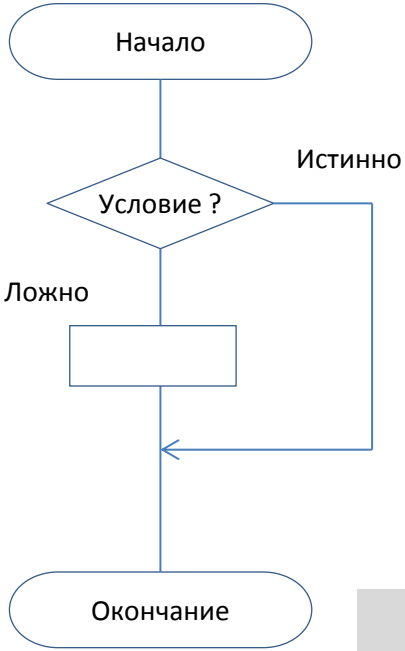
Машинно-зависимые языки программирования

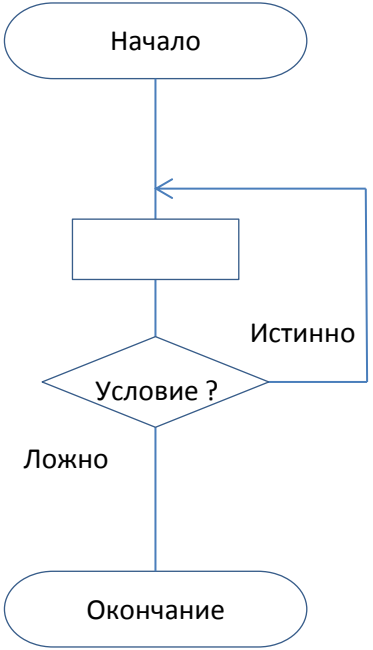
Савельев Игорь Леонидович

План

- Организация циклов (пред-, пост- условие, параметрический)

1. Организация цикла





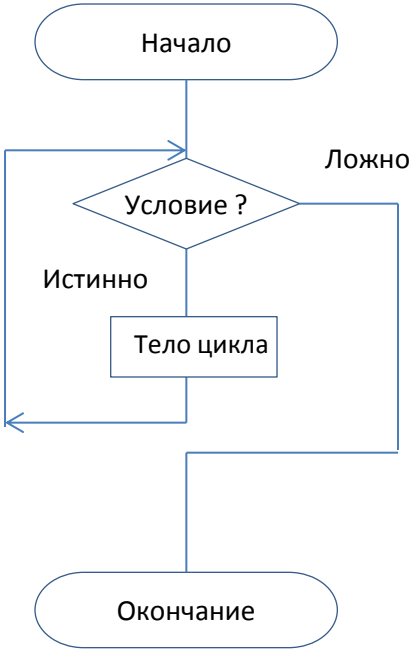

```
if(<условие>)
                {}
else
                {
                …..
                }
```

```
   mov eax, [a]
   cmp eax, 0;
   je @a_is_0
; …..
   jmp @end_if
@a_is_0:
@end_if:
   ret
```

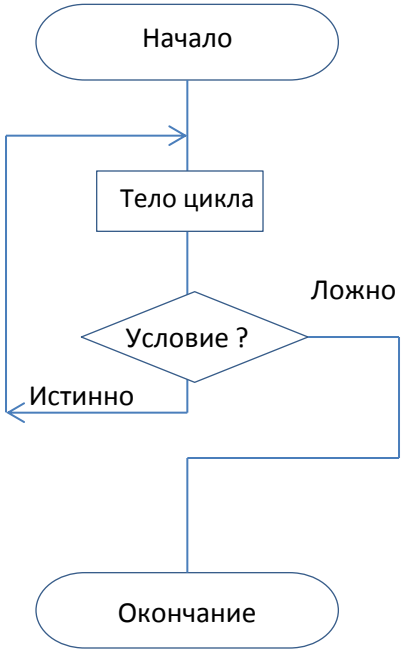
```
do {
                …..
} while (<условие>)
```

1. Организация цикла

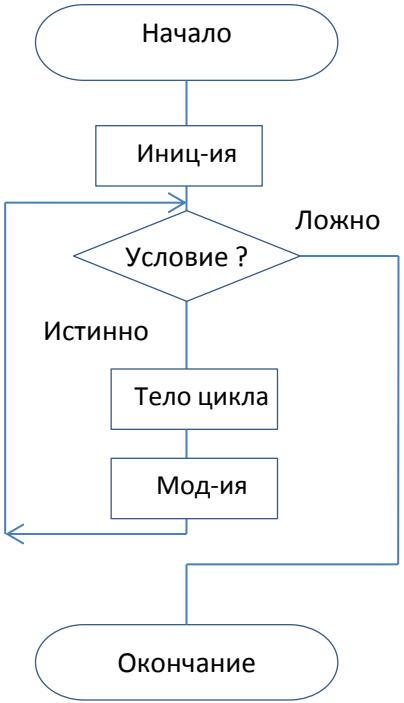
С предусловием



С постусловием



Параметрический



while, do-while, for, (if-else + goto)

1. Организация цикла

```
jmp $
```

```
@label
    .....
    jmp @label
```

```
@label:   jmp @label
```

1. Организация цикла

Организация цикла с помощью команд условного перехода и регистра ECX/CX .

Если некоторую группу команд необходимо повторить определенное количество раз, цикл можно организовать следующим образом:
1) поместить в регистр ecx/cx количество повторений;
2) первую команду тела цикла отметить меткой;
3) после выполнения тела цикла уменьшить содержимое регистра ecx/cx на 1;
4) выполнить сравнение содержимого регистра ecx/cxc нулем;
5) в случае, если ecx/cx≠ 0, осуществить переход на метку, иначе выполнять следующую за телом цикла команду.
Схема реализации будет выглядеть таким образом:

```
    mov cx, N
CYCL:
    <тело цикла>
    dec cx
    cmp cx,0
    jne CYCL
    ...
```

1. Организация цикла

Необходимо обращать внимание на выполнение «пустого» цикла. Если вдруг начальное значение в регистре CX равно нулю, то после уменьшения на единицу, содержимым регистра станет число **FFFFh** (-1 в десятичной системе счисления). В результате цикл повторится еще 65535 раз (или 4 294 967 295 раз при использовании регистра ecx). Поэтому следует всегда проверять содержимое регистра ecx/cx для предотвращения выполнения «пустого» цикла.

Такую проверку легко организовать с помощью команды условного перехода **jcxz/jecxz**

```
    mov cx,N
    jcxz Exit
CYCL:
    <тело цикла>
    dec cx
    cmp cx,0
    jne CYCL
Exit:  . . .
```

Замечание: Команда jcxz/jecxz может адресовать только короткие переходы (на –128 или на +127 байт от следующей за ней команды).

1. Организация цикла

Организация цикла с помощью команды безусловного перехода jmp и регистра ecx/cx.

Цикл можно организовать и следующим образом:
1) поместить в регистр ecx/cx количество повторений;
2) осуществить проверку на «пустой» цикл командой jecxz/jcxz <метка перехода>
3) эту команду проверки отметить меткой начала цикла;
4) после выполнения тела цикла уменьшить содержимое регистра ecx/cx на 1;
5) выполнить безусловный переход на начало цикла.
Схема реализации будет выглядеть таким образом:

```
      mov cx,N
  CYCL:
      jcxz Exit
      <тело цикла>
      dec cx
      jmp CYCL
  Exit:   . . .
```

Обратим внимание на то, что здесь команда jcxz играет двойную роль. Во-первых, предотвращает выполнение «пустого» цикла, и, во-вторых, отслеживает окончание цикла.

1. Организация цикла

Организация цикла с помощью специальных команд.

Заметим, что в описанных первых двух способах организации цикла большинство операций выполняются “вручную” (**декремент регистра cx**, **сравнение cx с нулем**, переход на начало цикла).

Учитывая важность циклов (практически ни одна программа не обходится без таких конструкций) разработчики системы команд микропроцессора ввели специальные команды, облегчающие (сокращающие) программирование циклических участков программ.

Это команды

loop <метка перехода>
loope/loopz <метка перехода>
loopne/loopnz <метка перехода>

Обратим внимание, эти команды также используют регистр ecx/cx как параметр (счетчик) цикла.

1. Организация цикла

Организация цикла с помощью команды loop

В переводе Loop означает «петля», или другими словами «повторить цикл».
Синтаксис команды:

loop <метка перехода>

Работа команды заключается в выполнении следующих действий:
1) уменьшение значения регистра **ecx/cx** на 1;
2) сравнение регистра **ecx/cx с** нулем:

если **ecx/cx = 0**, то осуществляется выход из цикла,
т.е. управление передается на следующую после loop команду;
иначе — управление передается на метку перехода (цикл повторяется).

1. Организация цикла

Порядок организации цикла с помощью команды loop :

1) в регистр cx поместить значение, равное количеству повторений;
2) установить метку на первую команду тела цикла;
3) выполнить команду loop <метка перехода>.

Схема реализации выглядит следующим образом:

```
        mov cx, N
        jcxz Exit
  CYCL :
        <тело цикла>
        loop CYCL
  Exit :     . . .
```

Обратим внимание также на то, что после выхода из цикла содержимое регистра **cx всегда равно нулю**..

!!! Проверяйте содержимое регистров

1. Организация цикла

Организация цикла с помощью команд loope / loopz.
Команды loope и loopz — абсолютные синонимы. Использовать можно любую из них.
Синтаксис команд:

loope <метка перехода>
loopz <метка перехода>

При использовании этих команд цикл повторяется до тех пор, пока **cx ≠ 0 и zf = 1**.
Выход из цикла осуществляется при выполнении одного из двух условий: **cx = 0 или zf = 0**.

Работа команд заключается в выполнении следующих действий:
1) уменьшение значения регистра ecx/cx на 1;
2) сравнение регистра ecx/cx с нулем: если (ecx/cx) = 0, осуществляется выход из цикла;
3) анализ состояния флага нуля zf: если zf = 0, осуществляется выход из цикла.
Таким образом, цикл повторяется, **если cx ≠ 0 и zf = 1**.

Поэтому, используя флаг ZF в качестве индикатора удобно организовать досрочный (до того, как значение cx станет 0) выход из цикла.

1. Организация цикла

Порядок организации цикла с помощью команды loope :

1) в регистр **cx** поместить значение, равное количеству повторений;
2) установить метку на первую команду тела цикла;
3) использовать команду **cmp** для сравнения операндов (здесь будет установлен **флаг** нуля);
4) выполнить команду **loope** <метка перехода>.

Схема реализации выглядит следующим образом:

```
        mov cx,N
        jcxz Exit
  CYCL :
        <тело цикла>
        cmp <оп1>,<оп2>
        loope CYCL
  Exit: . . .
```

Совет. Так как флаг нуля принимает значение 0 в случае несовпадения операндов и, соответственно, выход из цикла осуществляется именно при несовпадении операндов, команды loope/loopz удобно использовать для поиска **первого элемента** последовательности, **отличного** от заданной величины.

1. Организация цикла

Организация цикла с помощью команд loopne/loopnz.
Команды loopne/loopnz являются обратными командам loope/loopz.
loopne и **loopnz** также абсолютные синонимы, использовать можно любую из них.
Синтаксис команд:

loopne <метка перехода>
loopnz <метка перехода>

При использовании этих команд цикл повторяется до тех пор, пока **cx ≠ 0 и zf = 0**.
Выход из цикла осуществляется при выполнении одного из двух условий: **cx = 0 или zf = 1**.

Работа команд заключается в выполнении следующих действий:
1) уменьшение значения регистра ecx/cx на 1;
2) сравнение регистра ecx/cx с нулем: если (ecx/cx) = 0, осуществляется выход из цикла;
3) анализ состояния флага нуля zf : если zf = 1, осуществляется выход из цикла.
Таким образом, цикл повторяется, если cx ≠ 0 и zf = 0.

1. Организация цикла

Порядок организации цикла с помощью команды loopne :

1) в регистр **cx** поместить значение, равное количеству повторений;
2) установить метку на первую команду тела цикла;
3) использовать команду **cmp** для сравнения операндов (здесь будет установлен **флаг** нуля);
4) выполнить команду **loopne** <метка перехода>.

Схема реализации выглядит следующим образом:

```
        mov cx , N
        jcxz Exit
  CYCL :
        <тело цикла>
        cmp <оп1>,<оп2>
        loopne CYCL
  Exit:  . . .
```

Совет. Так как флаг нуля принимает значение 1 в случае совпадения операндов и, соответственно, выход из цикла осуществляется именно при совпадении операндов, команды loopne/loopnz удобно использовать для **поиска первого элемента последовательности**, **имеющего заданную величину.**

1. Организация цикла. Организация длинных циклов

Заметим, что специальные команды организации цикла loop , loope / loopz и loopne / loopnz , также как и команды условных переходов, реализуют только короткие переходы (от –128 до +127 байт), так как для адреса перехода в коде команд отводится только один байт. То есть тело цикла ограничивается только 128 байтами. Если тело цикла имеет большую длину, цикл необходимо организовывать другим образом.

Напомним, что только команда безусловного перехода **jmp** позволяет осуществлять длинные переходы, так как в коде команды под смещение адреса перехода отводится два байта.

Для организации длинных циклов следует использовать команды условного перехода и команду **jmp**.

Схема реализации длинного цикла:

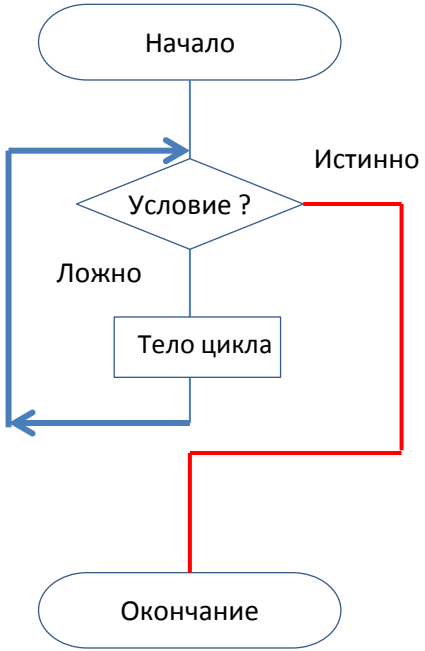
```
    mov cx, N
CYCL:
    <тело цикла>
    dec cx
    cmp cx,0
    je Out
    jmp CYCL
Out:  . . .
```

То есть, для организации длинного цикла необходимо «вручную» уменьшать содержимое параметра (счетчика) цикла и сравнивать полученное значение с нулем. Когда выполнено определенное количество повторений (cx = 0), с помощью команды условного перехода **je** следует выйти из цикла, иначе — командой безусловного перехода **jmp** необходимо осуществить переход на начало цикла.

1. Организация цикла. WHILE

Форма записи цикла while на языке C/C++:

```
while (/*условие продолжения цикла while*/)
{
        /*блок операторов*/;
}
```



Типичная структура такого цикла WHILE на языке ассемблера:

```
Cont:
        cmp X, Condition
        jxx Exit      ; условие ВЫХОДА!!
        ….. ;тело цикла
        jmp Cont
Exit: …
```

1. Организация цикла. WHILE

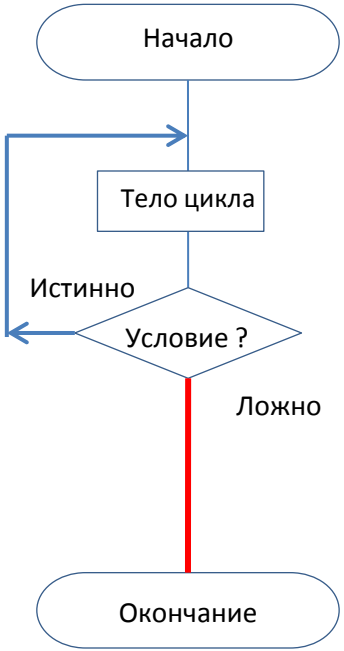
```
int i = 0;
while ( i <= 10) {
        i++;
}


        i dw 0
…
        mox ax, i;
Cont:
        cmp ax, 10          ; сравнение
        jg Exit             ; условие выхода!!
        inc ax              ;тело цикла
        jmp Cont
Exit: …
```

1. Организация цикла. DO WHILE

Форма записи цикла while на языке C/C++:

```
do
{
        /*блок операторов*/;
} while (/*условие продолжения цикла*/)
```



Типичная структура такого цикла DO WHILE на языке ассемблера:

```
Cont:
        …                   ;тело цикла
        CMP X, Condition
        Jxx Cont;           условие ПРОДОЛЖЕНИЯ ЦИКЛА
```

1. Организация цикла. DO WHILE

```
int i = 0;
do {
        i++;
} while ( i <= 10)


        i dw 0
…
        mox ax, i;
Cont:
        inc ax              ;тело цикла
        cmp ax, 10          ; сравнение
        jle Cont            ; условие продолжения!!
```

1. Организация цикла. FOR

Форма записи на языке C/C++:

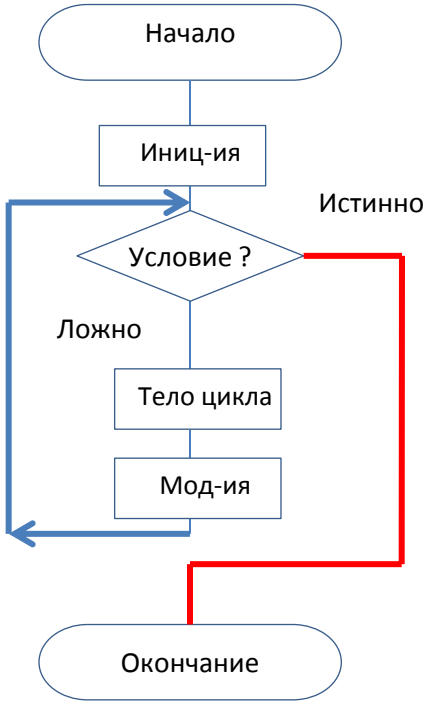
```
for (инициализация; условие; модификация) {
        тело цикла;
}
```

Типичная структура такого цикла на языке ассемблера:

```
        mov ax, N     ; Инициализация
Cont:
        cmp ax, Condition
        jxx Exit      ; условие ВЫХОДА!!
        ..... ;тело цикла
        inc ax        ; Модификация
        jmp Cont
Exit: …
```

Структура цикла с командой loop (число повторений N):

```
        mov cx, N     ; Инициализация
Cont:
        ..... ;тело цикла
        loop Cont
```



1. Организация цикла. Табулирование функции

Написать программу, которая сумму значений функции y = $x^2 + 4x - 5$ в диапазоне от 5 до 1 с шагом 1.

```
 x dw 0
 y dw 0

    mov ecx, 5       ; ecx = X
cycle:
    mov eax, ecx     ; eax = X
    imul ecx         ; edx:eax = ecx*eax = X^2
    mov ebx, ecx     ; ebx = X
    sal ebx, 2       ; ebx = X << 2 == 4*X
    add eax, ebx     ; eax = X^2 + 4X
    sub eax, 5       ; eax = X^2 + 4X - 5
    mov [y], eax     ; y = X^2 + 4X - 5
    loop cycle       ; ecx  = ecx - 1
```

где edx?

1. Организация цикла. Табулирование функции

Написать программу, которая сумму значений функции y = $x^2 + 4x - 5$ в диапазоне от 5 до 1 с шагом 1.

```
x dw 0
y dw 0

    mov ecx, 5      ; ecx = X
cycle:
    mov eax, ecx    ; eax = X
    imul ecx        ; edx:eax = ecx*eax = X^2
    mov ebx, ecx    ; ebx = X
    sal ebx, 2      ; ebx = X << 2 == 4*X
    add eax, ebx    ; eax = X^2 + 4X
    sub eax, 5      ; eax = X^2 + 4X - 5
    mov [y], eax    ; y = X^2 + 4X – 5
    dec ecx          ; ecx  = ecx - 1
    cmp ecx, 0
    jne cycle
```

```
x dw 0
y dw 0

    mov ecx, 5      ; ecx = X
cycle:
    mov eax, ecx    ; eax = X
    imul ecx        ; edx:eax = ecx*eax = X^2
    mov ebx, ecx    ; ebx = X
    sal ebx, 2      ; ebx = X << 2 == 4*X
    add eax, ebx    ; eax = X^2 + 4X
    sub eax, 5      ; eax = X^2 + 4X - 5
    mov [y], eax    ; y = X^2 + 4X – 5
    sub ecx, 1       ; ecx  = ecx - 1
    jne cycle
```

1. Организация цикла. Табулирование функции

Написать программу, которая сумму значений функции y = $x^2 + 4x - 5$ в диапазоне от 1 до 5 с шагом 1.

```
 x dw 0
 y dw 0

    mov ecx, 1       ; ecx = X
cycle:
     mov eax, ecx    ; eax = X
     imul ecx        ; edx:eax = ecx*eax = X^2
     mov ebx, ecx    ; ebx = X
     sal ebx, 2      ; ebx = X << 2 == 4*X
     add eax, ebx    ; eax = X^2 + 4X
     sub eax, 5      ; eax = X^2 + 4X - 5
     mov [y], eax    ; y = X^2 + 4X – 5
     inc ecx          ; ecx  = ecx + 1
     cmp ecx, 5
     jng cycle
```

1. Организация цикла. Число единичных битов в байте

```
section .data
   var db 0xAB              ; 10101011
   count db 0
section .text
global CMAIN
CMAIN:
   xor eax, eax                        ; al - счетчик
   xor ebx, ebx                        ; bl- для переменной
   xor ecx, ecx
   mov cx, 8                ; счетчик
   mov bl, [var]
next:
   shl bl, 1                ; CF – старший бит
   jnc @is_0
   ; is_1
   inc al                   ; count ++
   jmp cont
@is_0:
cont:
   loop next
   mov [count], al          ; Результат count=5
   ret
```

Спасибо