Лабораторная работа 3. Директивы управления

§1. Введение

Описание данных

Чтобы описать данные или зарезервировать для них место, используйте одну из директив, перечисленных в таблице 1.3. За директивой описания данных должно следовать одно или несколько числовых значений, разделенных запятыми. Эти выражения определяют значения для простейших элементов данных, размер которых зависит от того, какая директива используется. Например "db 1,2,3" описывает три байта со значениями 1, 2 и 3 соответственно.

Директивы "du" и "db" также поддерживают сроки любой длины, заключенные в кавычки, которые будут конвертированы в последовательность байтов, если использована директива "db", или в последовательность слов с нулевым верхним байтом, если использована директива "du". Например, `"db 'abc'"` определяет три байта со значениями 61, 62 и 63.

Директива "dp" или её синоним "df" допускают, чтобы значения состояли из двух числовых выражений, разделенных двоеточием, где первое значение - это верхнее слово, а второе - это нижнее двойное слово значения дальнего указателя. Также "dd" допускает такие указатели, состоящие из двух слов, разделенных двоеточием, и "dt" допускает слово и четверное слово, разделенные двоеточием, четверное слово запоминается первым. Директива "dt" с одним параметром допускает только значения с плавающей точкой и создает данные в FPU-формате двойной расширенной точности.

Все вышеперечисленные директивы поддерживают использование специального оператора "dup" для создания копий данных значений. Количество дубликатов должно стоять перед этим оператором, а их значение должно стоять после - это может быть даже цепь значений, разделенных запятыми, но эта цепь должна быть заключена в скобки, например "db 5 dup (1,2)" определяет пять копий данной последовательности из двух байт.

За директивой резервирования данных должно следовать одно числовое выражение, значение которого определяет количество резервируемых ячеек установленного размера. Все директивы описания данных также поддерживают значение "?", которое значит, что этой ячейке не должно быть присвоено какое-то значение. Эффект от этой директивы такой же, как от директивы резервирования данных. Неинициализированные данные не могут быть включены в файл вывода, и, таким образом, их значения всегда будут считаться неизвестными.

Таблица 1.3 Директивы данных

Размер (байты)	Определение данных	Резервирование данных
1	db file	rb
2	dw du	rw
4	dd	rd
6	dp df	rp rf
8	dq	rq
10	dt	rt

Константы и метки

В числовых выражениях вместо чисел вы также можете использовать константы и метки. Чтобы назначить их, используйте специальные директивы. Каждая метка может быть определена только однажды и она будет доступна из любой части кода (даже перед местом, где она была определена). Константа может быть переопределена много раз, но в этом случае она будет доступна только после присвоения значения и всегда будет равна значению из последнего определения перед местом, в котором она использована. Если константа определена лишь однажды, она, так же как и метка, доступна из любой части кода.

Определение константы состоит из имени константы, знака "=" и числового выражения, которое после вычисления становится значением константы. Это значение всегда вычисляется в то же время, что и определение константы. Например, с помощью директивы "count = 17" вы можете определить константу "count" и после использовать её в инструкциях ассемблера, таких как "mov cx,count" - которая превратится в "mov cx,17" во время процесса компиляции.

Существуют разные способы определения меток. Простейший из них - двоеточие после названия метки. За этой директивой на той же строке даже может следовать другая инструкция. Она определяет метку, значение которой равно смещению точки, в которой она определена. Этот метод обычно используется, чтобы пометить места в коде. Другой способ - это следование за именем метки (без двоеточия) какой-нибудь директивы описания данных. Метке присваивается значение адреса начала определенных в директиве данных и запоминается компилятором как метка для данных с размером ячейки, заданной директивой из таблицы 1.3.

Метка может быть обработана как константа со значением, равным смещению помеченного кода или данных. Например, если вы определяете данные, используя помеченную директиву "`char db 224`", для того, чтобы поместить адрес начала этих данных в регистр BX, вам нужно использовать инструкцию "`mov bx,char`", а для того, чтобы поместить в регистр DL значение байта, на который ссылается "char", нужно использовать "`mov dl,[char]`" (или "`mov dl,ptr char`").

Если вы попытаетесь ассемблировать "`mov ax,[char]`", FASM выдаст ошибку, так как он сравнивает размеры операндов, которые должны быть равны. Вы можете принудительно проассемблировать эту инструкцию, изменяя размер операнда: "`mov ax, word [char]`", но помните, что эта инструкция прочитает два байта, начинающихся с адреса "char", тогда как он был определен как один байт.

Последний и самый гибкий способ задания меток - это использование директивы "label". За этой директивой должно следовать имя метки, далее, опционально, размер оператора (может предваряться двоеточием), и далее, также опционально, оператор "at" и числовое выражение, определяющее адрес, на который данная метка должна ссылаться. Например, "label wchar word at char" определяет новую метку для 16-битных данных по адресу "char". Теперь инструкция "mov ax,[wchar]" после компиляции будет выглядеть так же, как "mov ax,word [char]". Если адрес не указан, директива "label" будет ссылаться на текущий адрес. Таким образом, "mov [wchar],57568" скопирует два байта, тогда как "mov [char],224" скопирует один байт на тот же адрес.

Метка, имя которой начинается с точки, обрабатывается как локальная, и её имя прикрепляется к имени последней глобальной метки (с названием, начинающемся с чего угодно, кроме точки) для создания полного имени этой метки. Так, вы можете использовать короткое имя (начинающееся с точки) где угодно перед следующей глобальной меткой, а в других местах вам придется пользоваться полным именем. Метки, начинающиеся с двух точек - исключения. Они имеют свойства глобальных, но не создают новый префикс для локальных меток.

"@@" обозначает анонимную метку, вы можете определить её множество раз. Символ "@b" (или эквивалент "@r") ссылается на ближайшую предшествующую анонимную метку, а символ "@f"

ссылается на ближайшую после неё анонимною метку. Эти специальные символы нечувствительны к регистру.

Числовые выражения

В предыдущих примерах все числовые выражения были обычными числами, константами или метками. Но они могут быть более сложными, использовать арифметические или логические операторы для вычисления во время компиляции. Все эти операторы с их значениями приоритета перечислены в таблице 1.4.

Операции с высшим приоритетом выполняются первыми, однако вы, конечно, можете изменить такой образ действий, заключив некоторые части выражения в скобочки. "+", "-", "*" и "/" - это стандартные арифметические операции, "mod" вычисляет остаток от деления нацело. "and", "or", "xor", "shl", "shr" и "not" совершают те же логические операции, что и инструкции ассемблера с такими же названиями. "rva" характерна только для формата вывода PE и производит превращение адреса в RVA.

Числа в выражениях по умолчанию обрабатываются как десятичные, двоичные числа должны иметь "b" в конце, восьмеричные числа должны заканчиваться на букву "o", шестнадцатеричные цифры должны начинаться символами "0x" (как в языке C), или символом "$" (как в языке Pascal) или должны заканчиваться буквой "h". Также заключенная в кавычки строка при включении в выражение будет конвертирована в число - первый символ станет минимальным значащим байтом числа. Числовые выражения, используемые как значения адреса, могут также содержать любой из общих регистров, используемых для адресации, они могут быть сложены или умножены на подходящие значения так, как это позволено в инструкциях архитектуры x86.

Любое численное выражение также может состоять из одного значения с плавающей точкой (flat assembler не может производить во время компиляции операции с плавающей точкой) в научной записи. Для распознания компилятором, эти значения должны содержать в конце букву "f", либо включать в себя по крайней мере один символ "." или "E". Так, "1.0", "1E0" и "1f" определяют одно и то же значение с плавающей точкой, когда как просто "1" определяет целочисленное значение.

Таблица 1.4 Арифметические и логические операторы в порядке приоритета

Приоритет	Операторы
0	+ -
1	* /
2	mod
3	and or xor
4	shl shr
5	not
6	rva

§2. Директивы управления

Этот параграф описывает директивы, которые управляют процессом ассемблирования. Эти директивы выполняются во время ассемблирования и могут делать так, чтобы некоторые блоки инструкций ассемблировались по-разному или не ассемблировались вовсе.

2.1 Условное ассемблирование

С помощью директивы "if" можно ассемблировать или не ассемблировать блок инструкций в зависимости от выполнения условия. За ней должно следовать логическое выражение, определяющее условие. Инструкции на следующих строках ассемблируются, только если это условие выполняется, иначе они пропускаются. Опциональная директива "elseif" со следующим за ней логическим выражением, определяющим дополнительное условие, начинает следующий блок инструкций, который ассемблируется, если предыдущие условия не выполняются, а данное дополнительное условие выполняется. Опциональная директива "else" начинает блок инструкций, которые ассемблируются, если не выполняется ни одно из условий. "end if" заканчивает последний блок инструкций.

Вы должны помнить, что директива "if" обрабатывается на стадии ассемблирования и поэтому не влияет на директивы препроцессора, такие как определения символьных констант и макроинструкции - когда ассемблер распознает директиву "if", весь препроцессинг уже закончен.

Логическое выражение состоит из логических значений и логических операторов. Логические операторы выглядят так: "~" для логического отрицания, "&" для логического И, "|" для логического ИЛИ. Отрицание имеет высший приоритет. Логическое значение может быть числовым выражением, оно будет считаться ложным в случае равенства нулю, иначе оно будет истинным. Для создания логического значения можно сравнить два числовых выражения, используя один из следующих операторов: "=" (равно), "<" (меньше), ">" (больше), "<=" (меньше или равно), ">=" (больше или равно), "<>" (не равно).

Следующий простой пример использует константу "count" которая должна быть определена где-то в коде:

```
if count>0
    mov cx,count
    rep movsb
end if
```

Эти две инструкции будут ассемблированы только если константа "count" больше нуля. Следующий пример показывает более комплексную условную структуру:

```
if count & ~ count mod 4
    mov cx,count/4
    rep movsd
else if count>4
    mov cx,count/4
    rep movsd
    mov cx,count mod 4
    rep movsb
else
    mov cx,count
    rep movsb
end if
```

Первый блок инструкций ассеблируется, если константа "count" не равна нулю и кратна четырем, если это условие не выполняется, оценивается второе логическое условие, следующее за "else if", и если оно верно, ассемблируется второй блок инструкций, иначе ассемблируется последний блок, который следует за строкой, содержащей только "else".

"used" со следующим за ним символом имени, это логическое значение, которое проверяет, использовался ли где-нибудь данный символ (он возвращает правильный результат даже если символ используется только после этой проверки).

```
use16
org 100h
```

```
    mov ah, 9
    a = 0
    b = a + 5
    if used a
       mov dx, hello
       int 21h
    else
        mov dx, by
        int 21h
    end if
    mov ah, 8
    int 21h
    int 20h

hello db "Hello!$"
by db "By!$"
```

Данная программа выведет на экран “Hello!”, так как константа a использовалась при определении константы b. Если проверить использование константы b, то будет выведено “By!”, так как эта константа не использовалась, хотя и была определена.

За оператором "defined" может следовать любое выражение, обычно это только одно символьное имя; этот оператор проверяет, содержит ли данное выражение исключительно символы, определенные в коде, и доступные из текущей позиции.

```
use16
org 100h
    mov ah, 9
    a = 0
    b = a + 5
    if defined b
       mov dx, hello
       int 21h
    else
        mov dx, by
        int 21h
    end if
    mov ah, 8
    int 21h
    int 20h

hello db "Hello!$"
by db "By!$"
```

Данная программа выведет на экран “Hello!”, так как константа b была определена, хотя и не использовалась.

Также есть операторы, которые позволяют сравнивать значения, которые представляют собой последовательности символов. "eq" проверяет такие значения на тождественность. Оператор "in" проверяет, принадлежит ли данное значение к списку значений, следующему за оператором. Список должен быть заключен между символами "<" и ">", а его члены должны быть разделены запятыми. Символы считаются одинаковыми, если они имеют одно и то же значение для ассемблера - например, "pword" и "fword" для ассемблера одинаковы поэтому не различаются вышеуказанными операторами. Так же "16 eq 10h" является истиной, однако "16 eq 10+4" нет.

В приведенном ниже примере, на экран будет выведено YES, так как последнее значение из набора равно искомому (16=10h)

```
use16
org 100h
    mov ah, 9
    if 10h in <16h,18,10+6,16>
       mov dx, yes
       int 21h
    end if
```

```
    mov ah, 8
    int 21h
    int 20h
yes db "YES$"
```

2.2 Повторение блоков инструкций

"**times**" повторяет одну инструкцию указанное количество раз. За ней должно следовать числовое выражение, определяющее количество повторений, и инструкция, которую нужно повторять (опционально для того, чтобы отделить число и инструкцию, можно использовать двоеточие).

Специальный символ "%", использующийся внутри инструкции, эквивалентен номеру текущего повтора. Например, "`times 5 db %`" определит пять байтов со значениями `1, 2, 3, 4, 5`. Поддерживается также рекурсивное использование директивы "`times`", например:

```
times 3 times % db %
```

определит шесть байтов со значениями 1, 1, 2, 1, 2, 3.

"**repeat**" повторяет целый блок инструкций. За ней должно следовать числовое выражение, определяющее количество повторений. Инструкции для повторения предполагаются на следующих строках, а заканчиваться блок должен директивой "`end repeat`", например:

```
repeat 8
    mov byte [bx],%
    inc bx
end repeat
```

Сгенерированный код сохраняет байты со значениями от одного до восьми в памяти, адресованной регистром BX.

Количество повторений может быть равным нулю, и в таком случае инструкции не будут ассемблироваться вовсе.

"**break**" позволяет остановить повторение раньше и продолжить ассемблирование с первой строки после "`end repeat`". В сочетании с директивой "if" она позволяет остановить повторение при выполнении некоторого особого условия, например:

```
s = x/2
repeat 100
    if x/s = s
        break
    end if
    s = (s+x/s)/2
end repeat
```

"**while**" повторяет блок инструкций, пока выполняется следующее за ней условие, определенное логическим выражением. Блок инструкций для повторения должен заканчиваться директивой "`end while`". Перед каждым повторением логическое выражение вычисляется и если его значение ложь, ассемблирование продолжается, начиная с первой строки после "`end while`". Также в этом случае символ "%" содержит номер текущего повторения. Директива "`break`" может быть использована для остановки этого типа цикла так же, как с директивой "`repeat`". Предыдущий пример может быть переписан с использованием "while" вместо "repeat" таким образом:

```
s = x/2
while x/s <> s
    s = (s+x/s)/2
```

```
    if % = 100
        break
    end if
end while
```

Блоки, определенные с использованием "`if`", "`repeat`" и "`while`" могут быть вложены в любом порядке, однако и закрыты в обратном. Директива "`break`" всегда останавливает обработку бока, который был начат последним либо директивой "`repeat`", либо "`while`".

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

Для решения заданий использовать константы и директивы управления. Результат поместить в регистры AX – BX (если требуется).

ВАРИАНТЫ

1. Даны положительные числа A и B (A > B). На отрезке длины A размещено максимально возможное количество отрезков длины B (без наложений). Не используя операции умножения и деления, найти длину незанятой части отрезка A.
2. Даны положительные числа A и B (A > B). На отрезке длины A размещено максимально возможное количество отрезков длины B (без наложений). Не используя операции умножения и деления, найти количество отрезков B, размещенных на отрезке A.
3. Даны целые положительные числа N и K. Используя только операции сложения и вычитания, найти частное от деления нацело N на K, а также остаток от этого деления.
4. Дано целое число N (> 0). Найти квадрат данного числа, используя для его вычисления следующую формулу: $N^2 = 1 + 3 + 5 + \cdots + (2N - 1)$.
5. Даны два целых числа A и B (A < B). Найти сумму квадратов всех целых чисел от A до B включительно.
6. Дано целое число N (> 0). Найти сумму $N^2 + (N + 1)^2 + (N + 2)^2 + \cdots + (2 \cdot N)^2$
7. Дано целое число N (> 0). Если оно является степенью числа 3, то вывести True, если не является — вывести False.
8. Дано целое число N (> 0). Найти наименьшее целое положительное число K, квадрат которого превосходит N.
9. Дано целое число N (> 0). Найти наибольшее целое число K, квадрат которого не превосходит N.
10. Дано целое число N (> 1). Найти наименьшее целое число K, при котором выполняется неравенство $3^K > N$.
11. Дано целое число N (> 1). Найти наибольшее целое число K, при котором выполняется неравенство $3^K < N$.
12. Дано целое число N (> 0). Используя операции деления нацело и взятия остатка от деления, найти количество и сумму его цифр.
13. Дано целое число N (> 0). Используя операции деления нацело и взятия остатка от деления, найти число, полученное при прочтении числа N справа налево.
14. Дано целое число N (> 0). С помощью операций деления нацело и взятия остатка от деления определить, имеется ли в записи числа N цифра «2». Если имеется, то вывести True, если нет — вывести False.
15. Дано целое число N (> 0). С помощью операций деления нацело и взятия остатка от деления определить, имеются ли в записи числа N нечетные цифры. Если имеются, то вывести True, если нет — вывести False.
16. Дано целое число N (> 1). Если оно является простым, то есть не имеет положительных делителей, кроме 1 и самого себя, то вывести True, иначе вывести False.
17. Дано целое число N (> 1). Если оно является недостаточным, то есть сумма положительных делителей, кроме самого себя, меньше N, то вывести True, иначе вывести False.
18. Дано целое число N (> 1). Если оно является избыточным, то есть сумма положительных делителей, кроме самого себя, больше N, то вывести True, иначе вывести False.

19. Дано целое число N (> 1). Если оно является почти совершенным (слегка недостаточным), то есть сумма собственных делителей которого меньше самого числа ровно на единицу, то вывести True, иначе вывести False.
20. Даны целые положительные числа A и B. Найти их наибольший общий делитель (НОД).
21. Дано целое число N (> 1). Если оно является совершенным, то вывести True, иначе вывести False.
22. Даны два натуральных числа A и B. Если они являются взаимно простыми, то вывести True, иначе вывести False.
23. Даны два натуральных числа A и B. Если они являются дружественными, то вывести True, иначе вывести False.
24. Дано целое число N (> 1). Последовательность чисел Фибоначчи F_K определяется следующим образом:
    $F_1 = 1$, $F_2 = 1$, $F_K = F_{K-2} + F_{K-1}$, $K = 3, 4, \ldots$.
    Проверить, является ли число N числом Фибоначчи. Если является, то вывести True, если нет — вывести False.
25. Дано целое число N (> 1). Последовательность чисел Фибоначчи F_K определяется следующим образом:
    $F_1 = 1$, $F_2 = 1$, $F_K = F_{K-2} + F_{K-1}$, $K = 3, 4, \ldots$.
    Дано целое число N (> 1). Найти первое число Фибоначчи, большее N.
26. Дано целое число N (> 1). Последовательность чисел Фибоначчи F_K определяется следующим образом:
    $F_1 = 1$, $F_2 = 1$, $F_K = F_{K-2} + F_{K-1}$, $K = 3, 4, \ldots$.
    Найти сумму первых N членов последовательности.