Практика 05: циклы и переходы

Для организации цикла предназначена команда **LOOP**. У этой команды один операнд — имя метки, на которую осуществляется переход. В качестве счётчика цикла используется регистр ECX. Команда LOOP выполняет декремент ECX, а затем проверяет его значение. Если содержимое ECX не равно нулю, то осуществляется переход на метку, иначе управление переходит к следующей после LOOP команде.
Содержимое ECX интерпретируется командой как число без знака. В ECX нужно помещать число, равное требуемому количеству повторений цикла. Основное ограничение связано с дальность перехода. Метка должна находиться в диапазоне -127…+128 байт от команды LOOP (если это не так, ASM сообщит об ошибке).

Пример 1:

печать алфавита в строку

Решение:

```
.data
    abcstr DB ‘A’,0
.code
…
    mov ecx,26
loop1:
    push ecx
    push STD_OUTPUT_HANDLE
    call [GetStdHandle]
    push 0
    push inputnumber
```

```
        push 1
        push hellostr
        push eax
        call [WriteConsole]
        inc byte[hellostr]
        pop ecx
        loop loop1
...
```

Иногда требуется организовать вложенный цикл, то есть цикл внутри другого цикла. В этом случае необходимо сохранить значение ECX перед началом вложенного цикла и восстановить после его завершения (перед командой LOOP внешнего цикла). Сохранить значение можно в другой регистр, во временную переменную или в стек.

Пример 2:

печать алфавита в две строки

Решение:

```
.data
        abcstr DB ‘A’,0
        newlinestr DB 10,13,0
.code
...
        mov ecx,2
loop1:
      push ecx
      mov ecx,13
loop2:
        push ecx
        push STD_OUTPUT_HANDLE
        call [GetStdHandle]
```

```
push 0
push inputnumber
push 1
push abcstr
push eax
call [WriteConsole]
inc byte[abcstr]
pop ecx
loop loop2
push STD_OUTPUT_HANDLE
call [GetStdHandle]
push 0
push inputnumber
push 2
push newlinestr
push eax
call [WriteConsole]
pop ecx
loop loop1
...
```

Задание (при необходимости):

предусмотреть в программе ввод с клавиатуры в какое количество строк выводить алфавит. Разумно ограничить вариантами 1,2,3,4.

Безусловный и условный переход.

В наборе инструкций intel x86 и — соответственно — в языке ассемблера существуют команды, позволяющие влиять на порядок исполнения команд в программе. Ранее были

рассмотрены два способа

- цикл с помощью команд JZ и JMP
- цикл с помощью команды LOOP

Второй цикл был подробно рассмотрен в начале этого занятия, а вот команды JZ и JMP не рассматривались. Восполним этот пробел.

Команда JMP - Безусловный переход . Формат команды следующий

JMP адрес

При исполнении команды JMP происходит переход по адресу в параметре **адрес** и выполнение программы продолжается с указанного адреса, то есть фактически команда изменяет регистр PC. В качестве адреса может выступать

1. метка в коде (прямой переход)
2. регистр (костенный переход)

Во втором случае регистр (или ячейка памяти) должна содержать адрес на который нужно перейти. Способ применяется в тех случаях, когда адрес перехода определяется в ходе исполнения приложения и не известен заранее.

Команда JZ - Условный переход . Формат команды и значения параметра аналогичны команде JMP. Переход на указанный адрес осуществляется, если результат предыдущей команды окончился нулём. Например, если предыдущая команда

SUB EAX,EBX

и в результате EAX = 0 , то осуществляется переход. 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 фактически одно и то же), для других нет.
Для наглядности приведем далее результаты команд для случая, когда в предыдущей команде было сравнение двух параметров **арг1** и **арг2** . Сначала приведем команды для случая, когда сравниваются два числа :

Условие	**Команда**	**Флаги**	**Примечание**
арг1 = арг2	JE	ZF=1	Для любых чисел
арг1 != арг2	JNE	ZF=0	
арг1 < арг2	JL/JNGE	SF!=OF	Для знакопеременных чисел (число со знаком)
арг1 <= арг2	JLE/JNG	SF!=OF или ZF=1	
арг1 > арг2	JG/JNLE	SF=OF и ZF=0	
арг1 >= арг2	JGE/JNL	SF=OF	
арг1 < арг2	JB/JNAE	CF=1	Для знакопостоянных чисел (число без
арг1 <= арг2	JBE/JNA	CF=1 или ZF=1	

арг1>арг2	JA/JNBE	CF=0 и ZF=0	знака)
арг1>=арг2	JAE/JNB	CF=0	

Далее приведем команды при использовании которых обычно ориентируются на значения флагов, а не на тип предыдущей операции.

Команда	**Флаги**	**Команда**	**Флаги**
JZ	ZF=1	JNZ	ZF=0
JS	SF=1	JNS	SF=0
JC	CF=1	JNC	CF=0
JO	OF=1	JNO	OF=0
JP	PF=1	JNP	PF=0

Разумеется если значения флагов у команд совпадают, то они взаимозаменяемые. Например, команды JE и JZ абсолютно заменяемые. Использование той, или иной команды обусловлено исключительно личными предпочтениями программиста и более ничем.
Отдельно стоят две команды, которые в качестве условия перехода рассматривают не значение флагов, а состояние регистра

1. JCXZ — переход, если CX=0
2. JECXZ — переход, если ECX=0

Эти команды выполняются достаточно долго, намного эффективнее заменить их конструкцией наподобие

TEST ECX,ECX
JZ метка

Задание:

ответить произойдет переход в следующем коде, или нет
MOV ECX,131072
JCXZ метка

Решение:
произойдет. Объяснить почему предлагается самостоятельно

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

cmp al,5
jl label1 ;числа со знаком

cmp al,5
jb label1 ;числа без знака

Команда **TEST** работает как логическое И, но также 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

Все команды условного перехода имеют такое же ограничение как и loop - адрес перехода не должен быть далее 128 команд, иначе ошибка.
Чтобы этого избежать можно применять следующий трюк: заменять условный переход на противоположный, а вместо условного использовать безусловный переход. Например вместо

jz label1

написать

jnz label2
jmp label1
label2:

Условные циклы.

Кроме команды loop и команд условных переходов, есть ещё команды условных циклов

- **loopz** (аналог loope) – переход к метке цикла осуществляется если ecx не ноль и если флаг ZF равен 1
- **loopnz** (аналог loopne) – переход к метке цикла осуществляется если ecx не ноль и если флаг ZF равен 0

Эти условные циклы удобны в тех алгоритмах, где цикл должен завершатся в одном из двух случаев

1. выполнено требуемое количество итераций
2. выполнено требуемое условие

Например, это поиск элемента в массиве. Попробуем в строке найти определенный символ, например букву „y”

```
.data
        textstr DB ‘Hello to everyone!’,0
        foundstr DB 'Found letter in string.',0
        notfoundstr DB 'Did not find letter in string.',0
.code
<..>
        mov edi,textstr
        dec edi
        mov ecx,18
loop1:
        inc edi
        cmp byte[edi],”y”
        loopne loop1
        test ecx,ecx
        jz notfound
        push STD_OUTPUT_HANDLE
        call [GetStdHandle]
        push 0
        push inputnumber
        push 23
        push foundstr
        push eax
        call [WriteConsole]
        jmp endfind
notfound:
        push STD_OUTPUT_HANDLE
        call [GetStdHandle]
        push 0
        push inputnumber
        push 31
        push notfoundstr
        push eax
        call [WriteConsole]
```

endfind:
<..>

Loopnz должно следовать сразу за cmp, иначе флаг обнулится и условный цикл не сработает. Можно проверить с разными буквами.

Задание:

вывести номер позиции найденого символа.

Решение:

добавить после строки ***jz notfound*** строки

```
mov eax,19
    sub eax,[number]
    mov [number],eax
    numtostr number,inputstr
```

и потом не забыть вывести inputstr

Задание:

проверить совпадают ли две стоки

Решение:

```
.data
text1str DB `String N1`,0
text2str DB `String N2`,0
…
.code
….
mov esi,text1str
mov edi,text2str
dec es
dec edi
mov ecx,10
loop1:
```

```
inc esi
inc edi
mov bl,byte[esi]
cmp byte[edi],bl
loope loop1
test ecx,ecx
….. ; ecx=0 -  строки совпали, иначе есть различие
```

Домашнее задание:

проверить есть ли в двух строках совпадающие символы.