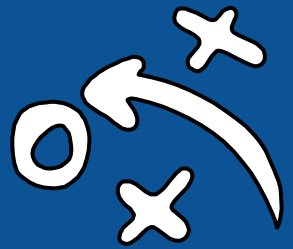
Алгоритмы и Алгоритмические Языки

Семинар #28:

1. Планы на модуль и система оценивания.
2. Организация ассемблерной программы.
3. Секции `.data`, `,rodata`, `.text`, `.bss` и их содержимое.
4. Пошаговое исполнение бинарного кода в отладчике.

16.01.2026

Планы на модуль
и система оценивания



ПУД и система оценивания

ПУД когда-то появится по ссылке.

Элемент контроля	Вес (без автомата)	Вес (с автоматом)
ДЗ №1	0.08	0.08
ДЗ №2	0.08	0.08
ДЗ №3	0.08	0.08
ДЗ №4	0.08	0.08
ДЗ №5 (проект)	0.08	0.08
КР №1	0.15	0.2
КР №2	0.15	0.2
Экзамен	0.3	**Автомат**

Полезная литература

Техническая документация:

1. Intel Software Development Manual
2. Документация NASM.
3. System V i386 Application Binary Interface

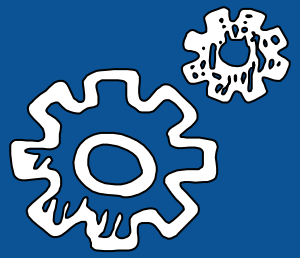
Учебники и методические пособия:

1. См. в специальном разделе на сайте курса.

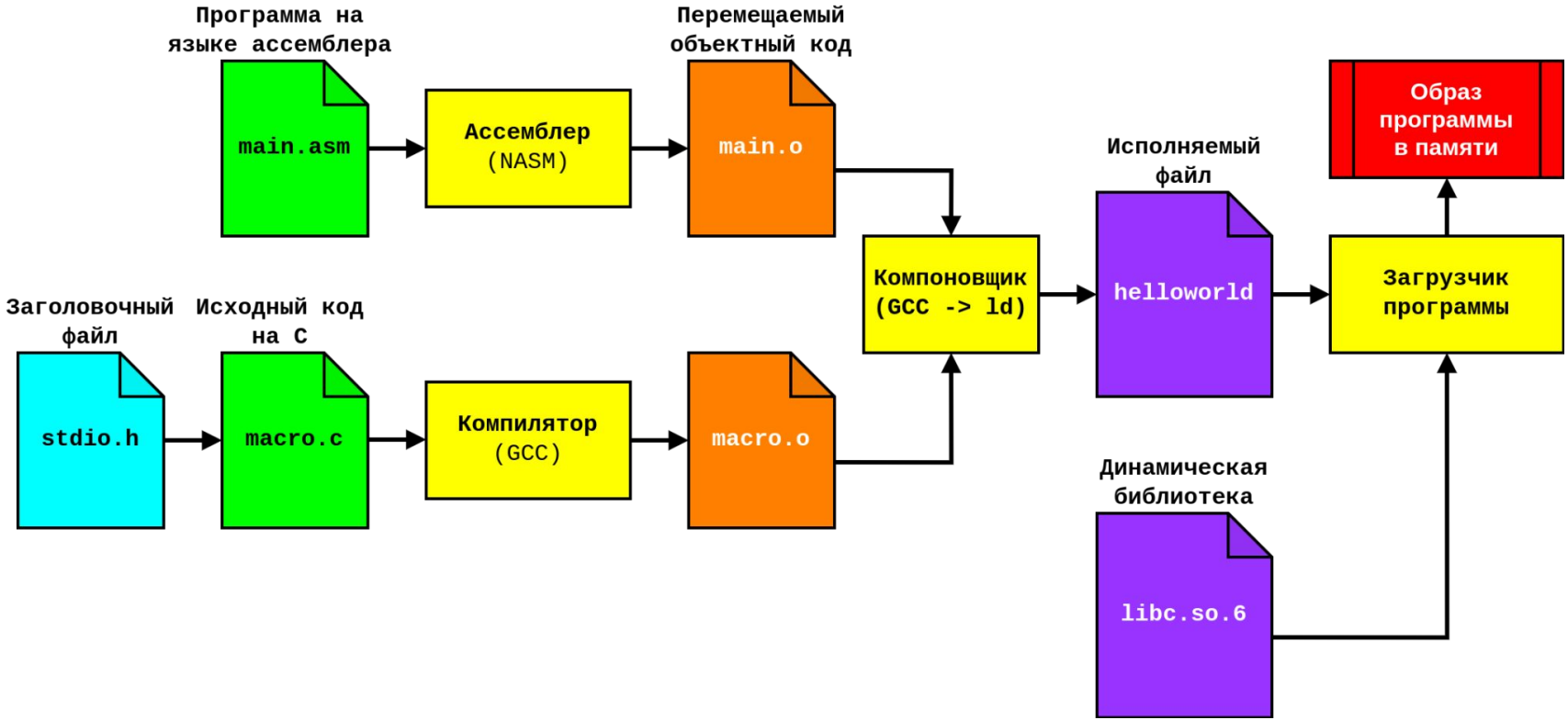
Онлайн-ресурсы:

1. Сайт курса в МГУ. В частности, инструкция по установке ПО.

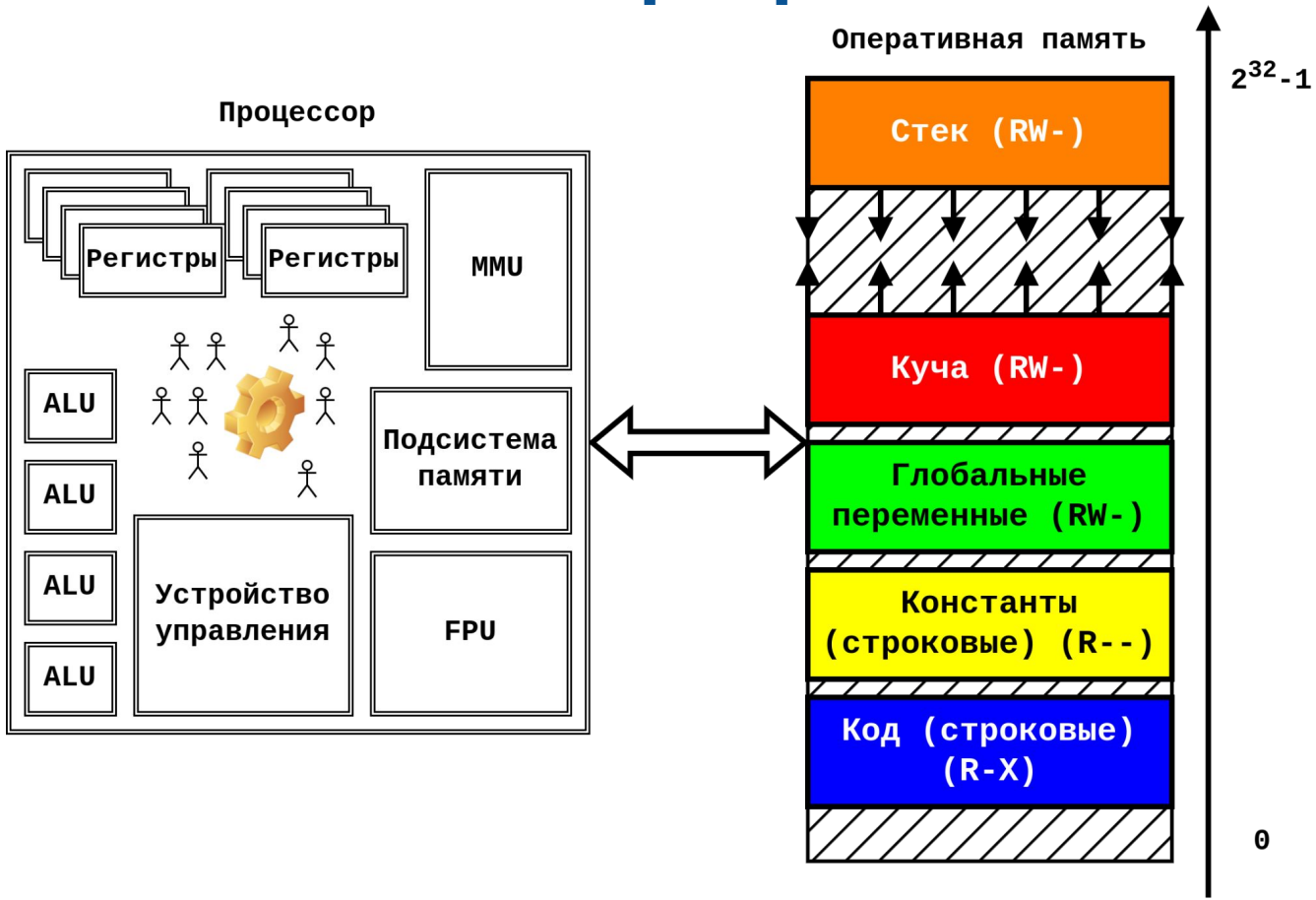
Организация ассемблерной программы



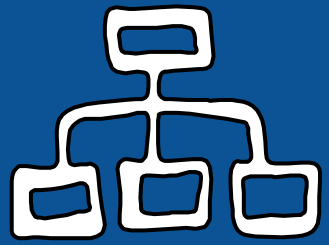
Сборка и выполнение программы



Работа программы



Секции .data, .text, .bss и их содержимое



Секция .text

```
; Секция .text содержит исполняемый код
section .text
global main
main:
    ; Вывод строки "Hello, world!"
    mov eax, hello ; eax = hello
    call io_print_string

    ; Вывод 32-битного шестнадцатеричного числа.
    ; io_print_hex принимает аргумент через eax
    mov eax, [dword var_c] ; eax = *var_c
    call io_print_hex
    call io_newline

    ; Вывод строки "Enter two numbers to be added:"
    mov eax, addstr ; eax = addstr
    call io_print_string

    ; Чтение 32-битного целого числа.
    ; Возвращаемое значение в регистре EAX
    call io_get_dec
    mov ebx, eax ; ebx = eax
```

Секция .text в дизассемблере

```
080491f0  int32_t main(int32_t argc, char** argv, char** envp)

080491f0  b808a00408          mov     eax, hello  {"Hello, world!\n"}
080491f5  e82e030000          call    io_print_string
080491fa  a13bc00408          mov     eax, dword [var_c]
080491ff  e899020000          call    io_print_hex
08049204  e869030000          call    io_newline
08049209  b817a00408          mov     eax, addstr  {"Enter two numbers to be added:\n"}
0804920e  e815030000          call    io_print_string
08049213  e858000000          call    io_get_dec
08049218  89c3                mov     ebx, eax
0804921a  e851000000          call    io_get_dec
0804921f  01c3                add     ebx, eax
08049221  b837a00408          mov     eax, rsltstr  {"The result is:\n"}
08049226  e8fd020000          call    io_print_string
0804922b  89d8                mov     eax, ebx
0804922d  e8db010000          call    io_print_dec
08049232  e83b030000          call    io_newline
08049237  31c0                xor     eax, eax  {0x0}
08049239  c3                  retn     {__return_addr}
```

Секции .data, .rodata, .bss

```
; Секция .rodata содержит неизменяемые инициализированные данные
section .rodata
hello:   db `Hello, world!\n`, 0
addstr:  db `Enter two numbers to be added:\n`, 0
rsltstr: db `The result is:\n`, 0

; Секция .data содержит изменяемые инициализированные данные.
section .data
var_a   db 0xAA                ; Байт данные
var_b   dw 0xAABB              ; Слово данных (16 бит)
var_c   dd 0xDEADBEEF          ; Двойное слово (32 бита)
var_d   dq 0xDEADBEEFB01DFACE  ; Четверное слово (64 бита)

; Секция .bss содержит изменяемые неинициализированные данные.
section .bss
bss_a   resb 20      ; Неинициализированная область размером  20 байт
bss_b   resw 20      ; Неинициализированная область размером  40 байт
bss_c   resd 20      ; Неинициализированная область размером  80 байт
bss_d   resq 20      ; Неинициализированная область размером 160 байт

; Повторение некоторого заданного паттерна.
pattern times 16 dw 0xABCD
```

Секция .rodata в дизассемблере

```
.rodata (PROGBITS) section started  {0x804a000-0x804a050}
0804a000  uint32_t _fp_hw = 0x3
0804a004  uint32_t _IO_stdin_used = 0x20001
0804a008  char const hello[0xf] = "Hello, world!\n", 0
0804a017  char const addstr[0x20] = "Enter two numbers to be added:\n", 0
0804a037  char const rsltstr[0x10] = "The result is:\n", 0
```

```
.rodata (PROGBITS) section started  {0x2000-0x2050}
00002000  03 00 00 00 01 00 02 00-48 65 6c 6c 6f 2c 20 77  ........Hello, w
00002010  6f 72 6c 64 21 0a 00 45-6e 74 65 72 20 74 77 6f  orld!..Enter two
00002020  20 6e 75 6d 62 65 72 73-20 74 6f 20 62 65 20 61   numbers to be a
00002030  64 64 65 64 3a 0a 00 54-68 65 20 72 65 73 75 6c  dded:..The resul
00002040  74 20 69 73 3a 0a 00 25-64 00 25 75 00 25 58 00  t is:..%d.%u.%X.
.rodata (PROGBITS) section ended  {0x2000-0x2050}
```

Секция .data в дизассемблере

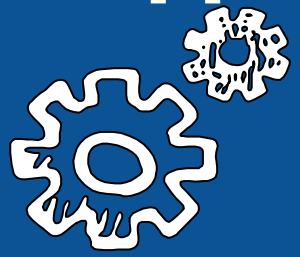
```
.data (PROGBITS) section started  {0x3030-0x3067}
00003030  00 00 00 00 00 00 00 00-aa bb aa ef be ad de ce
00003040  fa 1d b0 ef be ad de cd-ab cd ab cd ab cd ab cd
00003050  ab cd ab cd ab cd ab cd-ab cd ab cd ab cd ab cd
00003060  ab cd ab cd ab cd ab
.data (PROGBITS) section ended  {0x3030-0x3067}
```

```
.data (PROGBITS) section started  {0x804c030-0x804c067}
0804c030  __data_start:
0804c030  00 00 00 00                                      ....
0804c034  __dso_handle:
0804c034              00 00 00 00                              ....

0804c038  uint8_t var_a = 0xaa
0804c039  uint16_t var_b = 0xaabb
0804c03b  uint32_t var_c = 0xdeadbeef
0804c03f  uint64_t var_d = 0xdeadbeefb01dface
0804c047  uint16_t pattern = 0xabcd

0804c049                                cd ab cd ab cd ab cd           .......
0804c050  ab cd ab cd ab cd ab cd-ab cd ab cd ab cd ab cd  ................
0804c060  ab cd ab cd ab cd ab                                     .......
.data (PROGBITS) section ended  {0x804c030-0x804c067}
```

Секция .bss в дизассемблере

```
0x804c06c .bss (NOBITS) {0x804c068-0x804c198} Writable data

0804c06c  uint8_t bss_a = 0x0

0804c06d                                          00 00 00              ...
0804c070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................

0804c080  uint16_t bss_b = 0x0

0804c082        00 00 00 00 00 00-00 00 00 00 00 00 00 00    ..............
0804c090  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0804c0a0  00 00 00 00 00 00 00 00                          ........

0804c0a8  uint32_t bss_c = 0x0

0804c0ac                                       00 00 00 00              ....
0804c0b0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0804c0c0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0804c0d0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0804c0e0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0804c0f0  00 00 00 00 00 00 00 00                          ........

0804c0f8  uint64_t bss_d = 0x0

0804c100  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0804c110  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0804c120  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0804c130  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0804c140  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0804c150  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0804c160  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0804c170  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0804c180  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0804c190  00 00 00 00 00 00 00 00                          ........
.bss (NOBITS) section ended  {0x804c068-0x804c198}
```

Пошаговое исполнение бинарного кода в отладчике



Операции в отладчике

STEP – шаг на следующую строку кода со входом в функцию.
NEXT – шаг на следующую строку кода без входа в функции.

STEP INSTRUCTION – шаг на следующую инструкцию со входом в функции.
NEXT INSTRUCTION – шаг на следующую инструкцию без входа в функции.

BREAKPOINT func – установка точки останова на функцию func.
CONTINUE – продолжение исполнения программы.

Вопросы?

Красивые иконки взяты с сайта handdrawngoods.com