Машинно-зависимые языки программирования

Савельев Игорь Леонидович

План

- Массивы
  - Статический массив
  - Интеграция с Си
    - Статический массив
    - Динамический массив
  - Динамический массив
  - Многомерный массив
- Примеры работы с массивом

1. Массивы

Массив – агрегатный тип данных, содержащий множество переменных одного типа, расположенных в памяти последовательно.

Объявление: type name [count];

```
char  array[10];
int  array[20];
```

Инициализация

```
char array[10] ={}; // нули
char array [4] = “ABC”;
int array[3] = {1, 2, 3};
int array[] = {1,2,3,4}; // размер массива  - 4 элемента!

//!!
char  array[N][M][K]; //многомерные массивы
```

1. Массивы

Имя массива – указатель!

```
char array[8] = {1,2,3,4,5,6,7,8};
char * pA = array;
```

Big endian, 32 бита

Адрес		0	1	2	3
0xFF000000	array	1	2	3	4
0xFF000004		5	6	7	8
0xFF000008	pA	FF	00	00	00

Размер массива

```
char array[16];
sizeof(array); //?    16
```

```
int array[16];
sizeof(array); //?    64
```

sizeof(array) = count * sizeof(type)

count = sizeof(array) / sizeof(type)

sizeof(array) != sizeof(pA)

Объявление, инициализация

```
section .data
   arrByte  db 0 ; uint8_t arrByte[4] = {0, 1, 2, 3};
            db 1
            db 2
            db 3

  arrByte1 db 0, 1, 2, 3 ; uint8_t arrByte1[4] = {0, 1, 2, 3};


section .bss
   arrByte2 resb 4 ; uint8_t arrByte2[4];
```

Объявление, инициализация

```
section .data
   arrShort  dw 0 ; uint16_t arrShort[4] = {0, 1, 2, 3};
             dw 1
             dw 2
             dw 3


section .bss
   arrShort2 resw 4 ; uint16_t arrShort2[4];
```

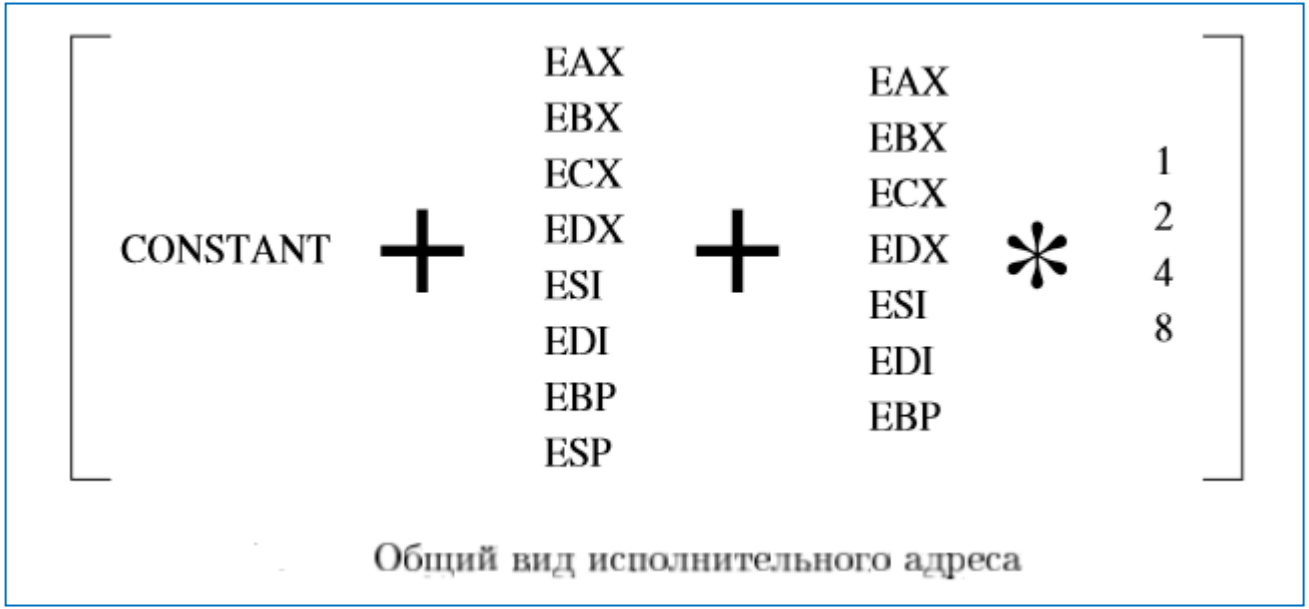
1. Массивы. Статический массив

Объявление, инициализация

```
section .data
    arrInt  times 4 dd 0 ; uint32_t arrInt[4] = {0};


section .bss
    arrInt2 resd 4 ; uint32_t arrInt2[4];
```

1. Массивы. Статический массив



Общий вид исполнительного адреса

1. «Имя» переменной (метка) == статический адрес == const
2. arr[0] == *(arr + 0) => [arr + 0]
3. arr[3] == *(arr + 3) => [arr + 3]
4. arr[i] == *(arr + i) => если mov ebx, [i] ,то [arr + ebx]

Предупреждение: справедливо только для массивов переменных типа byte

1. Массивы. Статический массив. Заполнение

```
section .data
   arrByte times 5 db 0          ; uint8_t arrByte[5] = {0, 0, 0, 0, 0}
   len       db $-arrByte        ; len = 5, так как массив физически занимает 5 байт
section .text
…
   xor eax, eax
   movzx ebx, byte [len]         ; условие выхода из цикла
   mov ecx, 0                    ; счетчик цикла
@lab:
   mov [arrByte + ecx], cl       ; *(arrByte + ecx) = cl – инициализируем значением
                                 ; младшего байта счетчика
   inc ecx                       ; увеличиваем счетчик
   cmp ecx, ebx                  ; проверяем условие выхода из цикла
   jb @lab
```

Итог: arrByte[5] = {0, 1, 2, 3, 4}

1. Массивы. Статический массив. Заполнение, регистр как база

```
section .data
   arrByte times 5 db 0          ; uint8_t arrByte[5] = {0, 0, 0, 0, 0}
   len       db $-arrByte        ; len = 5, так как массив физически занимает 5 байт
section .text
…
  xor eax, eax
   mov eax, arrByte              ; eax= arrByte – заносим адрес массива, &arrByte[0]
   movzx ebx, byte [len]         ; условие выхода из цикла
   mov ecx, 0                    ; счетчик цикла
@lab:
   mov [eax + ecx], cl           ; *(arrByte + ecx) = cl – инициализируем значением
                                 ; младшего байта счетчика
   inc ecx                       ; увеличиваем счетчик
   cmp ecx, ebx                  ; проверяем условие выхода из цикла
   jb @lab
```

Итог: arrByte[5] = {0, 1, 2, 3, 4}

1. Массивы. Статический массив. Доступ и изменение

```
section .data
   arrByte times 5 db 0          ; uint8_t arrByte[5] = {0, 0, 0, 0, 0}
   len       db $-arrByte        ; len = 5, так как массив физически занимает 5 байт
section .text
…
   movzx ebx, byte [len]         ; условие выхода из цикла
   mov ecx, 0                    ; счетчик цикла
@lab:
   mov al, byte [arrByte + ecx]  ; al = *(arrByte+ ecx) – читаем элемент массива
   add al, cl                    ; увеличиваем на значение младшего
                                 ; байта счетчика (игнорируем переполнение )
   mov [arrByte + ecx], al       ; *(arrByte + ecx) = al – сохраняем элемент массива
   inc ecx                       ; увеличиваем счетчик
   cmp ecx, ebx                  ; проверяем условие выхода из цикла
   jb @lab
```

Итог: arrByte[5] = {0, 1, 2, 3, 4}

1. Массивы. Статический массив. Адресная арифметика

```
uint8_t arr[3] = {1, 2, 3}; //sizeof(uint8_t) == 1

for(int i = 0; i < 3; i++)
{
    printf("Адрес 0x%x = %d\n", &arr[i], *(arr + i));
}
```

Output:
0x0acd0126 = 1;
0x0acd0127 = 2;
0x0acd0128 = 3;

1) Адреса ячеек отличаются на 1
2) i++ – изменяет адрес на 1

1. Массивы. Статический массив. Адресная арифметика

```
uint32_t arr[3] = {1, 2, 3}; //sizeof(uint32_t) == 4

for(int i = 0; i < 3; i++)
{
    printf("Адрес 0x%x = %d\n", &arr[i], *(arr + i));
}
```

Output:
0x0acd0126 = 1;
0x0acd012a = 2;
0x0acd012e = 3;

1) Адреса ячеек отличаются на 4
2) i++ – изменяет адрес на 4 (!)

В ассемблере (в отличии от Си) адресную арифметику надо делать вручную

1. Массивы. Статический массив. Доступ и изменение. DWORD

```
section .data
   arrInt times 5 dd 0          ; uint32_t arrInt[5] = {0, 0, 0, 0, 0}
   len       db $-arrInt        ; len = 5*4 = 20
section .text
…
   movzx ebx, byte [len]        ; условие выхода из цикла
   mov ecx, 0                   ; счетчик цикла
@lab:
   mov eax, dword [arrInt + ecx]  ; eax = *(arrInt+ ecx) – читаем элемент массива
   add eax, ecx                 ; увеличиваем на значение счетчика
                                ; (игнорируем переполнение )
   mov [arrInt + ecx], eax      ; *(arrInt + ecx) = eax – сохраняем элемент массива
   add ecx, 4                   ; увеличиваем счетчик на sizeof(uint32_t)
   cmp ecx, ebx                 ; проверяем условие выхода из цикла
   jb @lab
```

Итог: arrInt[5] = {0, 4, 8, 12, 16}

1. Массивы. Статический массив. Доступ и изменение. DWORD (2 вариант)

```
section .data
   arrInt times 5 dd 0              ; uint32_t arrInt[5] = {0, 0, 0, 0, 0}
   len            db $-arrInt       ; len = 5*4 = 20,
section .text
…
   movzx ebx, byte [len]            ; условие выхода из цикла
   shr ebx, 2                       ; делим на sizeof(uint32_t) => 20/4 = 5
   mov ecx, 0                       ; счетчик цикла
@lab:
   mov eax, dword [arrInt + ecx * 4]  ; eax = *(arrInt + ecx) – читаем элемент массива
                                      ; (адресная арифметика)
   add eax, ecx                     ; увеличиваем на значение счетчика (игнорируем переполнение )
   mov [arrInt + ecx * 4], eax      ; *(arrInt + ecx) = eax – сохраняем элемент массива
   inc ecx                          ; увеличиваем счетчик (адресная арифметика)
   cmp ecx, ebx                     ; проверяем условие выхода из цикла
   jb @lab
```

Итог: arrInt[5] = {0, 1, 2, 3, 4}

1. Массивы. Интеграция с Си. Статический массив.

```
uint8_t arrByte[5] = {0};
uint8_t len = 5;

void main() {
        asm_func();
}
```

Работа со статическим массивом, созданным в Си, такая же как и в ассемблере

```
section .data
    extern arrByte
    extern len
section .text
global asm_func
asm_func:
    xor eax, eax
    movzx ebx, byte [len]    ; условие выхода из цикла
    mov ecx, 0               ; счетчик цикла
@lab:
    mov [arrByte + ecx], cl  ; *(arrByte + ecx) = cl – инициализируем значением
                             ; младшего байта счетчика
    inc ecx                  ; увеличиваем счетчик
    cmp ecx, ebx             ; проверяем условие выхода из цикла
    jb @lab
    ret
```

Итог: arrByte[5] = {0, 1, 2, 3, 4}

1. Массивы. Интеграция с Си. Динамический массив.

Работа с динамическим массивом, созданным в Си, через указатель

```
uint8_t * pArr = 0;
uint8_t len = 3;

void main ()
{
   printf("Адрес pArr 0x%x\n", &pArr);
   pArr = (uint8_t*) malloc(sizeof(uint8_t) * len);
   printf("Адрес  массива  0x%x\n", pArr);

   asm_func();
   for(int i = 0; i < 3; i++)
   {
        printf("Адрес 0x%x = %d\n", &pArr[i], *(pArr + i));
   }
  free(pArr);
}
```

Output:
Адрес pArr 0x01034e5f
Адрес массива 0x454ef5ac

0x454ef5ac = 1;
0x454ef5ad = 2;
0x454ef5ae = 3;

1. Массивы. Интеграция с Си. Динамический массив.

Работа с динамическим массивом, созданным в Си, через указатель

```
section .data
   extern pArr
   extern len
section .text
global asm_func
asm_func:
   xor eax, eax
   mov edx, pArr;          ; edx = 0x01034e5f  (адрес указателя) (для 32 бит)
   mov eax, [pArr]         ; eax = 0x454ef5ac (адрес массива) (для 32 бит)
                           ; [eax]- значение первого элемента
   movzx ebx, byte [len]   ; условие выхода из цикла
   mov ecx, 0              ; счетчик цикла
@lab:
   mov [eax + ecx], cl     ; *(pArr + ecx) = cl – инициализируем значением
                           ; младшего байта счетчика
   inc ecx                 ; увеличиваем счетчик
   cmp ecx, ebx            ; проверяем условие выхода из цикла
   jb @lab
   ret
```

Итог: pArr => {0, 1, 2, 3, 4}

1. Массивы. Интеграция с Си. Динамический массив.

Работа с динамическим массивом, созданным в Си, через указатель, 64 бита

```
section .data
   extern pArr
   extern len
section .text
global asm_func
asm_func:
   xor eax, eax
   mov rdx, pArr;          ; rdx = 0x01034e5f  (адрес указателя) (для 64 бит), sizeof(uint8_t *) == 8 !!
   mov rax, [pArr]         ; eax = 0x454ef5ac (адрес массива) (для 64 бит)
                           ; [rax]- значение первого элемента
   movzx rbx, byte [len]   ; условие выхода из цикла
   mov rcx, 0              ; счетчик цикла
@lab:
   mov [rax + rcx], cl     ; *(pArr + rcx) = cl – инициализируем значением
                           ; младшего байта счетчика
   inc rcx                 ; увеличиваем счетчик
   cmp rcx, rbx            ; проверяем условие выхода из цикла
   jb @lab
   ret
```

Итог: pArr => {0, 1, 2, 3, 4}

1. Массивы. Динамический массив.

Аналог malloc() в ассемблере – sys_brk()

Для 32 бит:

```
mov eax, 45          ; 0x2d – sys_brk()
mov ebx, 0           ; получить начало
                     ; свободной памяти
int 0x80             ; в результате в eax – начало свободной памяти
```

Для выделения памяти нужно переместить границу – прибавить к «старой» границе нужное число байт. Начало выделенной памяти будет располагаться начиная со «старого» адреса.

```
add eax, 20          ; добавляем к старому адресу (см. выше) 20 байт
mov ebx, eax         ; указываем новую границу
mov eax, 45          ; 0x2d – sys_brk()
int 0x80             ; сдвгаем границу в бОльшую сторону на 20 байт;
                     ; в eax – новая граница
```

Для освобождения памяти – передвинуть границу в меньшую сторону

1. Массивы. Динамический массив.

Аналог malloc() в ассемблере – sys_brk()

Для 64 бит:

```
mov rax, 12          ; sys_brk()
mov rdi, 0           ; получить начало
                     ; свободной памяти
syscall              ; в результате в rax – начало свободной памяти
```

Для выделения памяти нужно переместить границу – прибавить к «старой» границе нужное число байт. Начало выделенной памяти будет располагаться начиная со «старого» адреса.

```
add rax, 20          ; добавляем к старому адресу (см. выше) 20 байт
mov rdi, rax         ; указываем новую границу
mov rax, 12          ; sys_brk()
syscall              ; сдвгаем границу в бОльшую сторону на 20 байт;
                     ; в rax – новая граница
```

Для освобождения памяти – передвинуть границу в меньшую сторону

1. Массивы. Динамический массив.

```
section .data
      pAddr dd 0;  32 битный указатель на int
section .text
global asmfunc
asmfunc:
   mov eax, 45; 0x2d – sys_brk()
   mov ebx, 0    ; получить начало
                  ; свободной памяти
   int 0x80       ; eax  - начало свободной памяти
   mov [pAddr], eax; запомнить
.
   mov ebx, eax   ; формируем адрес
                  ; новой границы
   add ebx, 12    ; добавляем к старому адресу
                   ; 12 байт = 3 * sizeof(int);
   mov eax, 45     ; sys_brk()
   int 0x80        ; eax  - новое значение
```

```
; далее работа как с динамическим массивом в Си

   mov eax, [pAddr] ;.
   mov ebx, 3 ; условие выхода из цикла
   mov ecx, 0 ; счетчик цикла
@lab:
   mov [eax + ecx], cl ; *(pArr + ecx) = cl
                       ; инициализируем
                       ;значением
                       ; младшего байта счетчика
   inc ecx             ; увеличиваем счетчик
   cmp ecx, ebx        ; проверяем условие
                       ; выхода из цикла
   jb @lab
   ret
```

Примечание: данный syscall() возвращает 32 битный указатель!!!

Некоторые выводы

1. Как выделять/освободить память?

   Использовать sys_brk()

2. Можно ли использовать для работы с памятью int 0x80 в 64 битном приложении?

   Можно, но осторожно. До тех пор, пока ОС выделяет вам память в пределах 2^32 байт (как в примерах выше, адреса динамических переменных укладываются в эту границу)

3. Можно ли, работая с указателями, переданными из Си (см. пример с mallloc()), использовать 32 битные регистры?

   - надо смотреть на размер указателя, что возвращает sizeof()
   - в 64 битном приложении адреса могут располагаться в границе 2^32 байт и работа с 32-битными регистрами не вызовет ошибку

1. Массивы. Многомерный массив

Двумерный массив.
Располагается в памяти последовательно, без разрывов.
Сначала первая строка, затем вторая и т.д.

Доступ к элементу
Пусть массив ARR, строк N, колонок M – **ARR[N][M].**
Размер одного элемента (sizeof)= K
Тогда:
Размер одной строки S == M * K байт
Начало X строки относительно начала массива == X * S
Начало Y колонки относительно начала строки== Y * K
Адрес элемента строки X и колонки Y **(&arr[X][Y])** :
== адрес начала массива + X * S + Y * K == адрес начала массива + (X * M + Y) * K
Значение элемента **(arr[X][Y])** == [адрес начала массива + (X * M + Y) * K]

1. Массивы. Многомерный массив

Двумерный массив.

```
;uint8_t arrByte[20][10];
;N == 20
;M == 10
;K == sizeof(uint8_t) == 1

; a = arrByte[5][8]
; X == 5, Y == 8
; адрес начала массива + (X * M + Y) * K
```

a = *(arrByte + (5 * 10 + 8) * 1) = *(arrByte + 58)

1. Массивы. Многомерный массив

Двумерный массив.

```
;uint32_t arrInt[20][10];
;N == 20
;M == 10
;K == sizeof(uint32_t) == 4

; a = arrInt[5][8]
; X == 5, Y == 8
; адрес начала массива + (X * M + Y) * K
```

a = *(arrInt + (5 * 10 + 8) * 4) = *(arrInt + 58 * 4) = *(arrInt + 232)

1. Массивы. Многомерный массив

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>

extern  int  asmfunc(void);
uint8_t arr[10][10]= {0};

int main (void)
{
   asmfunc();

   for (int i = 0; i < 10; i++) {
     for (int j = 0; j < 10; j++) {
       printf("%02d ", arr[i][j]);
     }
     printf ("\n");
   }
   return 0;
  }
```

```
section .data
extern arr
m db 10;   число колонок
n db 10;   число строк
; K = 1
; [адрес начала массива + (X * M + Y) * K]

section .text
global asmfunc
asmfunc:
  mov bl, 0   ;  индекс
  mov dl, 0   ;  индекс
```

```
saveliev@srv-3:~/jobs/10$ ./mytest
00 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29
30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 59
60 61 62 63 64 65 66 67 68 69
70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89
90 91 92 93 94 95 96 97 98 99
```

```
@loop_n:
@loop_m:
  xor eax, eax
  mov al, byte [m]
  mul bl
  add al, dl
  ; mul byte [k]
  mov byte [arr + eax], al
  inc dl
  cmp dl, [m]
  jne @loop_m
  inc bl
  cmp bl, [n]
  jne @loop_n
  ret
```

1. Массивы. Многомерный массив

Трехмерный массив.
uint8_t arr[10][20][30];

Располагается последовательно, без разрывов. Сначала первый двумерный массив, затем второй и т.д.

Доступ к элементу
Пусть массив ARR, таблиц V, строк N, колонок M – ARR[V][N][M].
Размер одного элемента (sizeof) = K
Тогда:
Размер одной строки S == M * K байт
Размер одной двумерной таблицы C == N * M * K байт
Начало X строки относительно начала таблицы == X * S
Начало Y колонки относительно начала строки == Y * K
Начало Z таблицы относительно начала массива == Z * C
Адрес элемента таблицы Z строки X и колонки Y **(&arr[Z][X][Y])**:
== адрес начала массива + Z * C + X * S + Y * K
== адрес начала массива + ((Z * N + X) * M + Y) * K
Значение элемента **(arr[Z][X][Y])** == [адрес начала массива + ((Z * N + X) * M + Y) * K]

2. Простой пример 1. Десятичное число в строку

```
section .data
   m_value dd -12345678    ; число 32 бита
   pStr times 11 db ' '    ; буфер для минуса и 10 знаков
   len db $-pStr           ; длина на буфера == 11
   sig db 0                ; знак

section .text
global CMAIN
CMAIN:
   mov  ecx, 10
   mov  eax, [m_value]
   test eax,eax
   jns @ns                 ; проверка на знак
   mov [sig], byte 1       ; запоминаем
   neg eax                 ; убираем знак
@ns:
   mov  edi, len           ; метка len - адрес
                           ; конца буфера pStr
                           ; edi == pStr + len
                           ; edi – указатель, куда писать
                           ; в буфер
```

```
@r:
   xor  edx,edx       ; обнуляем старшую часть для деления
   div  ecx           ; получаем в edx остаток деления
   add  dl, '0‘       ; перевод в ASCII
   dec  edi           ; смещаем указатель
   mov  [edi], dl     ; пишем в буфер
   test eax, eax      ; проверка на окончание цикла
   jnz    @r          ; если есть что делить - продолжаем
   mov al, [sig]      ; если число было отрицательным
   test al, al        ; то должны записать в буфер ‘–’
   jz @z              ; иначе - игнорируем
   dec edi            ; смещаем указатель
   mov [edi], byte '-'    ; пишем минус
@z:       ; закончили алгоритм, надо вывести в консоль
   mov eax, 4         ; sys_write()
   mov ebx, 1         ; stdout
   mov ecx, edi       ; edi - адрес начала готовой строки.
   mov edx, len
   sub edx, edi       ; edx - длина фактической строки
   int  0x80
   ret
```

2. Простой пример 2. Строку в десятичное число

```
section .data
   m_value dd 0         ; число 32 бита
   pStr db "-1234567"    ; буфер для минуса и 10 знаков
   len db $-pStr         ; для на буфера == 11
   sig db 0              ; знак

section .text
global CMAIN
CMAIN:
      xor eax, eax
      xor edx, edx              ; итог
      mov esi, pStr             ; начало строки
      mov ebx, len              ; длина
      cld                       ; для цепочечной команды
                                ; указываем направление
      mov ah,0
      lodsb                     ; берем первый символ
      cmp al, '-‘               ; проверяем на знак
      jnz @ns
      mov [sig], byte 1         ; сохраняем признак отр.знака
```

```
@ns:
@lp1:
                cmp ebx, esi  ;если вышли за пределы строки,
                jz @ex          ; то заканчиваем
                lodsb           ; берем символ
                sub al,'0'      ; получаем цифровое значение
                cmp al, 9       ; проверка на корректность
                ja @lp1
                imul edx, 10 ;умножаем сумму на 10
                add edx, eax;прибавляем текущее значение
                jmp @lp1
@ex:
      mov al, [sig]     ; если был знак, то восстанавливаем
      test al, al
      jz @ex1
      neg edx
@ex1:
      mov [m_value], edx      ; сохраняем результат
      ret
```

Спасибо