найти максимальный нечетный элемент;
26. Найти сумму максимального и минимального элементов.