

Лекция 3 Применение языка С для встраиваемых систем

План курса «Встраиваемые микропроцессорные системы»:

Лекция 1: Введение. Язык программирования С

Лекция 2: Язык программирования С. Стандартная библиотека языка С

Лекция 3: Применение языка С для встраиваемых систем

Лекция 4: Микроконтроллер

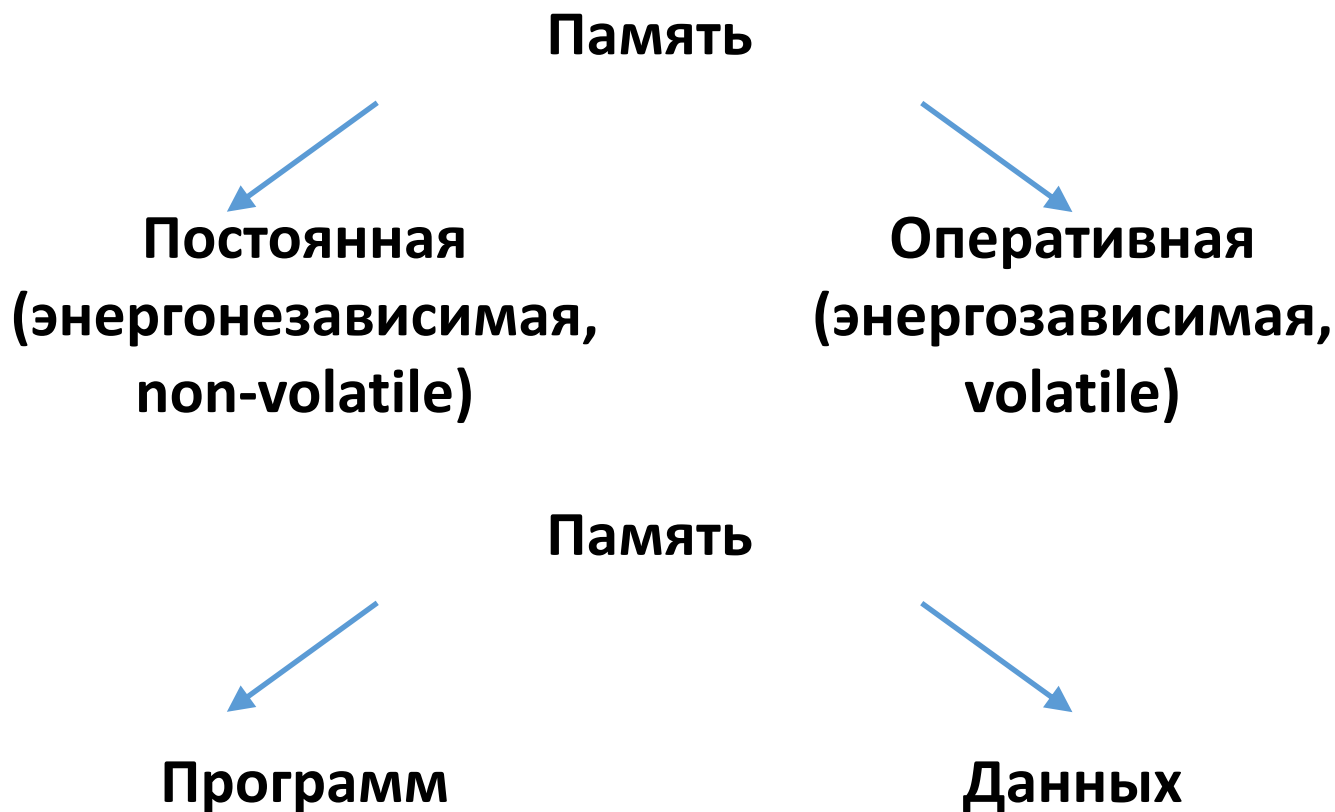
Лекция 5: Этапы разработки встраиваемых систем

Лекция 6: Разработка и отладка программ для встраиваемых систем

Лекция 7: Архитектура программ для встраиваемых систем

Лекция 8: Периферийные модули: DMA, USB, Ethernet

Классификация памяти



Постоянная память (ПЗУ) преимущественно используется как память программ, но также для данных – константы.

Оперативная память (ОЗУ) преимущественно используется как память данных, но в высокопроизводительных системах и как память программ.

Модель памяти в языке С



Динамическая и статическая память

Статическая память

Выделение памяти при компоновке (link time).

- + Легко управлять. Быстрая инициализация.
- + Детерминированное поведение. Атомарность.
- Фиксированный размер.

Оптимально когда нужно использовать ресурсы одновременно.

```
#define ARR_SIZE 32
char x[ARR_SIZE]; // создание
char a[ARR_SIZE];

...
{
    x = {...};          // инициализация
    a = {...};
    ...
    filter(x, a);      // исполнение
}
...
```

Динамическая ручная память

Выделение памяти при исполнении (run time).

- + Совместное использование общих ресурсов.
- + Память может быть освобождена.
- Сложнее управлять.
- Недетерминированное поведение. Не атомарность.

Оптимально когда неизвестен необходимый объем памяти или несколько задач использует одни и те же ресурсы.

```
#define ARR_SIZE 32
char* x = malloc(ARR_SIZE); // создание
char* a = malloc(ARR_SIZE);

...
x = {...};                  // инициализация
a = {...};
...
filter(x, a);               // исполнение
...
free(x);                    // освобождение
free(a);
```

Секции памяти

Наименование	Название компоновщика gcc	Название компоновщика Keil	Описание
Код программы	.text	ER_CODE, .text	Содержит исполняемый код.
Глобальные переменные, инициализированные константой	.data	RW_DATA, .data	Содержит переменные, которые инициализированы до старта программы.
Глобальные переменные, инициализированные нулем	.bss	RW_IRAM, .bss	Содержит переменные, которые инициализированы нулями.
Константы	.rodata	ER_RODATA, .constdata	Секция для констант и неизменяемых данных.
Стек	Неопределена по умолчанию (задается в скрипте линкера)	Неопределена по умолчанию (задается в скрипте линкера)	Хранит данные во время выполнения функций (локальные переменные, адреса возврата).
Куча	Неопределена по умолчанию (задается в скрипте линкера)	Неопределена по умолчанию (задается в скрипте линкера)	Используется для динамического выделения памяти во время выполнения программы.
Таблица векторов прерываний	Обычно находится в начале .text или отдельной секции .vectors (.isr_vectors)	ER_VECTORS, .intvec	Хранит адреса обработчиков прерываний.

Модель памяти в языке C

ОЗУ

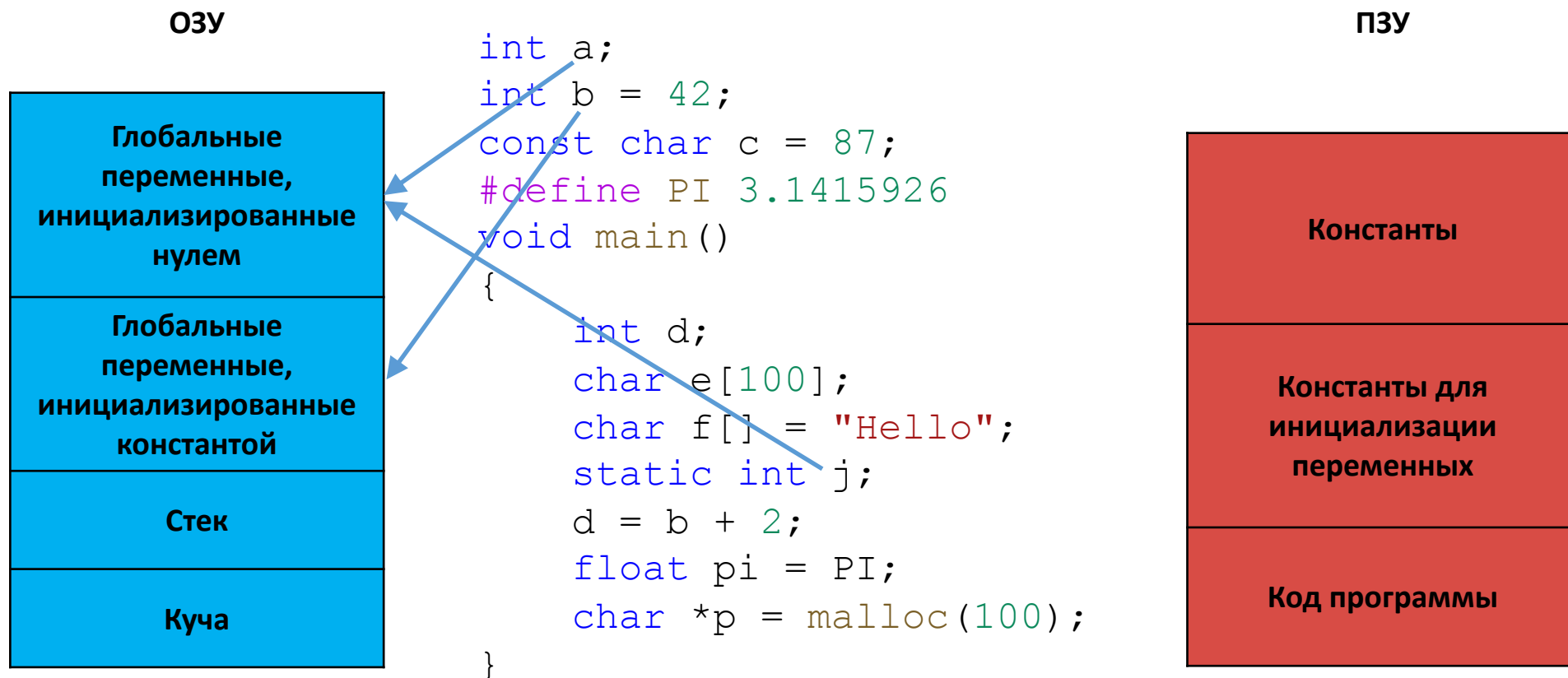
Глобальные переменные, инициализированные нулем
Глобальные переменные, инициализированные константой
Стек
Куча

```
int a;  
int b = 42;  
const char c = 87;  
#define PI 3.1415926  
void main()  
{  
    int d;  
    char e[100];  
    char f[] = "Hello";  
    static int j;  
    d = b + 2;  
    float pi = PI;  
    char *p = malloc(100);  
}
```

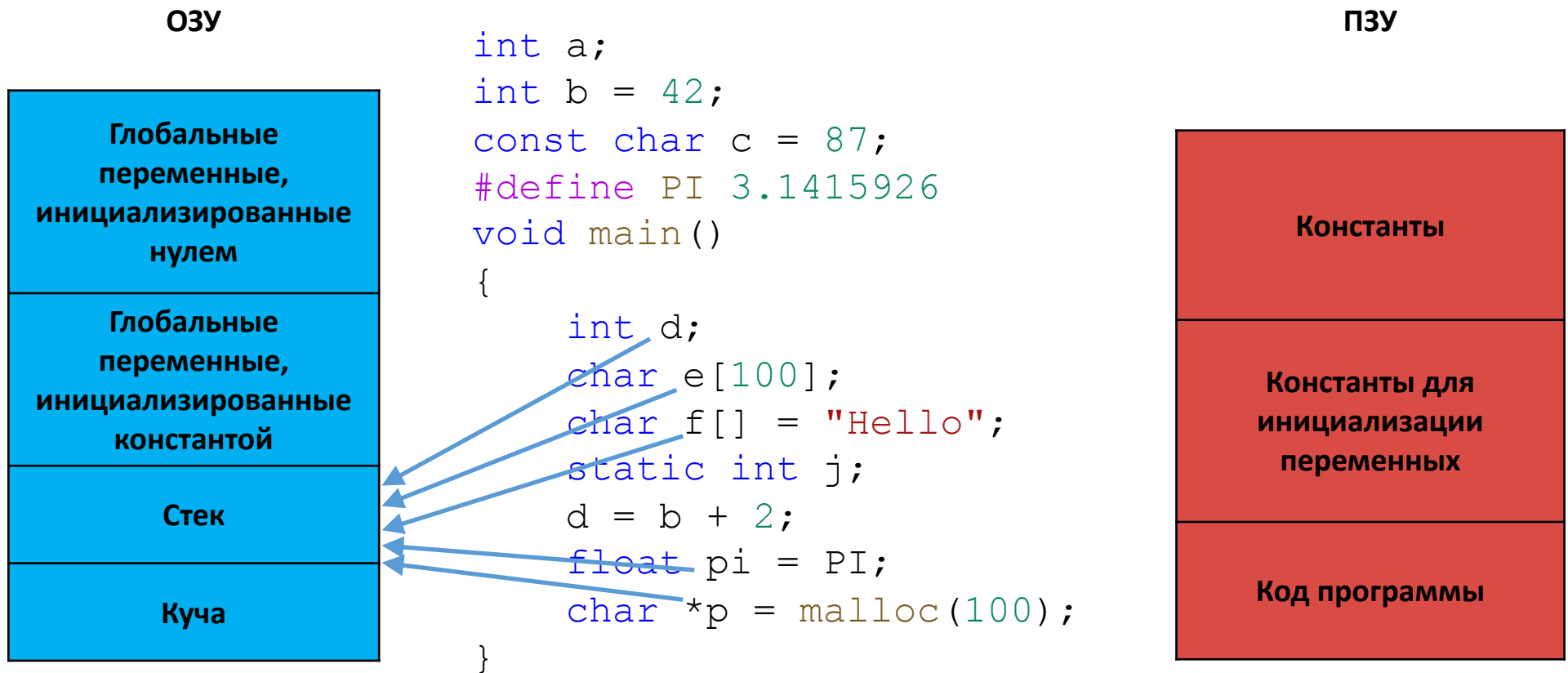
ПЗУ

Константы
Константы для инициализации переменных
Код программы

Модель памяти в языке C: глобальные переменные



Модель памяти в языке C: локальные переменные



Модель памяти в языке C: куча

ОЗУ

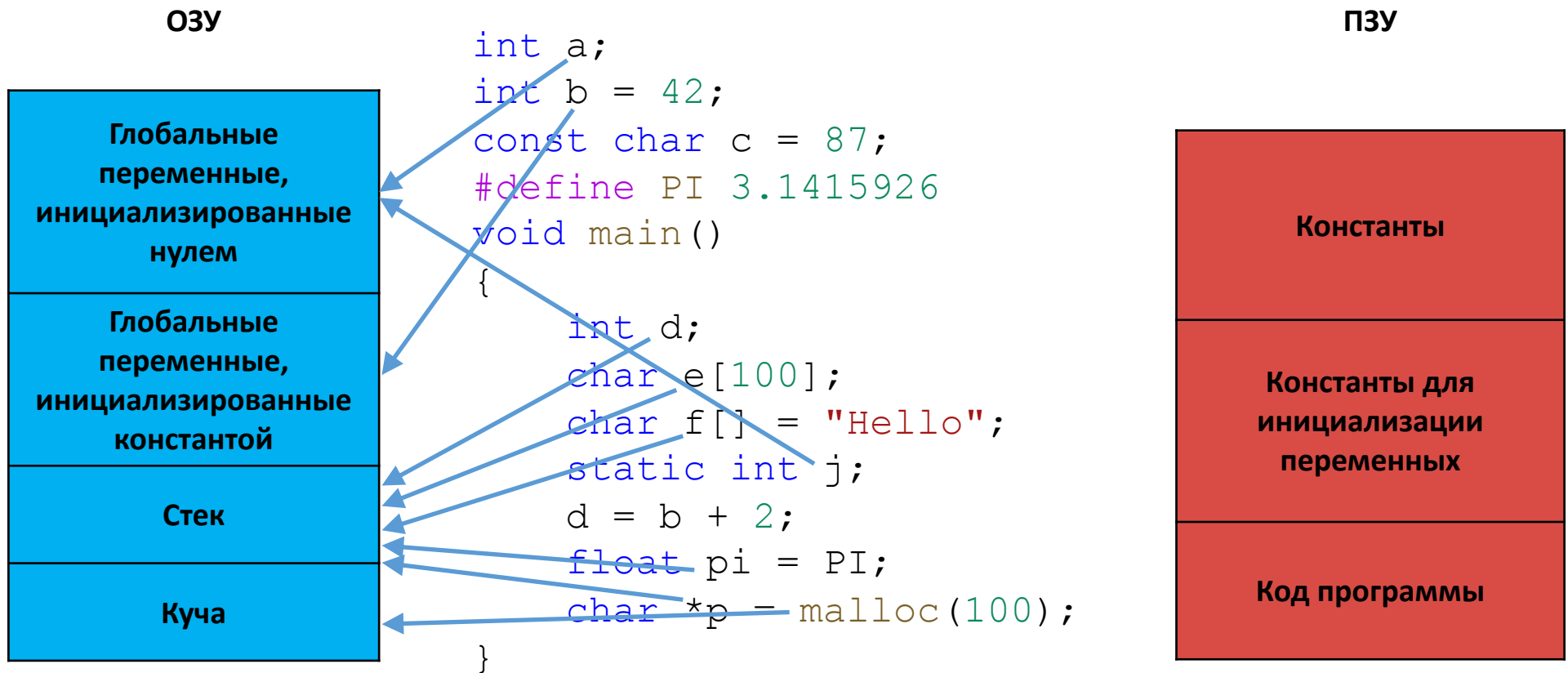


```
int a;  
int b = 42;  
const char c = 87;  
#define PI 3.1415926  
void main()  
{  
    int d;  
    char e[100];  
    char f[] = "Hello";  
    static int j;  
    d = b + 2;  
    float pi = PI;  
    char *p = malloc(100);  
}
```

ПЗУ



Модель памяти в языке C



Модель памяти в языке С: константы

ОЗУ



```
int a;  
int b = 42;  
const char c = 87;  
#define PI 3.1415926  
void main()  
{  
    int d;  
    char e[100];  
    char f[] = "Hello";  
    static int j;  
    d = b + 2;  
    float pi = PI;  
    char *p = malloc(100);  
}
```

ПЗУ



Модель памяти в языке C: макроопределения

ОЗУ



```
int a;  
int b = 42;  
const char c = 87;  
#define PI 3.1415926  
void main()  
{  
    int d;  
    char e[100];  
    char f[] = "Hello";  
    static int j;  
    d = b + 2;  
    float pi = PI;  
    char *p = malloc(100);  
}
```

ПЗУ



Модель памяти в языке C

ОЗУ

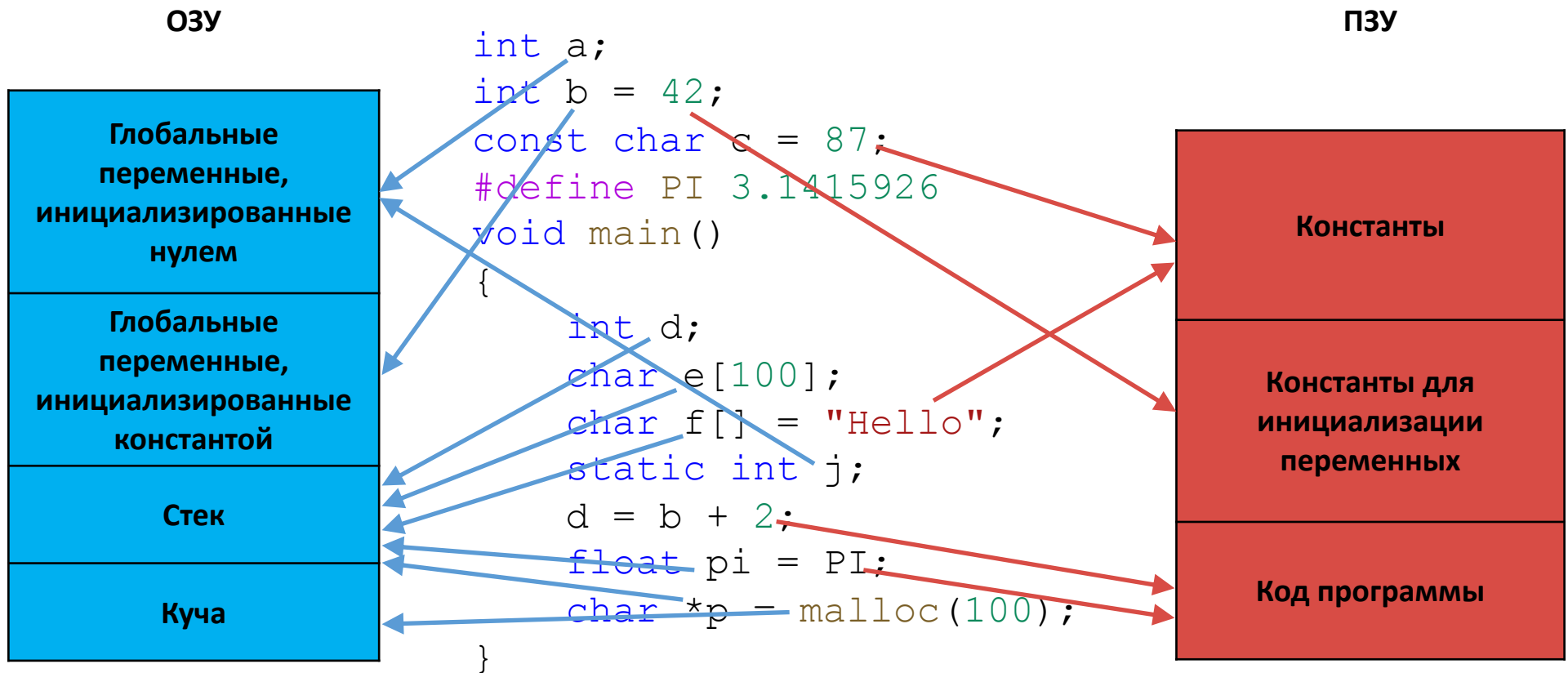


```
int a;  
int b = 42;  
const char c = 87;  
#define PI 3.1415926  
void main()  
{  
    int d;  
    char e[100];  
    char f[] = "Hello";  
    static int j;  
    d = b + 2;  
    float pi = PI;  
    char *p = malloc(100);  
}
```

ПЗУ



Модель памяти в языке C



Применение языка С для встраиваемых систем

Программирование встраиваемых систем – это низкоуровневое программирование. Поэтому язык должен иметь доступ к аппаратному обеспечению на регистровом уровне.

- Обращение к регистрам специальных функций периферийных модулей;
- Стартовый код (startup);
- Обработка прерываний;
- Ассемблерные вставки;
- Стандартная библиотека и системные вызовы.

Уровни абстракции:

Обращение к регистрам специальных функций

1. Язык C + документация

```
#define PORTC *((volatile unsigned int *) (0x400B8000))  
  
...  
PORTC = PORTC | 1; /* Установить 1 в PC0 */
```

2. Язык C + заголовочные файл + документация

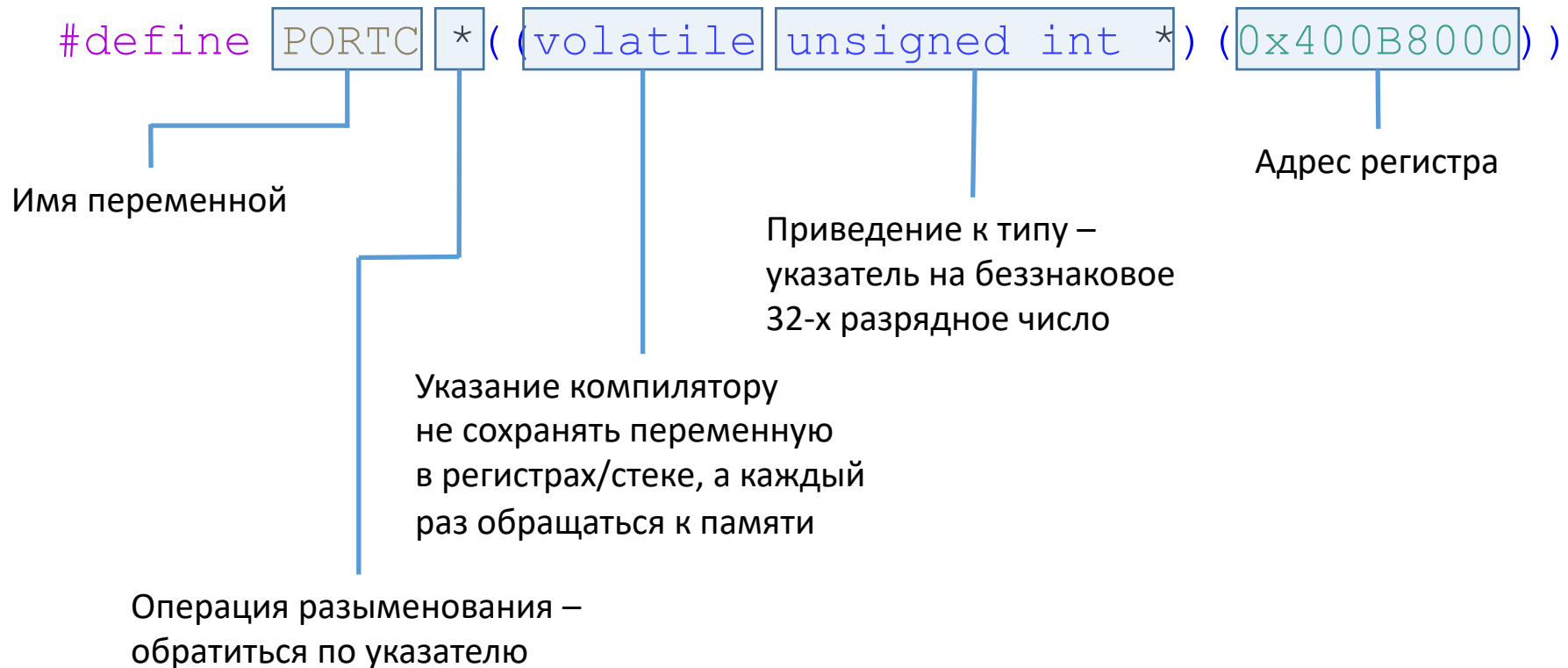
```
#include "MDR32Fx.h"  
  
...  
MDR_PORTC->RXTX = MDR_PORTC->RXTX | 1; /* Установить 1 в PC0 */
```

3. Язык C + библиотека (например, Standard Peripheral Library для K1986BE92QI)

```
#include <MDR32F9Qx_port.h>  
  
...  
PORT_SetBits(MDR_PORTC, PORT_Pin_0); /* Установить 1 в PC0 */
```


Уровни абстракции:

Обращение к регистрам специальных функций



Первая программа на С для микроконтроллера. Версия 1

```
#define RST_CLK_PER    *((volatile unsigned int *) (0x4002001C))
#define PORTC_RXTX    *((volatile unsigned int *) (0x400B8000))
#define PORTC_OE      *((volatile unsigned int *) (0x400B8004))
#define PORTC_ANALOG  *((volatile unsigned int *) (0x400B800C))
#define PORTC_PWR     *((volatile unsigned int *) (0x400B8018))

/* Функция main. Точка входа в программу */
int main(void)
{
    RST_CLK_PER = RST_CLK_PER | (1 << 23); /* Включаем тактирование порта С */

    PORTC_OE = PORTC_OE | 0x01;             /* Настраиваем ножку 0 порта С на вывод */
    PORTC_ANALOG = PORTC_ANALOG | 0x01;     /* Включаем цифровой режим работы ножки 0 */
    PORTC_PWR = PORTC_PWR | 0x02;           /* Настраиваем мощность выходного буфера ножки 0 */

    for (;;)
    {
        PORTC_RXTX = PORTC_RXTX ^ 0x01;    /* Инвертирование бита в регистре PORTC */
        for (int i = 0; i < 100000; i++); /* Программная задержка */
    }
}
```

Первая программа на С для микроконтроллера. Версия 2

```
#include <MDR32Fx.h>

/* Функция main. Точка входа в программу */
int main(void)
{
    /* Включаем тактирование порта С */
    MDR_RST_CLK->PER_CLOCK = MDR_RST_CLK->PER_CLOCK | (1 << 23);

    MDR_PORTC->OE = MDR_PORTC->OE | 0x01;          /* Настраиваем ножку 0 порта С на вывод */
    MDR_PORTC->ANALOG = MDR_PORTC->ANALOG | 0x01; /* Включаем цифровой режим работы ножки 0 */
    MDR_PORTC->PWR = MDR_PORTC->PWR | 0x02; /* Настраиваем мощность выходного буфера ножки 0 */

    for (;;)
    {
        MDR_PORTC->RXTX = MDR_PORTC->RXTX ^ 0x01; /* Инвертирование бита в регистре PORTC */
        for (int i = 0; i < 100000; i++); /* Программная задержка */
    }
}
```

Первая программа на С для микроконтроллера. Версия 3

```
#include <MDR32Fx.h>
#include <MDR32F9Qx_config.h>
#include <MDR32F9Qx_rst_clk.h>
#include <MDR32F9Qx_port.h>

int main() {
    RST_CLK_PCLKcmd(RST_CLK_PCLK_PORTC, ENABLE);
    PORT_InitTypeDef Port_InitStructure;
    PORT_StructInit(&Port_InitStructure);
    Port_InitStructure.PORT_Pin = PORT_Pin_0;
    Port_InitStructure.PORT_OE = PORT_OE_OUT;
    Port_InitStructure.PORT_FUNC = PORT_FUNC_PORT;
    Port_InitStructure.PORT_SPEED = PORT_SPEED_FAST;
    Port_InitStructure.PORT_MODE = PORT_MODE_DIGITAL;
    PORT_Init(MDR_PORTC, &Port_InitStructure);
    for(;;) {
        for (int i = 0; i < 100000; i++);
        PORT_SetBits(MDR_PORTC, PORT_Pin_0);
        for (int i = 0; i < 100000; i++);
        PORT_ResetBits(MDR_PORTC, PORT_Pin_0);
    }
}
```

Стартовый код (startup)

Стартовый код обеспечивает корректный запуск программы. Без инициализации ресурсов система может быть неправильно настроена.

Основные функции стартового кода:

1. Инициализация аппаратных ресурсов: настройка тактовых генераторов (как правило, только частоты работы процессора), настройка стека (Stack Pointer), инициализация памяти (копирование данных из секции `.data` в ОЗУ и очистка секции `.bss` (секции для неинициализированных данных));
2. Установка обработчиков прерываний;
3. Переход на точку входа в программу: передача управления функции `main()`;
4. Обработка аппаратных исключений: в случае возникновения аппаратных исключений (например, попытка доступа к неинициализированной памяти), startup код может содержать базовый обработчик для их отлова и корректной обработки;
5. Поддержка отладочных функций: включение/отключение отладочных интерфейсов и функций.

В операционных системах общего назначения (Linux, Windows, macOS) эти действия производит сама операционная система перед запуском любой программы.

Обработка прерываний в Cortex-M3

Файл startup_MDR32F9Qx.s:

```
AREA RESET, DATA, READONLY
__Vectors DCD __initial_sp ; Top of Stack
DCD Reset_Handler ; Reset Handler
DCD NMI_Handler ; NMI Handler
...
DCD Timer1_IRQHandler ; IRQ14
DCD ADC_IRQHandler ; IRQ17
...
```

Директива DCD (Define Constant Data) в ассемблере ARM используется для определения константных данных и размещения их в памяти.

Файлы разработчика, например, main.c:

```
/* Обработка прерывания по Timer1 */
void Timer1_IRQHandler(void) {
    ...
}

/* Обработка прерывания по ADC */
void ADC_IRQHandler(void) {
    ...
}
```

Ассемблерные вставки

Использование ассемблера:

- Для оптимизации по скорости выполнения и размеру программы;
- Для прямого манипулирования регистрами;
- Для использования старого ассемблерного кода в новых проектах;
- Для специальных инструкций (WFI, BKP, SVC);
- Для учебных целей.

Ассемблерные вставки

```
/* Для IDE Keil uVision */
__asm void add(int x1, int x2, int x3)
{
    ADDS R0, R0, R1
    ADDS R0, R0, R2
    BX LR
}

int swap32(int i)
{
    int res;
    __asm {
        REVSH res, i
    }
    return res;
}

__asm("WFI"); /* Выполнение одной команды */
```


Стандартная библиотека и системные вызовы

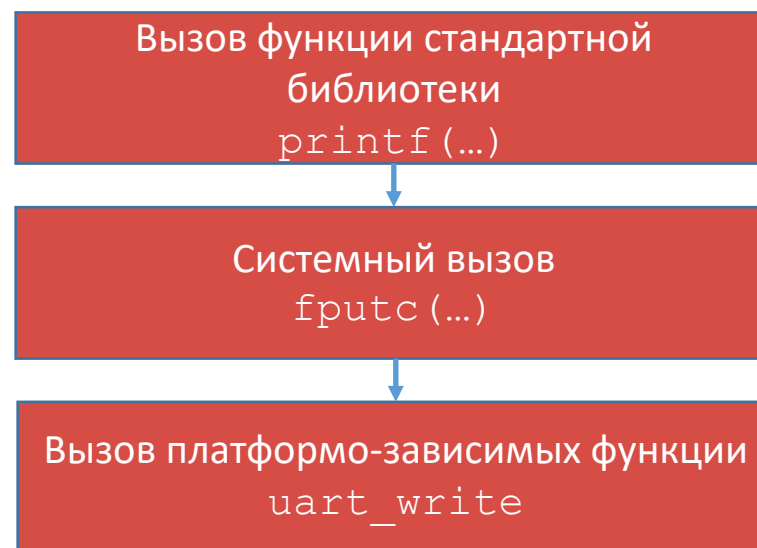
Для встраиваемых систем существует несколько стандартных библиотек C, которые разработаны с учетом ограниченных ресурсов этих систем. Например, [Newlib](#) и mbed C library (для RTOS и bare-metal), [uClibc-ng](#) и [musl](#) (для Embedded Linux).

Для библиотеки newlibc (Newlib), системные вызовы (sys calls) представляют собой интерфейс между библиотекой и целевой системой. Это низкоуровневые функции для выполнения операций, которые требуют взаимодействия с операционной системой, файловой системой, драйверами периферии и так далее. Это работа с файлами, ввод/вывод, управление процессами и т. д.

Системные вызовы в newlibc могут быть реализованы по-разному в зависимости от целевой платформы.

Основные системные вызовы включают в себя такие операции, как:

- `read` и `write` — для чтения и записи данных в файловые дескрипторы;
- `open` и `close` — для открытия и закрытия файлов;
- и так далее.



Перенаправление потоков ввода/вывода

Стандартная библиотека содержит функции ввода/вывода:

`printf`, `scanf`, `fopen`, `fprintf`, `fwrite`, `fread`, `fclose` и другие.

Функции `printf`, `scanf` используют дескрипторы по умолчанию:

`stdin` – стандартный ввод, `stdout` – стандартный вывод, `stderr` – поток ошибок.

На устройствах без ОС если только требуется печать и чтение из последовательного интерфейса, то достаточно переопределить функции `fputc` и `fgetc` для библиотеки `Newlibc`.

```
int fputc(int c, FILE * stream)
{
    uart_write(c);
    return c;
}
```

```
int fgetc(FILE * stream)
{
    char c = uart_read();
    return c;
}
```

Для разных компиляторов и стандартных библиотек порядок перенаправления может отличаться.

Результат компиляции: HCS08 (8-ми разрядный CISC)

Код на языке C

```
1. void func(void) {  
2.     return;  
3. }  
4.  
5. void main(void) {  
6.     char i;  
7.  
8.     i = 5;  
9.     while (i > 0) {  
10.         func();  
11.         i--;  
12.     }  
13.     for (;;) ;  
14. }
```

Результат компиляции для HCS08

```
...  
8:     char i;  
9:  
10:    i = 5;  
    LDA    #5  
    TSX  
    STA    ,X  
    L5:  
11:    while (i > 0)  
12:    {  
13:        func();  
    BSR    func  
14:        i--;  
    TSX  
    DEC    ,X  
    TST    ,X  
    BNE    L5  
    LC:  
15:    }  
16:    for (;;) ;  
    BRA    LC  
17: }
```

Результат компиляции: ARM Cortex-M3 (32-х разрядный RISC)

Код на языке C

```
1. void func(void) {  
2.     return;  
3. }  
4.  
5. void main(void) {  
6.     char i;  
7.  
8.     i = 5;  
9.     while (i > 0) {  
10.         func();  
11.         i--;  
12.     }  
13.     for (;;) ;  
14. }
```

Результат компиляции для ARM Cortex-M3

Адрес	Маш. код.	Ассемблер
8000136:	b580	push {r7, lr}
8000138:	b082	sub sp, #8
800013a:	af00	add r7, sp, #0
		char i;
		i = 5;
800013c:	2305	movs r3, #5
800013e:	71fb	strb r3, [r7, #7]
8000140:	e004	b.n 800014c <main+0x16>
		while (i > 0) {
		func();
8000142:	f7ff fff3	bl 800012c <func>
		i--;
8000146:	79fb	ldrb r3, [r7, #7]
8000148:	3b01	subs r3, #1
800014a:	71fb	strb r3, [r7, #7]
		while (i > 0)
800014c:	79fb	ldrb r3, [r7, #7]
800014e:	2b00	cmp r3, #0
8000150:	d1f7	bne.n 8000142 <main+0xc>
		}
		for (;;) ;
8000152:	e7fe	b.n 8000152 <main+0x1c>

Результат компиляции: RV32I (RISC-V, 32-х разрядный RISC)

Код на языке C

```
1. void func(void) {  
2.     return;  
3. }  
4.  
5. void main(void) {  
6.     char i;  
7.  
8.     i = 5;  
9.     while (i > 0) {  
10.         func();  
11.         i--;  
12.     }  
13.     for (;;) ;  
14. }
```

Результат компиляции для RISC-V

Адрес	Маш. код.	Ассемблер
1014a:	1101	addi sp,sp,-32
1014c:	ce06	sw ra,28(sp)
1014e:	cc22	sw s0,24(sp)
10150:	1000	addi s0,sp,32
		char i;
		i = 5;
10152:	4795	li a5,5
10154:	fef407a3	sb a5,-17(s0)
		while (i > 0)
10158:	a039	j 10166 <main+0x1c>
		{
		func();
1015a:	37cd	jal 1013c <func>
		i--;
1015c:	fef44783	lbu a5,-17(s0)
10160:	17fd	addi a5,a5,-1
10162:	fef407a3	sb a5,-17(s0)
		while (i > 0)
10166:	fef44783	lbu a5,-17(s0)
1016a:	fbe5	bnez a5,1015a <main+0x10>
		}
		for (;;) ;
1016c:	a001	j 1016c <main+0x22>

Образ программы

После компиляции набор инструментов создает образ программы (image).

Внутри образа программы находятся следующие компоненты:

- Таблица векторов прерываний (vector table);
- Код запуска (startup code);
- Константы;
- Функции среды исполнения C (C runtime library functions);
- Функции стандартной библиотеки C (C standard library functions);
- Прикладная программа (application code).

Формат образа программы может быть различным:

- **.hex (Intel HEX)** – широко используемый формат для хранения бинарного кода в шестнадцатеричном представлении. Файл состоит из строк, каждая из которых содержит адрес, данные и контрольную сумму.
- **.bin** – простой двоичный файл, содержащий «сырые» данные программы, без дополнительной информации (адресов, заголовков и т.д.).
- **.elf (Executable and Linkable Format)** – стандартный формат для исполняемых файлов, объектных файлов, библиотек и дампов памяти в UNIX-подобных системах. Включает как исполняемый код, так и отладочную информацию.
- **.srec (Motorola S-record)** – формат, похожий на Intel HEX, используемый для хранения бинарных данных в текстовом виде. Строки файла содержат адрес, данные и контрольную сумму.

Заключение

- Понимание модели памяти языка C позволяет писать оптимальный код.
- Язык C позволяет напрямую обращаться к памяти через указатели, но не позволяет обращаться к регистрам процессора.
- Применение для встраиваемых систем:
 - Обращение к регистрам специальных функций через указатели;
 - Стартовый код;
 - Обработка прерываний;
 - Ассемблерные вставки;
 - Системные вызовы.