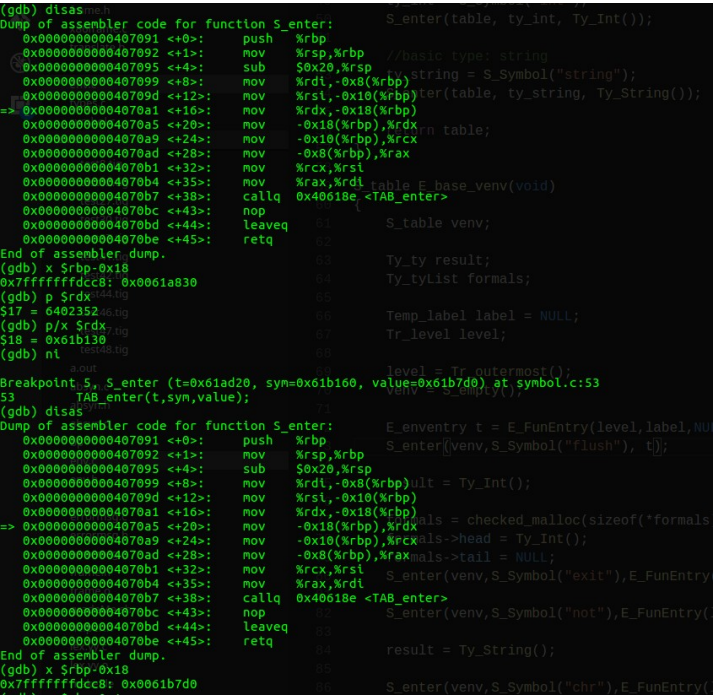
Ассемблер x86-64 (AMD64). Базовая техника



Используемые источники

Сайт программы: https://nasm.us/

Википедия: Ассемблер - https://ru.wikipedia.org/wiki/Ассемблер

Команды x86, x86-64: http://ccfit.nsu.ru/~kireev/lab2/lab2com.htm

Регистры x86, x86-64: http://ccfit.nsu.ru/~kireev/lab2/lab2reg.htm

Онлайн компиляторы, использующие NASM x86-64:

https://www.mycompiler.io/new/asm-x86_64

https://ideone.com/

Язык ассемблера для Intel x86 - Урок #4 - Целые числа, регистр EFLAGS, переполнение и арифметика: https://www.youtube.com/watch?v=dfv2VN6Xk0o

Learn programming through books and examples: https://riptutorial.com/

Лекционный материал по теме. Презентации, видео, примеры:
http://softcraft.ru/edu/comparch/lect/09-IntelAsmBase/

```
typedef struct rectangle {
    int x, y; // ширина, высота
} rectangle;

void InRndRectangle(rectangle *r) {
    r->x = Random();
    r->y = Random();
}

void OutRectangle(rectangle *r, FILE *ofst) {
    fprintf(ofst, "It is Rectangle: x = %d, y = %d. Perimeter = %g\n",
            r->x, r->y, PerimeterRectangle(r));
}

double PerimeterRectangle(rectangle *r) {
    return 2.0 * (r->x + r->y);
}
```

Язык C. Статически типизированное решение с использованием АТД

```
typedef enum key {RECTANGLE, TRIANGLE} key;
typedef struct shape {
    key k;
    union {
        rectangle r;
        triangle t;
    };
} shape;

bool InShape(shape *s, FILE *ifst) {
    int k;
    fscanf(ifst, "%d", &k);
    switch(k) {
        case 1:
            s->k = RECTANGLE;
            InRectangle(&(s->r), ifst);
            return true;
        case 2:
            s->k = TRIANGLE;
            InTriangle(&(s->t), ifst);
            return true;
        default:
            return false;
    }
}
```

```
void OutShape(shape *s, FILE *ofst) {
    switch(s->k) {
        case RECTANGLE:
            OutRectangle(&(s->r), ofst); break;
        case TRIANGLE:
            OutTriangle(&(s->t), ofst);  break;
        default:
            fprintf(ofst, "Incorrect figure!\n");
    }
}

double PerimeterShape(shape *s) {
    switch(s->k) {
        case RECTANGLE:
            return PerimeterRectangle(&(s->r));
            break;
        case TRIANGLE:
            return PerimeterTriangle(&(s->t));
            break;
        default:
            return 0.0;
    }
}
```

```c
typedef struct container {
    int len; // текущая длина
    shape cont[max_len];
} container;

void InContainer(container *c, FILE *ifst) {
    while(!feof(ifst)) {
        if(InShape(&((c->cont)[c->len]), ifst)) {
            c->len++;
        }
    }
}

void OutContainer(container *c, FILE *ofst) {
    fprintf(ofst, "Container contains %d elements.\n", c->len);
    for(int i = 0; i < c->len; i++) {
        fprintf(ofst, "%d: ", i);
        OutShape(&(c->cont)[i], ofst);
    }
}

double PerimeterSumContainer(container *c) {
    double sum = 0.0;
    for(int i = 0; i < c->len; i++) {
        sum += PerimeterShape(&(c->cont)[i]);
    }
    return sum;
}
```

```c
//-----------------------------------------------------------------------------
// data.h - Описание данных бестиповой программы
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// Раздел констант
//-----------------------------------------------------------------------------

// Константа, определяющая размер целого числа
int const intSize = sizeof(int);
// Константа, задающая размер для прямоугольника
int const rectSize = 2 * sizeof(int);
// Константа, задающая размер для треугольника
int const trianSize = 3 * sizeof(int);
// Константа, задающая размер для треугольника
int const shapeSize = sizeof(int) + (rectSize >= trianSize ? rectSize : trianSize);
// Константа, определяющая размерность массива фигур
int const maxSize = 10000 * shapeSize;
// Константа, задающая признак прямоугольника
int const RECTANGLE = 1;
// Константа, задающая признак треугольника
int const TRIANGLE = 2;
```

```c
int main(int argc, char* argv[]) {
    // Массив используемый для хранения данных
    //unsigned int cont[maxSize / intSize];
    //int cont[maxSize / intSize];
    unsigned char cont[maxSize];
    // Количество элементов в массиве
    int len = 0;
```

Язык С. Бестиповое решение

```c
// Ввод параметров прямоугольника из файла
void InRectangle(void *r, FILE *ifst) {
    fscanf(ifst, "%d%d", (int*)r, (int*)(r+intSize));
}

// Ввод параметров треугольника из файла
void InTriangle(void *t, FILE *ifst) {
    fscanf(ifst, "%d%d%d", (int*)t,
           (int*)(t+intSize), (int*)(t+2*intSize));
}

// Ввод параметров обобщенной фигуры из файла
int InShape(void *s, FILE *ifst) {
    int k;
    fscanf(ifst, "%d", &k);
    switch(k) {
        case 1:
            *((int*)s) = RECTANGLE;
            InRectangle(s+intSize, ifst);
            return 1;
        case 2:
            *((int*)s) = TRIANGLE;
            InTriangle(s+intSize, ifst);
            return 1;
        default:
            return 0;
    }
}
```

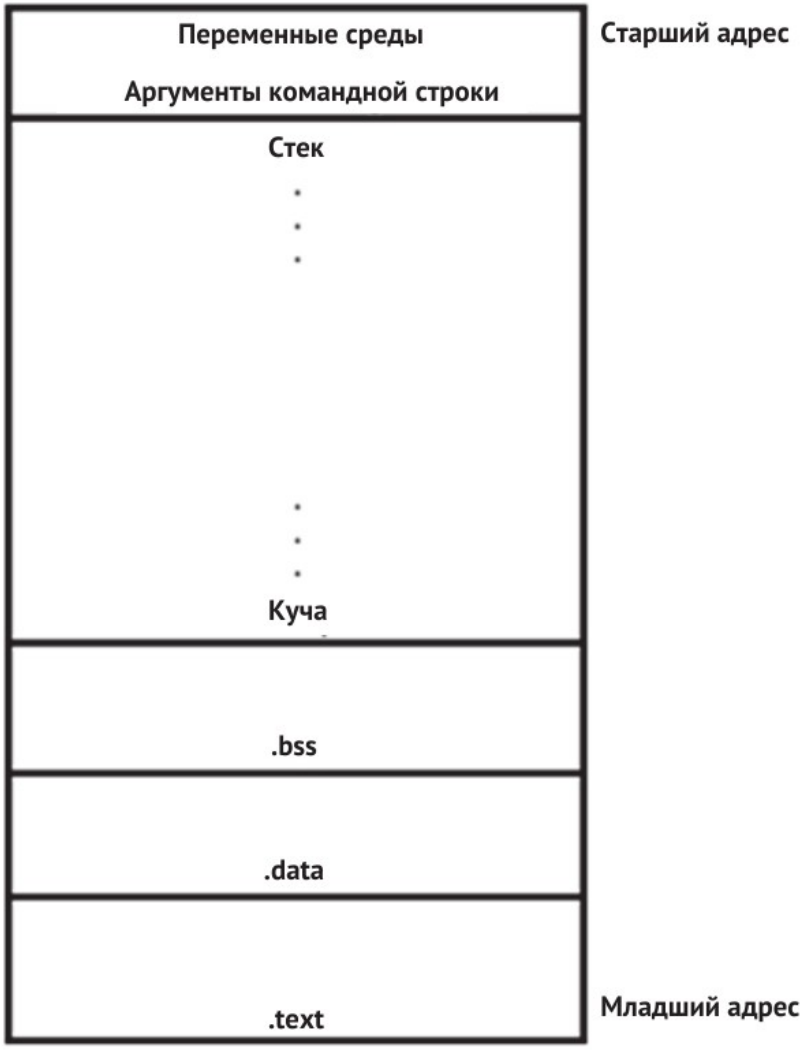
```c
// Ввод содержимого контейнера из указанного файла
void InContainer(void *c, int *len, FILE *ifst) {
    void *tmp = c;
    while(!feof(ifst)) {
        if(InShape(tmp, ifst)) {
            tmp = tmp + shapeSize;
            (*len)++;
        }
    }
}
```

Язык C. Бестиповое решение

```c
// Вычисление периметра прямоугольника
double PerimeterRectangle(void *r) {
    return 2.0 * (*((int*)r) + *((int*)(r+intSize)));
}

// Вычисление периметра треугольника
double PerimeterTriangle(void *t) {
    return (double)(*((int*)t) +
            *((int*)(t+intSize)) +
            *((int*)(t+2*intSize)));
}

// Вычисление периметра фигуры
double PerimeterShape(void *s) {
    int k = *((int*)s);
    if(k == RECTANGLE) {
        return PerimeterRectangle(s+intSize);
    }
    else if(k == TRIANGLE) {
        return PerimeterTriangle(s+intSize);
    }
    else {
        return 0.0;
    }
}
```

```c
// Вычисление суммы периметров всех фигур в контейнере
double PerimeterSumContainer(void *c, int len) {
    double sum = 0.0;
    void *tmp = c;
    for(int i = 0; i < len; i++) {
        sum += PerimeterShape(tmp);
        tmp = tmp + shapeSize;
    }
    return sum;
}
```

Организация памяти, выделяемой программе



Модуль тестирования программы вычисления периметров и суммы периметров

```asm
; main.asm
extern PerimeterRectangle
extern PerimeterTriangle
extern PerimeterShape
extern PerimeterSumContainer

global RECTANGLE
global TRIANGLE

extern  printf   ; declare the function as alien
section .data
    i       dd      10
    fmtstr  db      "%d -> %f",10,0     ; printf format
    pfmt    db      "Perimeter = %d, %f",10,0 ; printf format
    pcfmt   db      "Common Perimeter = %f",10,0 ; printf format
    RECTANGLE   dd  1
    TRIANGLE    dd  2
    shapeRect   dd  1
    rect    dd      5,7     ; прямоугольник с двумя сторонами
    empty   dd      0       ; пустое слово в фигуре для прямоугольника
    shapeTrian  dd  2
    trian   dd      3,4,5   ; три стороны треугольника
    ; Массив фигур
    cont    dd      1,2,3,0, 2,3,4,5, 1,2,8,0, 2,2,2,2, 2,5,7,4, 1,5,6,0
    len     dd      6
section .bss
    d       resq    1
    intP    resd    1       ; под целочисленное значение периметра
    p       resq    1       ; периметр как действительное
section .text
    global main
;---------------------------------------------
main:
    push    rbp
    mov     rbp,rsp

    mov     eax, [i]
    cvtsi2sd    xmm0, eax
    movsd [d], xmm0

    mov     rdi, fmtstr ; first argument for printf
    mov     rsi, [i]    ; second argument for printf
    movsd   xmm0, [d]   ; third argument for printf
    mov     rax, 1      ; xmm registers involved
    call    printf      ; call the function

    ; Вычисление периметра прямоугольника
    mov     rdi, rect
    call    PerimeterRectangle
    mov     [intP], eax
    movsd   [p], xmm0

    ; Вывод периметра прямоугольника
    mov     rdi, pfmt
    mov     rsi, rax
    movsd   xmm0, [p]   ; third argument for printf
    mov     rax, 1      ; only integer registers involved
    call    printf      ; call the function

    ; Вычисление периметра треугольника
    mov     rdi, trian
    call    PerimeterTriangle
    mov     [intP], eax
    movsd   [p], xmm0

    ; Вывод периметра треугольника
    mov     rdi, pfmt
    mov     rsi, rax
    movsd   xmm0, [p]   ; third argument for printf
    mov     rax, 1      ; only integer registers involved
    call    printf      ; call the function

    ; Вычисление периметра фигуры-прямоугольника
    mov     rdi, shapeRect
    call    PerimeterShape
    mov     [intP], eax
    movsd   [p], xmm0

    ; Вывод периметра фигуры-прямоугольника
    mov     rdi, pfmt
    mov     rsi, [intP]
    movsd   xmm0, [p]   ; third argument for printf
    mov     rax, 1      ; only integer registers involved
    call    printf      ; call the function

    ; Вычисление периметра фигуры-треугольника
    mov     rdi, shapeTrian
    call    PerimeterShape
    mov     [intP], eax
    movsd   [p], xmm0

    ; Вывод периметра фигуры-треугольника
    mov     rdi, pfmt
    mov     rsi, [intP]
    movsd   xmm0, [p]   ; third argument for printf
    mov     rax, 1      ; only integer registers involved
    call    printf      ; call the function

    ; Вычисление общего периметра 2-х фигур
    mov     rdi, shapeRect
    mov     rsi, 2
    call    PerimeterSumContainer
    movsd   [p], xmm0

    ; Вывод суммарного периметра 2-х фигур
    mov     rdi, pcfmt
    movsd   xmm0, [p]   ; third argument for printf
    mov     rax, 1      ; only integer registers involved
    call    printf      ; call the function

    ; Вычисление общего периметра нескольких фигур
    mov     rdi, cont
    mov     rsi, [len]
    call    PerimeterSumContainer
    movsd   [p], xmm0

    ; Вывод периметра нескольких фигур
    mov     rdi, pcfmt
    movsd   xmm0, [p]   ; third argument for printf
    mov     rax, 1      ; only integer registers involved
    call    printf      ; call the function

    pop     rbp
    mov     rax,60
    mov     rdi,0
syscall
```

Модуль вычисления периметров фигур и суммы периметров

```
;-----------------------------------------------------------------------------
; perimeter.asm - единица компиляции, вбирающая функции вычисления периметра
;-----------------------------------------------------------------------------

extern RECTANGLE
extern TRIANGLE

;-----------------------------------------------
; Вычисление периметра прямоугольника
;double PerimeterRectangle(void *r) {
;    return 2.0 * (*((int*)r)
;           + *((int*)(r+intSize)));
;}
global PerimeterRectangle
PerimeterRectangle:
section .text
push rbp
mov rbp, rsp

    ; В rdi адрес прямоугольника
    mov eax, [rdi]
    add eax, [rdi+4]
    shl eax, 1
    cvtsi2sd    xmm0, eax

leave
ret

;-----------------------------------------------
; double PerimeterTriangle(void *t) {
;    return (double)(*((int*)t)
;       + *((int*)(t+intSize))
;       + *((int*)(t+2*intSize)));
;}
global PerimeterTriangle
PerimeterTriangle:
section .text
push rbp
mov rbp, rsp

    ; В rdi адрес треугольника
    mov eax, [rdi]
    add eax, [rdi+4]
    add eax, [rdi+8]
    cvtsi2sd    xmm0, eax

leave
ret
```

```
;-----------------------------------------------
; Вычисление периметра фигуры
;double PerimeterShape(void *s) {
;    int k = *((int*)s);
;    if(k == RECTANGLE) {
;        return PerimeterRectangle(s+intSize);
;    }
;    else if(k == TRIANGLE) {
;        return PerimeterTriangle(s+intSize);
;    }
;    else {
;        return 0.0;
;    }
;}
global PerimeterShape
PerimeterShape:
section .text
push rbp
mov rbp, rsp

    ; В rdi адрес фигуры
    mov eax, [rdi]
    cmp eax, [RECTANGLE]
    je rectPerimeter
    cmp eax, [TRIANGLE]
    je trianPerimeter
    xor eax, eax
    cvtsi2sd    xmm0, eax
    jmp     return
rectPerimeter:
    ; Вычисление периметра прямоугольника
    add     rdi, 4
    call    PerimeterRectangle
    jmp     return
trianPerimeter:
    ; Вычисление периметра треугольника
    add     rdi, 4
    call    PerimeterTriangle
return:
leave
ret
```

```
;-----------------------------------------------
;// Вычисление суммы периметров всех фигур в контейнере
;double PerimeterSumContainer(void *c, int len) {
;    double sum = 0.0;
;    void *tmp = c;
;    for(int i = 0; i < len; i++) {
;        sum += PerimeterShape(tmp);
;        tmp = tmp + shapeSize;
;    }
;    return sum;
;}
global PerimeterSumContainer
PerimeterSumContainer:
section .data
    .sum    dq  0.0
section .text
push rbp
mov rbp, rsp

    ; В rdi адрес начала контейнера
    mov rbx, rsi            ; число фигур
    xor rcx, rcx            ; счетчик фигур
    movsd xmm1, [.sum]      ; перенос накопителя суммы в регистр 1
.loop:
    cmp rcx, rbx            ; проверка на окончание цикла
    jge .return             ; Перебрали все фигуры

    mov r10, rdi            ; сохранение начала фигуры
    call PerimeterShape     ; Получение периметра первой фигуры
    addsd xmm1, xmm0        ; накопление суммы
    inc rcx                 ; индекс следующей фигуры
    add r10, 16             ; адрес следующей фигуры
    mov rdi, r10            ; восстановление для передачи параметра
    jmp .loop
.return:
    movsd xmm0, xmm1
leave
ret
```

NASM. Главный модуль. Секция данных.

```
; main.asm
extern PerimeterRectangle
extern PerimeterTriangle
extern PerimeterShape
extern PerimeterSumContainer

global RECTANGLE
global TRIANGLE

extern printf   ; declare the function as alien
section .data
    i       dd      10
    fmtstr  db      "%d -> %f",10,0     ; printf format
    pfmt    db      "Perimeter = %d, %f",10,0 ; printf format
    pcfmt   db      "Common Perimeter = %f",10,0 ; printf format
    RECTANGLE   dd  1
    TRIANGLE    dd  2
    shapeRect   dd  1
    rect    dd      5,7     ; прямоугольник с двумя сторонами
    empty   dd      0       ; пустое слово в фигуре для прямоугольника
    shapeTrian  dd  2
    trian   dd      3,4,5   ; три стороны треугольника
    ; Массив фигур
    cont    dd      1,2,3,0, 2,3,4,5, 1,2,8,0, 2,2,2,2, 2,5,7,4, 1,5,6,0
    len     dd      6
section .bss
    d       resq    1
    intP    resd    1       ; под целочисленное значение периметра
    p       resq    1       ; периметр как действительное
```

NASM. Главный модуль. Фрагмент проверки команд

```
section .text
    global main
;-----------------------------------------------
main:
    push    rbp
    mov     rbp,rsp

    mov     eax, [i]
    cvtsi2sd    xmm0, eax
    movsd [d], xmm0

    mov     rdi, fmtstr ; first argument for printf
    mov     rsi, [i]    ; second argument for printf
    movsd   xmm0, [d]   ; third argument for printf
    mov     rax, 1      ; xmm registers involved
    call    printf      ; call the function
```

Соглашения о вызове внешних функций

Целочисленные аргументы:

- 1-й аргумент передается в регистре **rdi**;
- 2-й аргумент передается в регистре **rsi**;
- 3-й аргумент передается в регистре **rdx**;
- 4-й аргумент передается в регистре **rcx**;
- 5-й аргумент передается в регистре **r8**;
- 6-й аргумент передается в регистре **r9**.

Дополнительные аргументы передаются через стек в обратном порядке, чтобы можно было извлечь их в правильном порядке.

Аргументы с плавающей точкой:

- 1-й аргумент записывается в регистр **xmm0**;
- 2-й аргумент записывается в регистр **xmm1**;
- 3-й аргумент записывается в регистр **xmm2**;
- 4-й аргумент записывается в регистр **xmm3**;
- 5-й аргумент записывается в регистр **xmm4**;
- 6-й аргумент записывается в регистр **xmm5**;
- 7-й аргумент записывается в регистр **xmm6**;
- 8-й аргумент записывается в регистр **xmm7**.

NASM. Разработка фрагмента кода вычисления периметра прямоугольника

```
;-----------------------------------------------------------------------------
; perimeter.asm - единица компиляции, вбирающая функции вычисления периметра
;-----------------------------------------------------------------------------

extern RECTANGLE
extern TRIANGLE

;-----------------------------------------------
; Вычисление периметра прямоугольника
;double PerimeterRectangle(void *r) {
;    return 2.0 * (*((int*)r) + *((int*)(r+intSize)));
;}
global PerimeterRectangle
PerimeterRectangle:
section .text
push rbp
mov rbp, rsp

    ; В rdi адрес прямоугольника
    mov eax, [rdi]
    add eax, [rdi+4]
    shl eax, 1
    cvtsi2sd    xmm0, eax

leave
ret
```

NASM. Главный модуль. Фрагмент проверки прямоугольника

```
; Вычисление периметра прямоугольника
mov     rdi, rect
call    PerimeterRectangle
mov     [intP], eax
movsd   [p], xmm0

; Вывод периметра прямоугольника
mov     rdi, pfmt
mov     rsi, rax
movsd   xmm0, [p]   ; third argument for printf
mov     rax, 1      ; only integer registers involved
call    printf      ; call the function
```

```
rect    dd      5,7     ; прямоугольник с двумя сторонами
```

NASM. Разработка фрагмента кода вычисления периметра треугольника

```
;-----------------------------------------------
; double PerimeterTriangle(void *t) {
;    return (double)(*((int*)t) + *((int*)(t+intSize)) + *((int*)(t+2*intSize)));
;}
global PerimeterTriangle
PerimeterTriangle:
section .text
push rbp
mov rbp, rsp

    ; В rdi адрес треугольника
    mov eax, [rdi]
    add eax, [rdi+4]
    add eax, [rdi+8]
    cvtsi2sd    xmm0, eax

leave
ret
```

NASM. Главный модуль. Фрагмент проверки треугольника

```
; Вычисление периметра треугольника
mov     rdi, trian
call    PerimeterTriangle
mov     [intP], eax
movsd   [p], xmm0

; Вывод периметра треугольника
mov     rdi, pfmt
mov     rsi, rax
movsd   xmm0, [p]    ; third argument for printf
mov     rax, 1       ; only integer registers involved
call    printf       ; call the function
```

```
trian   dd      3,4,5   ; три стороны треугольника
```

NASM. Разработка фрагмента кода вычисления периметра фигуры

```
global PerimeterShape
PerimeterShape:
section .text
push rbp
mov rbp, rsp

    ; В rdi адрес фигуры
    mov eax, [rdi]
    cmp eax, [RECTANGLE]
    je rectPerimeter
    cmp eax, [TRIANGLE]
    je trianPerimeter
    xor eax, eax
    cvtsi2sd    xmm0, eax
    jmp     return
rectPerimeter:
    ; Вычисление периметра прямоугольника
    add     rdi, 4
    call    PerimeterRectangle
    jmp     return
trianPerimeter:
    ; Вычисление периметра треугольника
    add     rdi, 4
    call    PerimeterTriangle
return:
leave
ret
```

```
;-----------------------------------------
; Вычисление периметра фигуры
;double PerimeterShape(void *s) {
;    int k = *((int*)s);
;    if(k == RECTANGLE) {
;        return PerimeterRectangle(s+intSize);
;    }
;    else if(k == TRIANGLE) {
;        return PerimeterTriangle(s+intSize);
;    }
;    else {
;        return 0.0;
;    }
;}
```

Инструкции перехода и значения флагов

Инструкция	Флаги	Описание	Использование
je	ZF=1	Переход, если равно	Знаковые, беззнаковые
jne	ZF=0	Переход, если не равно	Знаковые, беззнаковые
jg	((SF XOR OF) OR ZF) = 0	Переход, если больше	Знаковые
jge	(SF XOR OF) = 0	Переход, если больше или равно	Знаковые
jl	(SF XOR OF) = 1	Переход, если меньше	Знаковые
jle	(SF XOR OF) OR ZF) = 1	Переход, если меньше или равно	Знаковые
ja	(CF OR ZF) = 0	Переход по условию «больше»	Беззнаковые
jae	CF=0	Переход по условию «больше или равно»	Беззнаковые
jb	CF=1	Переход по условию «меньше»	Беззнаковые
jbe	(CF OR ZF) = 1	Переход по условию «меньше или равно»	Беззнаковые

NASM. Главный модуль. Фрагмент проверки фигуры

```
; Вычисление периметра фигуры-прямоугольника
mov     rdi, shapeRect
call    PerimeterShape
mov     [intP], eax
movsd   [p], xmm0

; Вывод периметра фигуры-прямоугольника
mov     rdi, pfmt
mov     rsi, [intP]
movsd   xmm0, [p]   ; third argument for printf
mov     rax, 1      ; only integer registers involved
call    printf      ; call the function

; Вычисление периметра фигуры-треугольника
mov     rdi, shapeTrian
call    PerimeterShape
mov     [intP], eax
movsd   [p], xmm0

; Вывод периметра фигуры-треугольника
mov     rdi, pfmt
mov     rsi, [intP]
movsd   xmm0, [p]   ; third argument for printf
mov     rax, 1      ; only integer registers involved
call    printf      ; call the function
```

```
shapeRect   dd  1
rect        dd      5,7     ; прямоугольник с двумя сторонами
empty       dd      0       ; пустое слово в фигуре для прямоугольника
shapeTrian  dd  2
trian       dd      3,4,5   ; три стороны треугольника
```

NASM. Разработка фрагмента кода вычисления периметра

```
global PerimeterSumContainer
PerimeterSumContainer:
section .data
    .sum    dq  0.0
section .text
push rbp
mov rbp, rsp

    ; В rdi адрес начала контейнера
    mov rbx, rsi            ; число фигур
    xor rcx, rcx            ; счетчик фигур
    movsd xmm1, [.sum]      ; перенос накопителя суммы в регистр 1
.loop:
    cmp rcx, rbx            ; проверка на окончание цикла
    jge .return             ; Перебрали все фигуры

    mov r10, rdi            ; сохранение начала фигуры
    call PerimeterShape     ; Получение периметра первой фигуры
    addsd xmm1, xmm0        ; накопление суммы
    inc rcx                 ; индекс следующей фигуры
    add r10, 16             ; адрес следующей фигуры
    mov rdi, r10            ; восстановление для передачи параметра
    jmp .loop
.return:
    movsd xmm0, xmm1
leave
ret
```

```
;-----------------------------------------------
;// Вычисление суммы периметров всех фигур в контейнере
;double PerimeterSumContainer(void *c, int len) {
;    double sum = 0.0;
;    void *tmp = c;
;    for(int i = 0; i < len; i++) {
;        sum += PerimeterShape(tmp);
;        tmp = tmp + shapeSize;
;    }
;    return sum;
;}
```

NASM. Главный модуль. Фрагмент проверки 2-х фигур

```
; Вычисление общего периметра 2-х фигур
mov     rdi, shapeRect
mov     rsi, 2
call    PerimeterSumContainer
movsd   [p], xmm0

; Вывод суммарного периметра 2-х фигур
mov     rdi, pcfmt
movsd   xmm0, [p]   ; third argument for printf
mov     rax, 1      ; only integer registers involved
call    printf      ; call the function
```

```
shapeRect   dd  1
rect    dd      5,7     ; прямоугольник с двумя сторонами
empty   dd      0       ; пустое слово в фигуре для прямоугольника
shapeTrian  dd  2
trian   dd      3,4,5   ; три стороны треугольника
```

NASM. Главный модуль. Фрагмент проверки нескольких фигур

```
    ; Вывод периметра нескольких фигур
    mov     rdi, pcfmt
    movsd   xmm0, [p]   ; third argument for printf
    mov     rax, 1      ; only integer registers involved
    call    printf      ; call the function

    pop     rbp
    mov     rax,60
    mov     rdi,0
syscall
```

```
        ; Массив фигур
        cont    dd      1,2,3,0, 2,3,4,5, 1,2,8,0, 2,2,2,2, 2,5,7,4, 1,5,6,0
        len     dd      6
```

Makefile для сборки многофайлового проекта

```
# makefile for perimeter.asm
task: perimeter.o main.o
»       gcc -o perimeter main.o perimeter.o -no-pie
main.o: main.asm
»       nasm -f elf64 -g -F dwarf main.asm -l main.lst
perimeter.o: perimeter.asm
»       nasm -f elf64 -g -F dwarf perimeter.asm -l perimeter.lst
```

C+NASM. Совместное использование

```
;-----------------------------------------------------------------------------
; perimeter.asm - единица компиляции, вбирающая функции вычисления периметра
;-----------------------------------------------------------------------------

extern RECTANGLE
extern TRIANGLE

;-----------------------------------------------
; Вычисление периметра прямоугольника
;double PerimeterRectangle(void *r) {
;    return 2.0 * (*((int*)r) + *((int*)(r+intSize)));
;}
global PerimeterRectangle
PerimeterRectangle:
section .text
push rbp
mov rbp, rsp

    ; В rdi адрес прямоугольника
    mov eax, [rdi]
    add eax, [rdi+4]
    shl eax, 1
    cvtsi2sd    xmm0, eax

leave
ret

;-----------------------------------------------
; double PerimeterTriangle(void *t) {
;    return (double)(*((int*)t) + *((int*)(t+intSize)) + *((int*)(t+2*intSize)));
;}
global PerimeterTriangle
PerimeterTriangle:
section .text
push rbp
mov rbp, rsp

    ; В rdi адрес треугольника
    mov eax, [rdi]
    add eax, [rdi+4]
    add eax, [rdi+8]
    cvtsi2sd    xmm0, eax

leave
ret
```

```
;-----------------------------------------------
; Вычисление периметра фигуры
;double PerimeterShape(void *s) {
;    int k = *((int*)s);
;    if(k == RECTANGLE) {
;        return PerimeterRectangle(s+intSize);
;    }
;    else if(k == TRIANGLE) {
;        return PerimeterTriangle(s+intSize);
;    }
;    else {
;        return 0.0;
;    }
;}
global PerimeterShape
PerimeterShape:
section .text
push rbp
mov rbp, rsp

    ; В rdi адрес фигуры
    mov eax, [rdi]
    cmp eax, [RECTANGLE]
    je rectPerimeter
    cmp eax, [TRIANGLE]
    je trianPerimeter
    xor eax, eax
    cvtsi2sd    xmm0, eax
    jmp     return
rectPerimeter:
    ; Вычисление периметра прямоугольника
    add     rdi, 4
    call    PerimeterRectangle
    jmp     return
trianPerimeter:
    ; Вычисление периметра треугольника
    add     rdi, 4
    call    PerimeterTriangle
return:
leave
ret
```

C+NASM. Совместное использование

```
;-----------------------------------------------
;// Вычисление суммы периметров всех фигур в контейнере
;double PerimeterSumContainer(void *c, int len) {
;    double sum = 0.0;
;    void *tmp = c;
;    for(int i = 0; i < len; i++) {
;        sum += PerimeterShape(tmp);
;        tmp = tmp + shapeSize;
;    }
;    return sum;
;}
global PerimeterSumContainer
PerimeterSumContainer:
section .data
    .sum    dq  0.0
section .text
push rbp
mov rbp, rsp

    ; В rdi адрес начала контейнера
    mov rbx, rsi            ; число фигур
    xor rcx, rcx            ; счетчик фигур
    movsd xmm1, [.sum]      ; перенос накопителя суммы в регистр 1
.loop:
    cmp rcx, rbx            ; проверка на окончание цикла
    jge .return             ; Перебрали все фигуры

    mov r10, rdi            ; сохранение начала фигуры
    call PerimeterShape     ; Получение периметра первой фигуры
    addsd xmm1, xmm0        ; накопление суммы
    inc rcx                 ; индекс следующей фигуры
    add r10, 16             ; адрес следующей фигуры
    mov rdi, r10            ; восстановление для передачи параметра
    jmp .loop
.return:
    movsd xmm0, xmm1
leave
ret
```

C+NASM. Makefile

```
# makefile for perimeter.asm
task: main.c input.c inrnd.c output.c perimeter.o
»       gcc -g -o task main.c input.c inrnd.c output.c perimeter.o -no-pie
perimeter.o: perimeter.asm
»       nasm -f elf64 -g -F dwarf perimeter.asm -l perimeter.lst
```

Выполнение программ

10000 элементов	C & АТД	C & void	C & NASM
1	0.000298	0.000306	0.00018
2	0.000294	0.000297	0.000184
3	0.000296	0.000296	0.000182

Добавление модуля вывода данных в поток

```
; output.asm - вывод данных в поток
extern printf
extern fprintf

extern PerimeterRectangle
extern PerimeterTriangle

extern RECTANGLE
extern TRIANGLE

;-----------------------------------------------
;// Вывод параметров прямоугольника в файл
;void OutRectangle(void *r, FILE *ofst) {
;    fprintf(ofst, "It is Rectangle: x = %d, y = %d. Perimeter = %g\n",
;            *((int*)r), *((int*)(r+intSize)), PerimeterRectangle(r));
;}
global OutRectangle
OutRectangle:
section .data
    .outfmt db "It is Rectangle: x = %d, y = %d. Perimeter = %g",10,0
    ;.outfmt db "It is Rectangle",10,0
section .bss
    .prect  resq  1
    .FILE   resq  1       ; временное хранение указателя на файл
    .p      resq  1       ; вычисленный периметр прямоугольника
section .text
push rbp
mov rbp, rsp

    ; Сохранени принятых аргументов
    mov     [.prect], rdi          ; сохраняется адрес прямоугольника
    mov     [.FILE], rsi           ; сохраняется указатель на файл

    ; Вычисление периметра прямоугольника (адрес уже в rdi)
    call    PerimeterRectangle
    movsd   [.p], xmm0             ; сохранение (может лишнее) периметра

    ; Вывод информации о прямоугольнике в файл
    mov     rdi, [.FILE]
    mov     rsi, .outfmt           ; Формат - 2-й аргумент
    mov     rax, [.prect]           ; адрес прямоугольника
    mov     edx, [rax]             ; x
    mov     ecx, [rax+4]           ; y
    movsd   xmm0, [.p]
    mov     rax, 1                 ; есть числа с плавающей точкой
    call    fprintf

leave
ret
```

```
;-----------------------------------------------
; // Вывод параметров треугольника в файл
; void OutTriangle(void *t, FILE *ofst) {
;     fprintf(ofst, "It is Triangle: a = %d, b = %d, c = %d. Perimeter = %g\n",
;             *((int*)t), *((int*)(t+intSize)), *((int*)(t+2*intSize)),
;              PerimeterTriangle(t));
; }
global OutTriangle
OutTriangle:
section .data
    .outfmt db "It is Triangle: a = %d, b = %d, c = %d. Perimeter = %g",10,0
section .bss
    .ptrian  resq  1
    .FILE   resq  1       ; временное хранение указателя на файл
    .p      resq  1       ; вычисленный периметр треугольника
section .text
push rbp
mov rbp, rsp

    ; Сохранени принятых аргументов
    mov     [.ptrian], rdi         ; сохраняется адрес треугольника
    mov     [.FILE], rsi           ; сохраняется указатель на файл

    ; Вычисление периметра треугольника (адрес уже в rdi)
    call    PerimeterTriangle
    movsd   [.p], xmm0             ; сохранение (может лишнее) периметра

    ; Вывод информации о треугольнике в файл
    mov     rdi, [.FILE]
    mov     rsi, .outfmt           ; Формат - 2-й аргумент
    mov     rax, [.ptrian]         ; адрес треугольника
    mov     edx, [rax]             ; x
    mov     ecx, [rax+4]           ; b
    mov      r8, [rax+8]           ; c
    movsd   xmm0, [.p]
    mov     rax, 1                 ; есть числа с плавающей точкой
    call    fprintf

leave
ret
```

Добавление модуля вывода данных в поток

```
92  ;-----------------------------------------------
93  ; // Вывод параметров текущей фигуры в файл
94  ; void OutShape(void *s, FILE *ofst) {
95  ;     int k = *((int*)s);
96  ;     if(k == RECTANGLE) {
97  ;         OutRectangle(s+intSize, ofst);
98  ;     }
99  ;     else if(k == TRIANGLE) {
100 ;         OutTriangle(s+intSize, ofst);
101 ;     }
102 ;     else {
103 ;         fprintf(ofst, "Incorrect figure!\n");
104 ;     }
105 ; }
106 global OutShape
107 OutShape:
108 section .data
109     .erShape db "Incorrect figure!",10,0
110 section .text
111 push rbp
112 mov rbp, rsp
113
114     ; В rdi адрес фигуры
115     mov eax, [rdi]
116     cmp eax, [RECTANGLE]
117     je .rectOut
118     cmp eax, [TRIANGLE]
119     je .trianOut
120     mov rdi, .erShape
121     mov rax, 0
122     call fprintf
123     jmp     .return
124 .rectOut:
125     ; Вывод прямоугольника
126     add     rdi, 4
127     call    OutRectangle
128     jmp     .return
129 .trianOut:
130     ; Вывод треугольника
131     add     rdi, 4
132     call    OutTriangle
133 .return:
134 leave
135 ret
136
```

```
137 ;-----------------------------------------------
138 ; // Вывод содержимого контейнера в файл
139 ; void OutContainer(void *c, int len, FILE *ofst) {
140 ;     void *tmp = c;
141 ;     fprintf(ofst, "Container contains %d elements.\n", len);
142 ;     for(int i = 0; i < len; i++) {
143 ;         fprintf(ofst, "%d: ", i);
144 ;         OutShape(tmp, ofst);
145 ;         tmp = tmp + shapeSize;
146 ;     }
147 ; }
148 global OutContainer
149 OutContainer:
150 section .text
151 push rbp
152 mov rbp, rsp
153
154     ; В rdi адрес начала контейнера
155     mov rbx, rsi            ; число фигур
156     xor rcx, rcx            ; счетчик фигур = 0
157     mov rsi, rdx            ; перенос указателя на файл
158 .loop:
159     cmp rcx, rbx            ; проверка на окончание цикла
160     jge .return             ; Перебрали все фигуры
161
162     push rbx
163     push rcx
164     push rdi
165     push rsi
166     call OutShape     ; Получение периметра первой фигуры
167     pop rsi
168     pop rdi
169     pop rcx
170     pop rbx
171     inc rcx                 ; индекс следующей фигуры
172     add rdi, 16             ; адрес следующей фигуры
173     jmp .loop
174 .return:
175 leave
176 ret
```

Добавление модуля ввода данных из потока

```
; input.asm - ввод данных из потока
extern printf
extern fscanf

extern RECTANGLE
extern TRIANGLE

;-----------------------------------------------
; // Ввод параметров прямоугольника из файла
; void InRectangle(void *r, FILE *ifst) {
;     fscanf(ifst, "%d%d", (int*)r, (int*)(r+intSize));
; }
global InRectangle
InRectangle:
section .data
    .infmt db "%d%d",0
section .bss
    .FILE   resq    1   ; временное хранение указателя на файл
    .prect  resq    1   ; адрес прямоугольника
section .text
push rbp
mov rbp, rsp

    ; Сохранение принятых аргументов
    mov     [.prect], rdi          ; сохраняется адрес прямоугольника
    mov     [.FILE], rsi           ; сохраняется указатель на файл

    ; Ввод прямоугольника из файла
    mov     rdi, [.FILE]
    mov     rsi, .infmt            ; Формат - 1-й аргумент
    mov     rdx, [.prect]          ; &x
    mov     rcx, [.prect]
    add     rcx, 4                 ; &y = &x + 4
    mov     rax, 0                 ; нет чисел с плавающей точкой
    call    fscanf

leave
ret
```

```
; // Ввод параметров треугольника из файла
; void InTriangle(void *t, FILE *ifst) {
;     fscanf(ifst, "%d%d%d", (int*)t,
;            (int*)(t+intSize), (int*)(t+2*intSize));
; }
global InTriangle
InTriangle:
section .data
    .infmt db "%d%d%d",0
section .bss
    .FILE   resq    1   ; временное хранение указателя на файл
    .trian  resq    1   ; адрес треугольника
section .text
push rbp
mov rbp, rsp

    ; Сохранение принятых аргументов
    mov     [.trian], rdi          ; сохраняется адрес треугольника
    mov     [.FILE], rsi           ; сохраняется указатель на файл

    ; Ввод треугольника из файла
    mov     rdi, [.FILE]
    mov     rsi, .infmt            ; Формат - 1-й аргумент
    mov     rdx, [.trian]          ; &a
    mov     rcx, [.trian]
    add     rcx, 4                 ; &b = &a + 4
    mov     r8, [.trian]
    add     r8, 8                  ; &c = &x + 8
    mov     rax, 0                 ; нет чисел с плавающей точкой
    call    fscanf

leave
ret
```

Добавление модуля ввода данных из потока

```
; // Ввод параметров обобщенной фигуры из файла
; int InShape(void *s, FILE *ifst) {
;     int k;
;     fscanf(ifst, "%d", &k);
;     switch(k) {
;         case 1:
;             *((int*)s) = RECTANGLE;
;             InRectangle(s+intSize, ifst);
;             return 1;
;         case 2:
;             *((int*)s) = TRIANGLE;
;             InTriangle(s+intSize, ifst);
;             return 1;
;         default:
;             return 0;
;     }
; }
global InShape
InShape:
section .data
    .tagFormat   db      "%d",0
    .tagOutFmt   db     "Tag is: %d",10,0
section .bss
    .FILE       resq    1   ; временное хранение указателя на файл
    .pshape     resq    1   ; адрес фигуры
    .shapeTag   resd    1   ; признак фигуры
section .text
push rbp
mov rbp, rsp

    ; Сохранение принятых аргументов
    mov     [.pshape], rdi          ; сохраняется адрес фигуры
    mov     [.FILE], rsi            ; сохраняется указатель на файл

    ; чтение признака фигуры и его обработка
    mov     rdi, [.FILE]
    mov     rsi, .tagFormat
    mov     rdx, [.pshape]      ; адрес начала фигуры (ее признак)
    xor     rax, rax            ; нет чисел с плавающей точкой
    call    fscanf

    mov rcx, [.pshape]          ; загрузка адреса начала фигуры
    mov eax, [rcx]              ; и получение прочитанного признака
    cmp eax, [RECTANGLE]
    je .rectIn
    cmp eax, [TRIANGLE]
    je .trianIn
    xor eax, eax    ; Некорректный признак - обнуление кода возврата
    jmp     .return
.rectIn:
    ; Ввод прямоугольника
    mov     rdi, [.pshape]
    add     rdi, 4
    mov     rsi, [.FILE]
    call    InRectangle
    mov     rax, 1  ; Код возврата - true
    jmp     .return
.trianIn:
    ; Ввод треугольника
    mov     rdi, [.pshape]
    add     rdi, 4
    mov     rsi, [.FILE]
    call    InTriangle
    mov     rax, 1  ; Код возврата - true
.return:

leave
ret
```

Добавление модуля ввода данных из потока

```
; // Ввод содержимого контейнера из указанного файла
; void InContainer(void *c, int *len, FILE *ifst) {
;     void *tmp = c;
;     while(!feof(ifst)) {
;         if(InShape(tmp, ifst)) {
;             tmp = tmp + shapeSize;
;             (*len)++;
;         }
;     }
; }
global InContainer
InContainer:
section .bss
    .pcont  resq    1   ; адрес контейнера
    .plen   resq    1   ; адрес для сохранения числа введенных элементов
    .FILE   resq    1   ; указатель на файл
section .text
push rbp
mov rbp, rsp

    mov [.pcont], rdi   ; сохраняется указатель на контейнер
    mov [.plen], rsi    ; сохраняется указатель на длину
    mov [.FILE], rdx    ; сохраняется указатель на файл
    ; В rdi адрес начала контейнера
    xor rbx, rbx        ; число фигур = 0
    mov rsi, rdx        ; перенос указателя на файл
.loop:
    ; сохранение рабочих регистров
    push rdi
    push rbx

    mov     rsi, [.FILE]
    mov     rax, 0      ; нет чисел с плавающей точкой
    call    InShape     ; ввод фигуры
    cmp rax, 0          ; проверка успешности ввода
    jle  .return        ; выход, если признак меньше или равен 0

    pop rbx
    inc rbx

    pop rdi
    add rdi, 16             ; адрес следующей фигуры

    jmp .loop
.return:
    mov rax, [.plen]    ; перенос указателя на длину
    mov [rax], ebx      ; занесение длины
leave
ret
```

Добавление модуля генерации случайных данных

```asm
; input.asm - ввод данных из потока
extern printf
extern fscanf

extern RECTANGLE
extern TRIANGLE

;-----------------------------------------------
; // Ввод параметров прямоугольника из файла
; void InRectangle(void *r, FILE *ifst) {
;     fscanf(ifst, "%d%d", (int*)r, (int*)(r+intSize));
; }
global InRectangle
InRectangle:
section .data
    .infmt db "%d%d",0
section .bss
    .FILE   resq    1   ; временное хранение указателя на файл
    .prect  resq    1   ; адрес прямоугольника
section .text
push rbp
mov rbp, rsp

    ; Сохранение принятых аргументов
    mov     [.prect], rdi          ; сохраняется адрес прямоугольника
    mov     [.FILE], rsi           ; сохраняется указатель на файл

    ; Ввод прямоугольника из файла
    mov     rdi, [.FILE]
    mov     rsi, .infmt            ; Формат - 1-й аргумент
    mov     rdx, [.prect]          ; &x
    mov     rcx, [.prect]
    add     rcx, 4                 ; &y = &x + 4
    mov     rax, 0                 ; нет чисел с плавающей точкой
    call    fscanf

leave
ret
```

```asm
; // Ввод параметров треугольника из файла
; void InTriangle(void *t, FILE *ifst) {
;     fscanf(ifst, "%d%d%d", (int*)t,
;            (int*)(t+intSize), (int*)(t+2*intSize));
; }
global InTriangle
InTriangle:
section .data
    .infmt db "%d%d%d",0
section .bss
    .FILE   resq    1   ; временное хранение указателя на файл
    .trian  resq    1   ; адрес треугольника
section .text
push rbp
mov rbp, rsp

    ; Сохранение принятых аргументов
    mov     [.trian], rdi          ; сохраняется адрес треугольника
    mov     [.FILE], rsi           ; сохраняется указатель на файл

    ; Ввод треугольника из файла
    mov     rdi, [.FILE]
    mov     rsi, .infmt            ; Формат - 1-й аргумент
    mov     rdx, [.trian]          ; &a
    mov     rcx, [.trian]
    add     rcx, 4                 ; &b = &a + 4
    mov     r8, [.trian]
    add     r8, 8                  ; &c = &x + 8
    mov     rax, 0                 ; нет чисел с плавающей точкой
    call    fscanf

leave
ret
```

Добавление модуля генерации случайных данных

```
; inrnd.asm - порождение случайных данных
extern printf
extern rand

extern RECTANGLE
extern TRIANGLE


;-----------------------------------------------
; // rnd.c - содержит генератор случайных чисел в диапазоне от 1 до 20
; int Random( ) {
;     return rand( ) % 20 + 1;
; }
global Random
Random:
section .data
    .i20     dq      20
    .rndNumFmt       db "Random number = %d",10,0
section .text
push rbp
mov rbp, rsp

    xor     rax, rax    ;
    call    rand        ; запуск генератора случайных чисел
    xor     rdx, rdx    ; обнуление перед делением
    idiv    qword[.i20]       ; (/%) -> остаток в rdx
    mov     rax, rdx
    inc     rax         ; должно сформироваться случайное число

leave
ret
```

```
;-----------------------------------------------
;// Случайный ввод параметров прямоугольника
;void InRndRectangle(void *r) {
    ;int x = Random( );
    ;*((int*)r) = x;
    ;int y = Random( );
    ;*((int*)(r+intSize)) = y;
;//     printf("    Rectangle %d %d\n", *((int*)r), *((int*)r+1));
;}
global InRndRectangle
InRndRectangle:
section .bss
    .prect  resq 1   ; адрес прямоугольника
section .text
push rbp
mov rbp, rsp

    ; В rdi адрес прямоугольника
    mov     [.prect], rdi
    ; Генерация сторон прямоугольника
    call    Random
    mov     rbx, [.prect]
    mov     [rbx], eax
    call    Random
    mov     rbx, [.prect]
    mov     [rbx+4], eax

leave
ret
```

Добавление модуля генерации случайных данных

```
;-----------------------------------------------
;// Случайный ввод параметров треугольника
;void InRndTriangle(void *t) {
    ;int a, b, c;
    ;a = *((int*)t) = Random();
    ;b = *((int*)(t+intSize)) = Random();
    ;do {
        ;c = *((int*)(t+2*intSize)) = Random();
    ;} while((c >= a + b) || (a >= c + b) || (b >= c + a));
;//     printf("    Triangle %d %d %d\n", *((int*)t), *((int*)t+1), *((int*)t+2));
;}
global InRndTriangle
InRndTriangle:
section .bss
    .ptrian  resq 1   ; адрес треугольника
section .text
push rbp
mov rbp, rsp

    ; В rdi адрес треугольника
    mov     [.ptrian], rdi
    ; Генерация сторон треугольника
    call    Random
    mov     rbx, [.ptrian]
    mov     [rbx], eax
    call    Random
    mov     rbx, [.ptrian]
    mov     [rbx+4], eax
.repeat:
    call    Random
    mov     rbx, [.ptrian]
    mov     [rbx+8], eax    ; c
    ; Проверка корректности сторон
    mov     ecx, [rbx]      ; a
    mov     edx, [rbx+4]      ; b
    mov     ebx, eax        ; c
    add     eax, ecx        ; c+a
    cmp     edx, eax        ; b - (c+a)
    jge     .repeat
    mov     eax, ecx
    add     eax, edx        ; a+b
    cmp     ebx, eax        ; c - (a+b)
    jge     .repeat
    mov     eax, edx
    add     eax, ebx        ; b+c
    cmp     ecx, eax        ; a - (b+c)
    jge     .repeat

leave
ret
```

```
;-----------------------------------------------
;// Случайный ввод обобщенной фигуры
;int InRndShape(void *s) {
    ;int k = rand() % 2 + 1;
    ;switch(k) {
        ;case 1:
            ;*((int*)s) = RECTANGLE;
            ;InRndRectangle(s+intSize);
            ;return 1;
        ;case 2:
            ;*((int*)s) = TRIANGLE;
            ;InRndTriangle(s+intSize);
            ;return 1;
        ;default:
            ;return 0;
    ;}
;}
global InRndShape
InRndShape:
section .data
    .rndNumFmt       db "Random number = %d",10,0
section .bss
    .pshape     resq    1   ; адрес фигуры
    .key        resd    1   ; ключ
section .text
push rbp
mov rbp, rsp

    ; В rdi адрес фигуры
    mov [.pshape], rdi

    ; Формирование признака фигуры
    xor     rax, rax    ;
    call    rand        ; запуск генератора случайных чисел
    and     eax, 1      ; очистка результата кроме младшего разряда (0 или 1)
    inc     eax         ; фомирование признака фигуры (1 или 2)

    mov     rdi, [.pshape]
    mov     [rdi], eax      ; запись ключа в фигуру
    cmp eax, [RECTANGLE]
    je .rectInrnd
    cmp eax, [TRIANGLE]
    je .trianInRnd
    xor eax, eax        ; код возврата = 0
    jmp     .return
.rectInrnd:
    ; Генерация прямоугольника
    add     rdi, 4
    call    InRndRectangle
    mov     eax, 1      ;код возврата = 1
    jmp     .return
.trianInRnd:
    ; Генерация треугольника
    add     rdi, 4
    call    InRndTriangle
    mov     eax, 1      ;код возврата = 1
.return:
leave
ret
```

Добавление модуля генерации случайных данных

```
;-----------------------------------------------
;// Случайный ввод содержимого контейнера
;void InRndContainer(void *c, int *len, int size) {
    ;void *tmp = c;
    ;while(*len < size) {
        ;if(InRndShape(tmp)) {
            ;tmp = tmp + shapeSize;
            ;(*len)++;
        ;}
    ;}
;}
global InRndContainer
InRndContainer:
section .bss
    .pcont  resq    1   ; адрес контейнера
    .plen   resq    1   ; адрес для сохранения числа введенных элементов
    .psize  resd    1   ; число порождаемых элементов
section .text
push rbp
mov rbp, rsp

    mov [.pcont], rdi   ; сохраняется указатель на контейнер
    mov [.plen], rsi    ; сохраняется указатель на длину
    mov [.psize], edx    ; сохраняется число порождаемых элементов
    ; В rdi адрес начала контейнера
    xor ebx, ebx        ; число фигур = 0
.loop:
    cmp ebx, edx
    jge     .return
    ; сохранение рабочих регистров
    push rdi
    push rbx
    push rdx

    call    InRndShape     ; ввод фигуры
    cmp rax, 0          ; проверка успешности ввода
    jle  .return        ; выход, если признак меньше или равен 0

    pop rdx
    pop rbx
    inc rbx

    pop rdi
    add rdi, 16             ; адрес следующей фигуры

    jmp .loop
.return:
    mov rax, [.plen]    ; перенос указателя на длину
    mov [rax], ebx      ; занесение длины
leave
ret
```

Реализация главной функции

```
;-----------------------------------------------------------------------------
; main.asm - содержит главную функцию,
; обеспечивающую простое тестирование
;-----------------------------------------------------------------------------
; main.asm

global  RECTANGLE
global  TRIANGLE

%include "macros.mac"

section .data
    RECTANGLE   dd  1
    TRIANGLE    dd  2
    oneDouble   dq  1.0
    erMsg1  db "Incorrect number of arguments = %d: ",10,0
    rndGen  db "-n",0
    fileGen  db "-f",0
    errMessage1 db  "incorrect command line!", 10,"  Waited:",10
                db  "     command -f infile outfile01 outfile02",10,"  Or:",10
                db  "     command -n number outfile01 outfile02",10,0
    errMessage2 db  "incorrect qualifier value!", 10,"  Waited:",10
                db  "     command -f infile outfile01 outfile02",10,"  Or:",10
                db  "     command -n number outfile01 outfile02",10,0
    len         dd  0           ; Количество элементов в массиве
section .bss
    argc        resd    1
    num         resd    1
    sum         resq    1
    start       resq    1       ; начало отсчета времени
    delta       resq    1       ; интервал отсчета времени
    startTime   resq    2       ; начало отсчета времени
    endTime     resq    2       ; конец отсчета времени
    deltaTime   resq    2       ; интервал отсчета времени
    ifst        resq    1       ; указатель на файл, открываемый файл для чтения фигур
    ofst1       resq    1       ; указатель на файл, открываемый файл для записи контейнера
    ofst2       resq    1       ; указатель на файл, открываемый файл для записи периметра
    cont        resb    160000  ; Массив используемый для хранения данных

```

Макроопределения

```
; macros.mac - файл с макроопределениями

extern  fopen
extern  fscanf
extern  fclose
extern  fprintf
extern  printf
extern  stdout
extern  time
extern  srand
extern  strcmp
extern  atoi
extern  clock

extern  InContainer
extern  InRndContainer
extern  OutContainer
extern  PerimeterSumContainer

;-----------------------------------------------------------------
; Вывод строки символов из буфера
%macro  PrintStrBuf 2
        mov rdi, %2
        mov rsi, %1
        xor rax, rax
        call fprintf
%endmacro

;-----------------------------------------------------------------
; Вывод строки, передаваемой непосредственно макросу
%macro  PrintStr    2
    section .data
        %%arg1  db  %1,0        ; first argument
    section .text               ; the printf arguments
        mov rdi, %2
        mov rsi, %%arg1
        mov rax, 0              ; no floating point
        call fprintf
%endmacro
```

Макроопределения

```
41  ;-----------------------------------------------------------------
42  ; Вывод строки, передаваемой непосредственно макросу
43  ; с переводом на следующую строку
44  %macro  PrintStrLn    2
45      section .data
46          %%arg1  db  %1,10,0     ; first argument
47      section .text               ; the printf arguments
48          mov rdi, %2
49          mov rsi, %%arg1
50          mov rax, 0              ; no floating point
51          call fprintf
52  %endmacro
53
54  ;-----------------------------------------------------------------
55  ; Вывод целого числа
56  %macro  PrintInt    2
57      section .data
58          %%arg1  db  "%d",0     ; first argument
59      section .text               ; the printf arguments
60          mov rdi, %2
61          mov rsi, %%arg1
62          mov rdx, %1
63          mov rax, 0              ; no floating point
64          call fprintf
65  %endmacro
66
67  ;-----------------------------------------------------------------
68  ; Вывод 64-разрядного беззнакового целого числа
69  %macro  PrintLLUns    2
70      section .data
71          %%arg1  db  "%llu",0     ; first argument
72      section .text               ; the printf arguments
73          mov rdi, %2
74          mov rsi, %%arg1
75          mov rdx, %1
76          mov rax, 0              ; no floating point
77          call fprintf
78  %endmacro
79
```

```
80  ;-----------------------------------------------------------------
81  ; Вывод действительного числа двойной точности
82  %macro  PrintDouble    2
83      section .data
84          %%arg1  db  "%g",0     ; first argument
85      section .text               ; the printf arguments
86          mov rdi, %2
87          mov rsi, %%arg1
88          movsd xmm0, %1
89          mov rax, 1              ; no floating point
90          call fprintf
91  %endmacro
92
93  ;-----------------------------------------------------------------
94  ; Вывод содержимого контейнера
95  %macro  PrintContainer    3
96      mov     rdi, %1
97      mov     esi, %2
98      mov     rdx, %3
99      mov     rax, 0              ; нет чисел с плавающей точкой
100     call    OutContainer
101 %endmacro
102
103 ;-----------------------------------------------------------------
104 ; Вычисление суммы периметров фигур в контейнере
105 %macro  ContainerSum    3
106     mov     rdi, %1
107     mov     rsi, %2
108     call    PerimeterSumContainer
109     movsd   %3, xmm0
110 %endmacro
```

Макроопределения

```
112 ;-----------------------------------------------------------------
113 ; Открытие файла для чтения, записи
114 %macro  FileOpen   3
115     section .data
116         %%rw  db  %2,0          ; признак доступа
117     section .text               ; the printf arguments
118         mov     rdi, %1         ; адрес строки открываемого файла
119         mov     rsi, %%rw       ; строка определяющая чтение-запись. Задается явно
120         ; Или аналог:
121         ;lea     rsi, [%%rw]    ; строка определяющая чтение-запись. Задается явно
122         mov     rax, 0          ; нет чисел с плавающей точкой
123         call    fopen
124         mov     [%3], rax
125 %endmacro
126
127 ;-----------------------------------------------------------------
128 ; Закрытие открытого файла
129 %macro  FileClose   1
130     mov     rdi, %1             ; передача указателя на закрываемый файл
131     mov     rax, 0              ; нет чисел с плавающей точкой
132     call    fclose
133 %endmacro
```

Реализация главной функции

```
40  section .text
41      global main
42  main:
43  push rbp
44  mov rbp,rsp
45
46      mov dword [argc], edi ;rdi contains number of arguments
47      mov r12, rdi ;rdi contains number of arguments
48      mov r13, rsi ;rsi contains the address to the array of arguments
49
50  .printArguments:
51      PrintStrLn "The command and arguments:", [stdout]
52      mov rbx, 0
53  .printLoop:
54      PrintStrBuf qword [r13+rbx*8], [stdout]
55      PrintStr    10, [stdout]
56      inc rbx
57      cmp rbx, r12
58      jl .printLoop
59
60      cmp r12, 5      ; проверка количества аргументов
61      je .next1
62      PrintStrBuf errMessage1, [stdout]
63      jmp .return
64  .next1:
65      PrintStrLn "Start", [stdout]
```

Реализация главной функции

```
66      ; Проверка второго аргумента
67      mov rdi, rndGen      ; признак для сравнения
68      mov rsi, [r13+8]     ; второй аргумент командной строки
69      mov rcx, 3           ; сравнение, включая нуль (обе строки завершились)
70      cld                  ; обнуление флага направления (инкремент)
71      repe cmpsb
72      ;;call strcmp
73      ;;cmp rax, 0          ; строки равны "-n"
74      je .next2
75      mov rdi, fileGen
76      mov rsi, [r13+8]     ; второй аргумент командной строки
77      ;;mov rcx, 3          ; сравнение, включая нуль (обе строки завершились)
78      ;;cld                 ; обнуление флага направления (инкремент)
79      ;;repe cmpsb
80      call strcmp
81      cmp rax, 0           ; строки равны "-f"
82      je .next3
83      PrintStrBuf errMessage2, [stdout]
84      jmp .return
```

Реализация главной функции

```
85  .next2:
86      ; Генерация случайных фигур
87      mov rdi, [r13+16]
88      call atoi
89      mov [num], eax
90      PrintInt [num], [stdout]
91      PrintStrLn "", [stdout]
92      mov eax, [num]
93      cmp eax, 1
94      jl .fall1
95      cmp eax, 10000
96      jg .fall1
97      ; Начальная установка генератора случайных чисел
98      xor     rdi, rdi
99      xor     rax, rax
100     call    time
101     mov     rdi, rax
102     xor     rax, rax
103     call    srand
104     ; Заполнение контейнера случайными фигурами
105     mov     rdi, cont   ; передача адреса контейнера
106     mov     rsi, len    ; передача адреса для длины
107     mov     edx, [num]  ; передача количества порождаемых фигур
108     call    InRndContainer
109     jmp .task2
110
```

Реализация главной функции

```
111  .next3:
112      ; Получение фигур из файла
113      FileOpen [r13+16], "r", ifst
114      ; Заполнение контейнера фигурами из файла
115      mov     rdi, cont           ; адрес контейнера
116      mov     rsi, len            ; адрес для установки числа элементов
117      mov     rdx, [ifst]         ; указатель на файл
118      xor     rax, rax
119      call    InContainer         ; ввод данных в контейнер
120      FileClose [ifst]
121
122  .task2:
123      ; Вывод содержимого контейнера
124      PrintStrLn "Filled container:", [stdout]
125      PrintContainer cont, [len], [stdout]
126
127      FileOpen [r13+24], "w", ofst1
128      PrintStrLn "Filled container:", [ofst1]
129      PrintContainer cont, [len], [ofst1]
130      FileClose [ofst1]
```

Реализация главной функции

```
132     ; Вычисление времени старта
133     mov rax, 228    ; 228 is system call for sys_clock_gettime
134     xor edi, edi    ; 0 for system clock (preferred over "mov rdi, 0")
135     lea rsi, [startTime]
136     syscall         ; [time] contains number of seconds
137                     ; [time + 8] contains number of nanoseconds
138
139     ContainerSum cont, [len], [sum]
140
141     ; Вычисление времени завершения
142     mov rax, 228    ; 228 is system call for sys_clock_gettime
143     xor edi, edi    ; 0 for system clock (preferred over "mov rdi, 0")
144     lea rsi, [endTime]
145     syscall         ; [time] contains number of seconds
146                     ; [time + 8] contains number of nanoseconds
147
148     ; Получение времени работы
149     mov rax, [endTime]
150     sub rax, [startTime]
151     mov rbx, [endTime+8]
152     mov rcx, [startTime+8]
153     cmp rbx, rcx
154     jge .subNanoOnly
155     ; иначе занимаем секунду
156     dec rax
157     add rbx, 1000000000
158 .subNanoOnly:
159     sub rbx, [startTime+8]
160     mov [deltaTime], rax
161     mov [deltaTime+8], rbx
```

Реализация главной функции

```
163     ; Вывод периметра нескольких фигур
164     PrintStr "Perimeter sum = ", [stdout]
165     PrintDouble [sum], [stdout]
166     PrintStr ". Calculaton time = ", [stdout]
167     PrintLLUns [deltaTime], [stdout]
168     PrintStr " sec, ", [stdout]
169     PrintLLUns [deltaTime+8], [stdout]
170     PrintStr " nsec", [stdout]
171     PrintStr 10, [stdout]
172
173     FileOpen [r13+32], "w", ofst2
174     PrintStr  "Perimeter sum = ", [ofst2]
175     PrintDouble [sum], [ofst2]
176     PrintStr ". Calculaton time = ", [ofst2]
177     PrintLLUns [deltaTime], [ofst2]
178     PrintStr " sec, ", [ofst2]
179     PrintLLUns [deltaTime+8], [ofst2]
180     PrintStr " nsec", [ofst2]
181     PrintStr 10, [ofst2]
182     FileClose [ofst2]
183
184     PrintStrLn "Stop", [stdout]
185     jmp .return
186 .fall1:
187     PrintStr "incorrect numer of figures = ", [stdout]
188     PrintInt [num], [stdout]
189     PrintStrLn ". Set 0 < number <= 10000", [stdout]
190 .return:
191 leave
192 ret
```

Makefile

```
1  # makefile for task.asm
2  task: main.o input.o inrnd.o output.o perimeter.o
3  »       gcc -g -o task main.o input.o inrnd.o output.o perimeter.o -no-pie
4  main.o: main.asm macros.mac
5  »       nasm -f elf64 -g -F dwarf main.asm -l main.lst
6  input.o: input.asm
7  »       nasm -f elf64 -g -F dwarf input.asm -l input.lst
8  inrnd.o: inrnd.asm
9  »       nasm -f elf64 -g -F dwarf inrnd.asm -l inrnd.lst
10 output.o: output.asm
11 »       nasm -f elf64 -g -F dwarf output.asm -l output.lst
12 perimeter.o: perimeter.asm
13 »       nasm -f elf64 -g -F dwarf perimeter.asm -l perimeter.lst
```

Пример выполнения

```
$ make
nasm -f elf64 -g -F dwarf main.asm -l main.lst
nasm -f elf64 -g -F dwarf input.asm -l input.lst
nasm -f elf64 -g -F dwarf inrnd.asm -l inrnd.lst
nasm -f elf64 -g -F dwarf output.asm -l output.lst
nasm -f elf64 -g -F dwarf perimeter.asm -l perimeter.lst
gcc -g -o task main.o input.o inrnd.o output.o perimeter.o -no-pie
 $   ./task -n 5 1 2
The command and arguments:
./task
-n
5
1
2
Start
5
Filled container:
0: It is Triangle: a = 10, b = 9, c = 17. Perimeter = 36
1: It is Triangle: a = 13, b = 12, c = 17. Perimeter = 42
2: It is Triangle: a = 14, b = 15, c = 2. Perimeter = 31
3: It is Triangle: a = 19, b = 18, c = 15. Perimeter = 52
4: It is Rectangle: x = 19, y = 9. Perimeter = 56
Perimeter sum = 217. Calculaton time = 0 sec, 1500 nsec
Stop
$ ./task -f tests/test01.txt 1 2
The command and arguments:
./task
-f
tests/test01.txt
1
2
Start
Filled container:
0: It is Rectangle: x = 3, y = 4. Perimeter = 14
1: It is Triangle: a = 1, b = 1, c = 1. Perimeter = 3
2: It is Rectangle: x = 30, y = 40. Perimeter = 140
3: It is Triangle: a = 2, b = 2, c = 1. Perimeter = 5
4: It is Rectangle: x = 13, y = 14. Perimeter = 54
5: It is Triangle: a = 10, b = 10, c = 10. Perimeter = 30
6: It is Rectangle: x = 330, y = 49. Perimeter = 758
7: It is Triangle: a = 3, b = 4, c = 5. Perimeter = 12
Perimeter sum = 1016. Calculaton time = 0 sec, 1857 nsec
Stop
[al@parsys 11-asm]$
```

Обработка строк символов. Вычисление длины строки

```
65  ; FUNCTIONS ============================================================
66
67  ;-----------------------------------------------------------------------
68  ; Вычисление длины строки с учетом конечного нулевого символа
69  strlen0:
70      ; Адрес сообщения уже загружен в rdi
71      mov ecx, -1      ; ecx должен быть < 0
72      xor al, al       ; конечный символ = 0
73      cld              ; направление обхода от начала к концу
74      repne   scasb    ; while(msg[rdi]) != al) {rdi++, rcx--}
75      neg ecx
76      sub ecx, 1       ; ecx = length(msg)
77      mov eax, ecx
78      ret
79
80  ;-----------------------------------------------------------------------
81  ; Вычисление длины строки как в языке C
82  strlen:
83      ; Адрес сообщения уже загружен в rdi
84      mov ecx, -1      ; ecx должен быть < 0
85      xor al, al       ; конечный симврл = 0
86      cld              ; направление обхода от начала к концу
87      repne   scasb    ; while(msg[rdi]) != al) {rdi++, rcx--}
88      neg ecx
89      sub ecx, 2       ; ecx = length(msg)
90      mov eax, ecx
91      ret
```

Обработка строк символов. Вычисление длины строки

```
; test-strlen.asm - вычисление длины строки
%include "macros.mac"

section .data
    NL          equ     10

    msg         db      "Hello World!", 10,0
    info        db      "Message: ",0
    emptyMsg    db      0
    fDecimal    db      "%d",10,0
    len         dd      0

section .bss
    strBuf      resb    256

section .text
    global main
;-------------------------------------------
main:
    push    rbp
    mov     rbp,rsp

    PrintStrBuf info, [stdout]
    PrintStrBuf msg, [stdout]

    mov     rdi, msg
    call    strlen0
    mov     [len], eax
    PrintStr    "msg length with zero symbol = ", [stdout]
    PrintInt    [len], [stdout]
    PrintStr    NL, [stdout]

    mov     rdi, msg
    call    strlen
    mov     [len], eax
    PrintStr    "msg length as C-strlen = ", [stdout]
    PrintInt    [len], [stdout]
    PrintStr    NL, [stdout]

    PrintStrBuf info, [stdout]
    PrintStrBuf emptyMsg, [stdout]
    PrintStr    NL, [stdout]

    mov     rdi, emptyMsg
    call    strlen0
    mov     [len], eax
    PrintStr    "emptyMsg length with zero symbol = ", [stdout]
    PrintInt    [len], [stdout]
    PrintStr    NL, [stdout]

    mov     rdi, emptyMsg
    call    strlen
    mov     [len], eax
    PrintStr    "emptyMsg length as C-strlen = ", [stdout]
    PrintInt    [len], [stdout]
    PrintStr    NL, [stdout]

    PrintStrLn "Bye!", [stdout]

    pop     rbp
    mov     rax,60
    mov     rdi,0
syscall

```

```
Message: Hello World!
msg length with zero symbol = 14
msg length as C-strlen = 13
Message:
emptyMsg length with zero symbol = 1
emptyMsg length as C-strlen = 0
Bye!
```

Обработка строк символов. Копирование строки

```
95  ;-----------------------------------------------------------------------
96  ; Копирование строки как в языке C
97  strcpy:
98      ; Адрес приемника уже загружен в rdi
99      ; Адрес источника уже загружен в rsi
100     cld             ; направление обхода от начала к концу
101     xor al, al      ; конечный симвло = 0
102     xor ecx, ecx    ; счетчик символов = 0
103 .loopstrcpy:
104     cmp al, [rsi]
105     je .endloopstrcpy
106     movsb           ; dst[] = src
107     inc ecx         ; ecx++
108     jmp .loopstrcpy
109 .endloopstrcpy:
110     mov [edi], al
111     mov eax, ecx
112     inc eax
113     ret
114
115 ;-----------------------------------------------------------------------
116 ; Копирование строки с использованием strlen0 и rep
117 strcpy2:
118     ; Адрес приемника уже загружен в rdi
119     mov r10, rdi    ; перенос во временное хранилище
120     ; Адрес источника уже загружен в rsi
121     mov r11, rsi    ; перенос во временное хранилище
122
123     mov rdi, rsi
124     call strlen0    ; вычисление длины строки с нулем
125
126     mov ecx, eax
127     mov rdi, r10
128     mov rsi, r11
129     cld             ; направление обхода от начала к концу
130     rep movsb
131     ret
```

Обработка строк символов. Копирование строки

```
; test-strcpy.asm - копирование строк
%include "macros.mac"

section .data
    NL          equ     10

    msg         db      "Hello World!", 10,0
    info        db      "Message: ",0
    emptyMsg    db      0
    fDecimal    db      "%d",10,0
    len         dd      0

section .bss
    strBuf      resb    256

section .text
    global main
;-----------------------------------------------
main:
    push    rbp
    mov     rbp,rsp

    PrintStrBuf info, [stdout]
    PrintStrBuf msg, [stdout]

    mov     rdi, msg
    call    strlen0
    mov     [len], eax
    PrintStr    "msg length with zero symbol = ", [stdout]
    PrintInt    [len], [stdout]
    PrintStr    NL, [stdout]

    mov     rdi, info
    call    strlen0
    mov     [len], eax
    PrintStr    "msg length with zero symbol = ", [stdout]
    PrintInt    [len], [stdout]
    PrintStr    NL, [stdout]

    mov     rdi, strBuf
    mov     rsi, msg
    call    strcpy
    mov     [len], eax
    PrintStr    "strcpy: strBuf = ", [stdout]
    PrintStrBuf strBuf, [stdout]
    PrintStr    "strBuf len = ", [stdout]
    PrintInt [len], [stdout]
    PrintStr    NL, [stdout]

    mov     rdi, strBuf
    mov     rsi, info
    call    strcpy2
    mov     [len], eax
    PrintStr    "strcpy2: strBuf = ", [stdout]
    PrintStrBuf strBuf, [stdout]
    PrintStr    NL, [stdout]
    PrintStr    "strBuf len = ", [stdout]
    PrintInt [len], [stdout]
    PrintStr    NL, [stdout]

    PrintStrLn "Bye!", [stdout]
    pop     rbp
    mov     rax,60
    mov     rdi,0
syscall
```

```
Message: Hello World!
msg length with zero symbol = 14
msg length with zero symbol = 10
strcpy: strBuf = Hello World!
strBuf len = 14
strcpy2: strBuf = Message:
strBuf len = 10
Bye!
```

Обработка строк символов. Проверка на равенство

```
135 ;-----------------------------------------------------------------------
136 ; Сравнение двух строк на равенство
137 streq:
138     ; Адрес приемника уже загружен в rdi
139     ; Адрес источника уже загружен в rsi
140     cld             ; направление обхода от начала к концу
141     xor eax, eax    ;
142 .loopStrEq:
143     cmp al, [rsi]
144     je .endSrc
145     cmp al, [rdi]
146     je .isNe
147     cmpsb
148     je .loopStrEq
149     jmp .isNe
150 .endSrc:
151     cmp al, [edi]
152     je .isEq
153     jmp .isNe
154 .isEq:
155     mov eax, 1      ; eax != 0 - true
156     ret
157 .isNe:
158     ;xor eax, eax
159     ret
```

Обработка строк символов. Проверка на равенство

```asm
; test-streq.asm - сравнение строк на равенство
%include "macros.mac"

section .data
    NL          equ     10

    msg         db      "Hello World!", 10,0
    info        db      "Message: ",0
    emptyMsg    db      0
    fDecimal    db      "%d",10,0
    len         dd      0
    flag        dd      0

section .bss
    strBuf      resb    256

section .text
    global main
;-------------------------------------------
main:
    push    rbp
    mov     rbp,rsp

    PrintStrBuf info, [stdout]
    PrintStr    NL, [stdout]
    PrintStrBuf msg, [stdout]

    mov     rdi, info
    mov     rsi, msg
    call    streq       ; != (false)
    mov     [flag], eax
    PrintStr    "Compare flag = ", [stdout]
    PrintInt    [flag], [stdout]
    PrintStr    NL, [stdout]

    mov     rdi, msg
    mov     rsi, msg
    call    streq       ; == (true)
    mov     [flag], eax
    PrintStr    "Compare flag = ", [stdout]
    PrintInt    [flag], [stdout]
    PrintStr    NL, [stdout]

    mov     rdi, strBuf
    mov     rsi, msg
    call strcpy
    mov     rdi, msg
    mov     rsi, strBuf
    call    streq       ; == (true)
    mov     [flag], eax
    PrintStr    "Compare flag = ", [stdout]
    PrintInt    [flag], [stdout]
    PrintStr    NL, [stdout]

    mov     rdi, info
    mov     rsi, strBuf
    call    streq       ; != (false)
    mov     [flag], eax
    PrintStr    "Compare flag = ", [stdout]
    PrintInt    [flag], [stdout]
    PrintStr    NL, [stdout]

    PrintStrLn "Bye!", [stdout]
    pop     rbp
    mov     rax,60
    mov     rdi,0
syscall
```

Обработка строк символов. Проверка на равенство

```
Message:
Hello World!
Compare flag = 0
Compare flag = 1
Compare flag = 1
Compare flag = 0
Bye!
```

Обработка строк символов. Аналог strcmp языка C

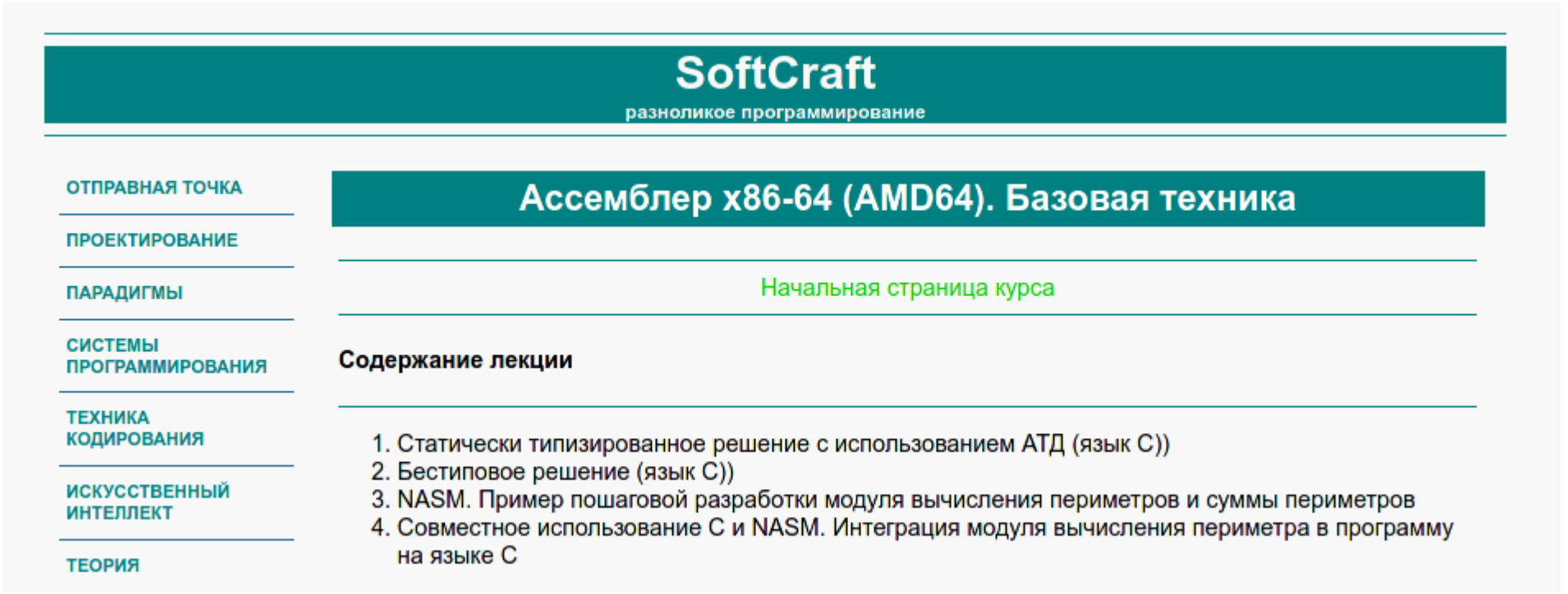
```
169 ;-----------------------------------------------------------------------
170 ; Сравнение двух строк по аналогии с strcmp языка C
171 strcmp:
172     ; Адрес приемника уже загружен в rdi
173     ; Адрес источника уже загружен в rsi
174     cld             ; направление обхода от начала к концу
175     xor eax, eax    ;
176 .loopStrEq:
177     cmp al, [rsi]
178     je .endSrc
179     cmp al, [rdi]
180     je .isGt
181     cmpsb
182     je .loopStrEq
183     jl .isLt
184     jmp .isGt
185 .endSrc:
186     cmp al, [edi]
187     je .isEq
188     jmp .isLt
189 .isEq:
190     mov eax, 0      ; [rsi] == [rdi]
191     ret
192 .isLt:
193     mov eax, -1     ; [rsi] < [rdi]
194     ret
195 .isGt:
196     mov eax, 1      ; [rsi] > [rdi]
197     ret
```

Обработка строк символов. Аналог strcmp языка C

```
; test-strcmp.asm - сравнение строк в стиле C-strcmp
%include "macros.mac"

section .data
    NL          equ     10

    msg         db      "Hello World!", 10,0
    info        db      "Message: ",0
    emptyMsg    db      0
    fDecimal    db      "%d",10,0
    len         dd      0
    flag        dd      0

section .bss
    strBuf      resb    256

section .text
    global main
;-------------------------------------------
main:
    push    rbp
    mov     rbp,rsp

    PrintStrBuf info, [stdout]
    PrintStr    NL, [stdout]
    PrintStrBuf msg, [stdout]

    mov     rsi, msg
    mov     rdi, info
    call    strcmp          ; < (-1)
    mov     [flag], eax
    PrintStr    "Compare flag = ", [stdout]
    PrintInt    [flag], [stdout]
    PrintStr    NL, [stdout]

    mov     rsi, info
    mov     rdi, msg
    call    strcmp          ; > (1)
    mov     [flag], eax
    PrintStr    "Compare flag = ", [stdout]
    PrintInt    [flag], [stdout]
    PrintStr    NL, [stdout]

    mov     rsi, msg
    mov     rdi, msg
    call    strcmp          ; == (0)
    mov     [flag], eax
    PrintStr    "Compare flag = ", [stdout]
    PrintInt    [flag], [stdout]
    PrintStr    NL, [stdout]

    mov     rsi, msg
    mov     rdi, strBuf
    call strcpy
    mov     rdi, msg
    mov     rsi, strBuf
    call    strcmp          ; == (0)
    mov     [flag], eax
    PrintStr    "Compare flag = ", [stdout]
    PrintInt    [flag], [stdout]
    PrintStr    NL, [stdout]

    mov     rsi, info
    mov     rdi, strBuf
    call    strcmp          ; < (-1)
    mov     [flag], eax
    PrintStr    "Compare flag = ", [stdout]
    PrintInt    [flag], [stdout]
    PrintStr    NL, [stdout]

    PrintStrLn "Bye!", [stdout]
    pop     rbp
    mov     rax,60
    mov     rdi,0
syscall
```

Обработка строк символов. Аналог strcmp языка C

```
Message:
Hello World!
Compare flag = -1
Compare flag = 1
Compare flag = 0
Compare flag = 0
Compare flag = 1
Bye!
```



http://softcraft.ru/edu/comparch/lect/09-IntelAsmBase/