

# 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:** Ядро ARM Cortex-M3. Микроконтроллер

**Лекция 5:** Этапы разработки микропроцессорных систем

**Лекция 6:** Разработка и отладка программ для встраиваемых систем

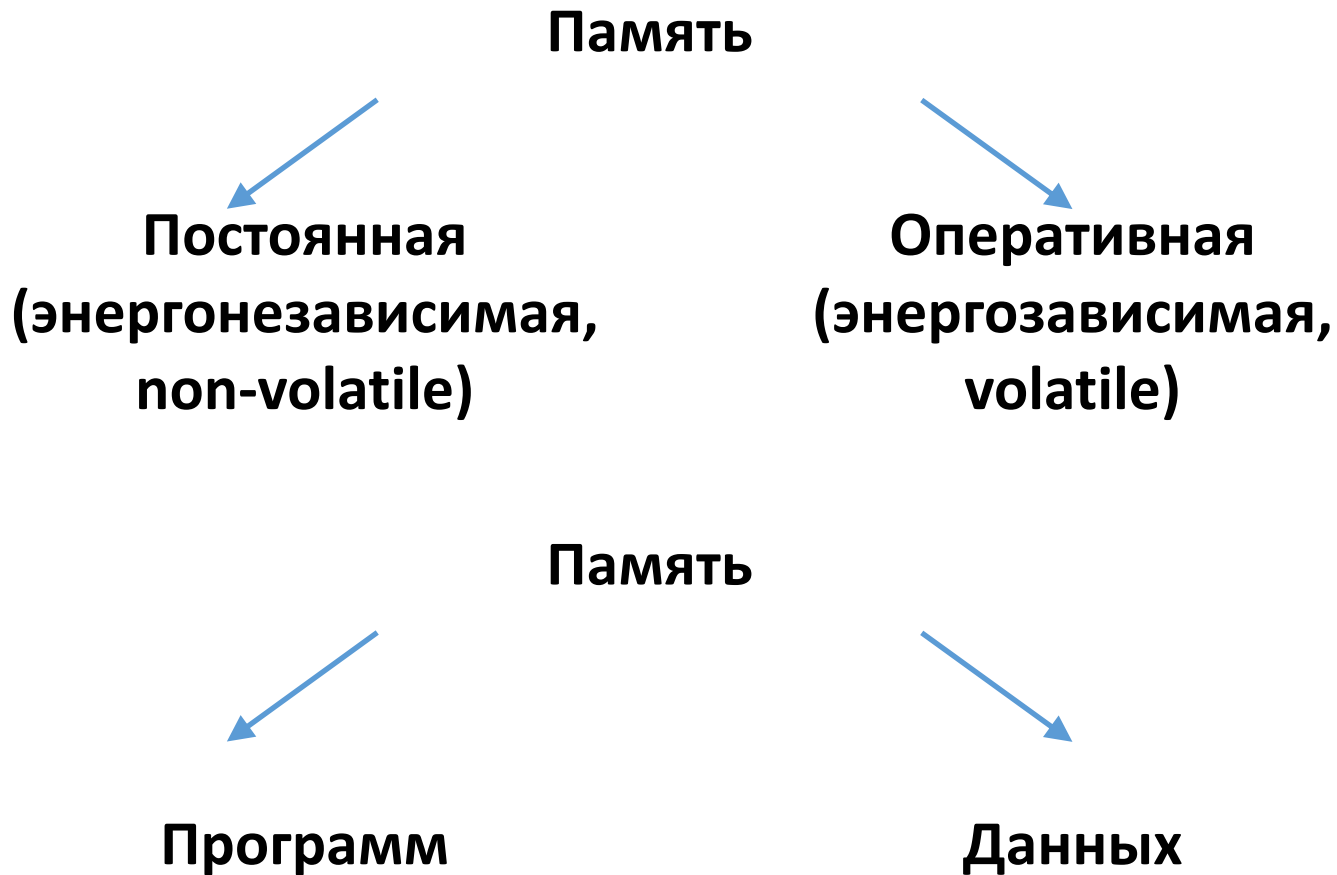
**Лекция 7:** Архитектура программного обеспечения

**Лекция 8:** Периферийные модули: Timer, DMA, ADC, DAC

**Лекция 9:** Периферийные модули: 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 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);
```

# Модель памяти в языке C

ОЗУ

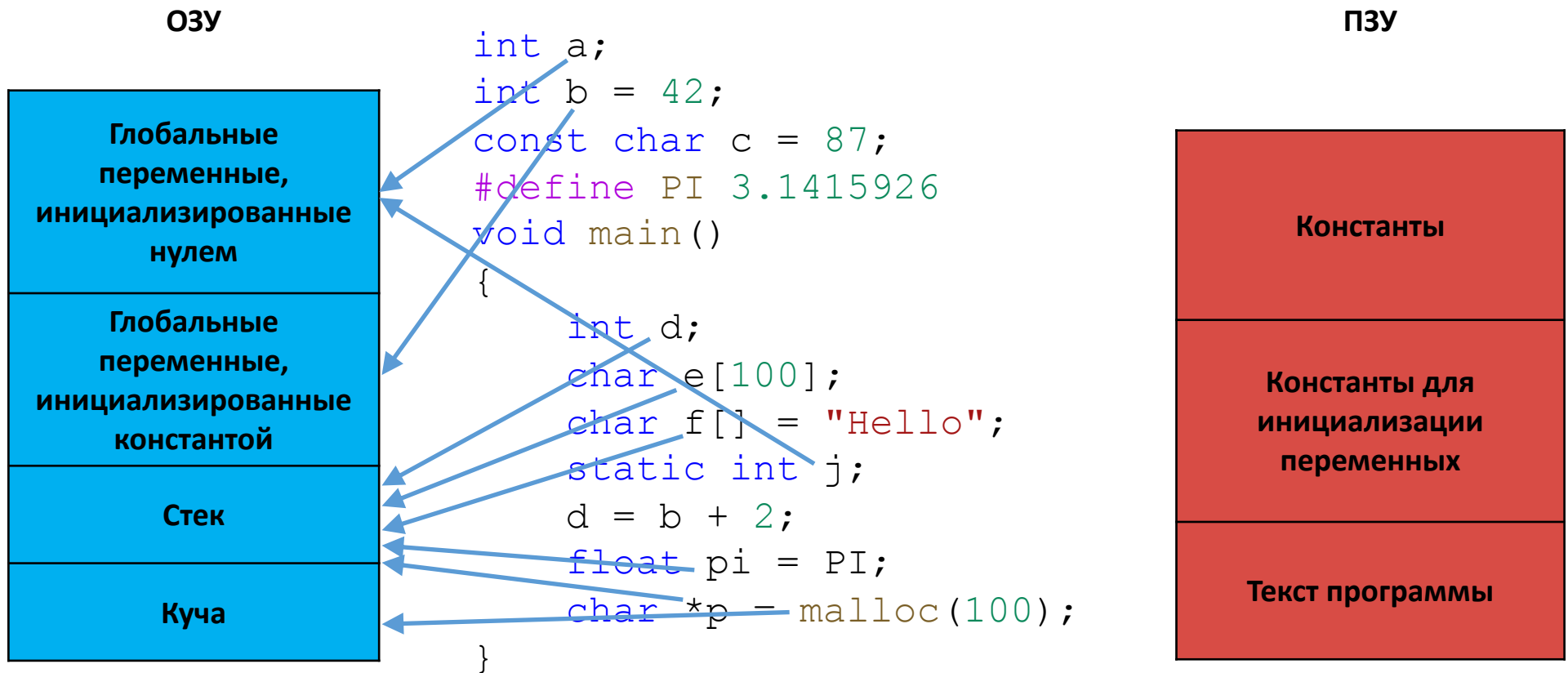
Глобальные переменные, инициализированные нулем
Глобальные переменные, инициализированные константой
Стек
Куча

```
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
Текст программы

# Модель памяти в языке C



# Модель памяти в языке C

ОЗУ

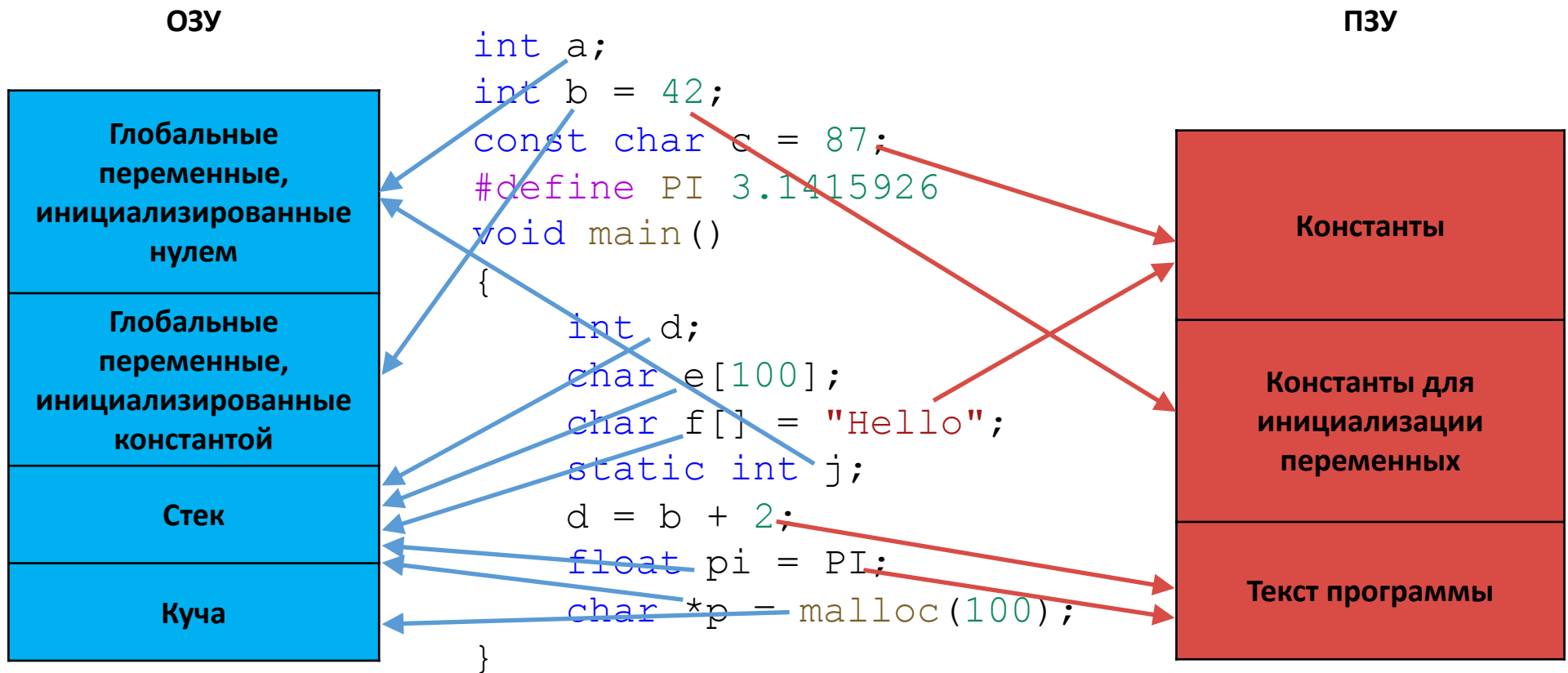


```
int a;  
int b = 42;  
const char c = 87;  
#define PI 3.1415926  
void main()  
{  
    int d;  
    char e[100];  
    char f[] = "Hello";  
    static int j;  
    d = b + 2;  
    float pi = PI;  
    char *p = malloc(100);  
}
```

ПЗУ



# Модель памяти в языке C





- Обращение к регистрам специальных функций периферийных модулей;
- Обработка прерываний;
- Ассемблерные вставки.

# 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 */
```

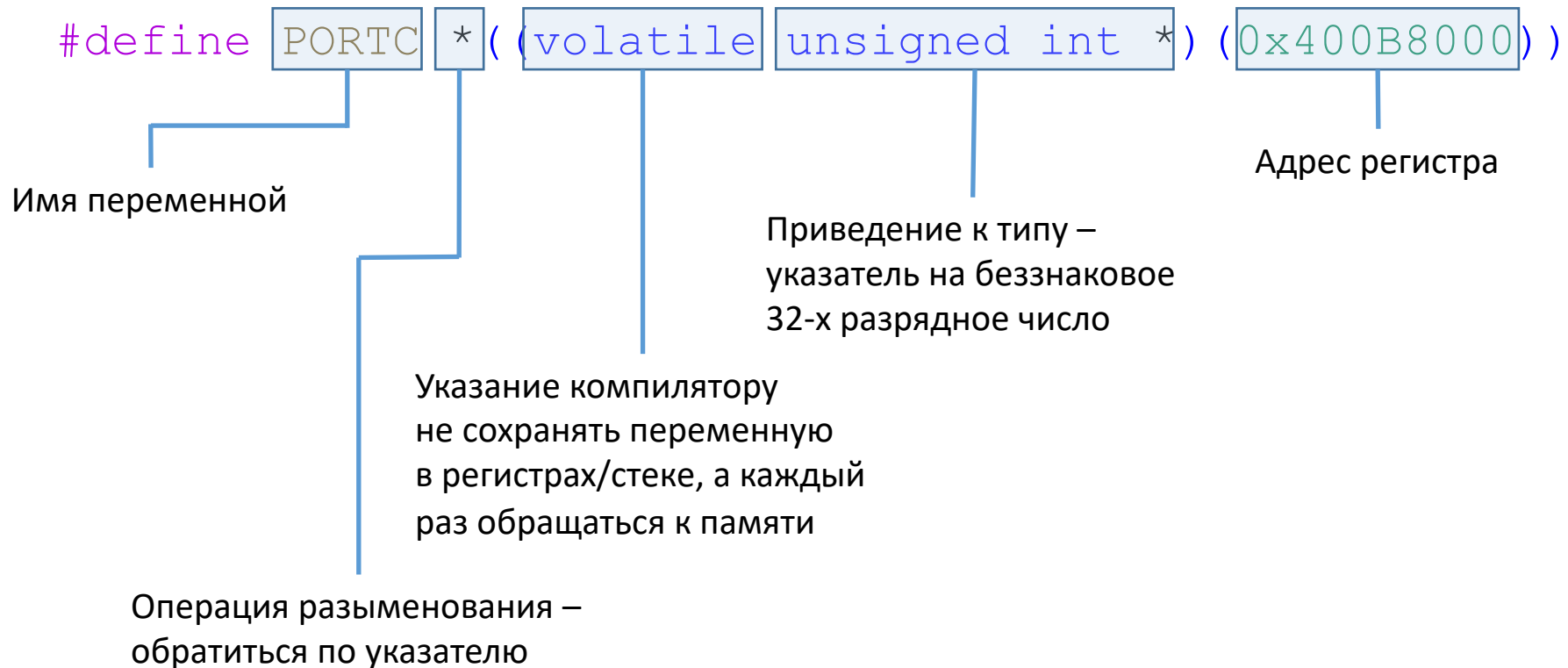
### 3. Язык C + библиотека (например, Standard Peripheral Library для K1986BE92QI)

```
#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);
    }
}
```

# Обработка прерываний в Cortex-M3

---

Файл `startup_MDR32F9Qx.s`:

```
...  
DCD Timer1_IRQHandler ; IRQ14  
DCD ADC_IRQHandler ; IRQ17  
...
```

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 */
```

# Результат компиляции: HCS08 (8-ми разрядный CISC)

Код на языке C

```
1. void func(void)
2. {
3.     return;
4. }
5.
6. void main(void)
7. {
8.     char i;
9.
10.    i = 5;
11.    while (i > 0)
12.    {
13.        func();
14.        i--;
15.    }
16.    for (;;) ;
17. }
```

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. {
3.     return;
4. }
5.
6. void main(void)
7. {
8.     char i;
9.
10.    i = 5;
11.    while (i > 0)
12.    {
13.        func();
14.        i--;
15.    }
16.    for (;;)
17. }
```

Результат компиляции для ARM Cortex-M3

```
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

Результат компиляции для RISC-V

```
1. void func(void)
2. {
3.     return;
4. }
5.
6. void main(void)
7. {
8.     char i;
9.
10.    i = 5;
11.    while (i > 0)
12.    {
13.        func();
14.        i--;
15.    }
16.    for (;;)
17. }
```

```
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);
- Вектор сброса/код запуска (reset handler/startup code);
- Код запуска программы на C (C startup code);
- Прикладная программа (application code);
- Функции среды исполнения C (C runtime library functions);
- Другие данные.

# Перенаправление потоков ввода/вывода

---

Стандартная библиотека содержит функции ввода/вывода:

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

Функция `fopen` возвращает файловый дескриптор – `File* fd`.

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

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

На устройствах без ОС требуется указать куда осуществляется ввод/вывод. Для этого следует переопределить функции `fputc` и `fgetc`.

```
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.

# 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;
  - Обработка прерываний;
  - Ассемблерные вставки.