



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

ЛЕКЦИОННЫЕ МАТЕРИАЛЫ

по дисциплине

Цифровые устройства и микропроцессоры

Часть 2 (6 семестр)

Учебный план на весну 2022 г.

Лекций:	16
Практических занятий:	8
Лабораторных работ:	3
Курсовая работа (для групп РССО)	
Экзамен	

Рекомендуемая литература

Основная

Новожилов О.П. Основы микропроцессорной техники. В 2 т.— М.: ИП Радиософт, 2007. Т. 1. — 432 с.

Микушин А.В., Сажнев А.М., Сединин В.И. Цифровые устройства и микропроцессоры. — СПб.: БВХ-Петербург, 2010. — 832 с.

Магда Ю.С. Современные микроконтроллеры. Архитектура, програм-мирование, разработка устройств. — М.: ДМК, 2010. — 228 с.

Дополнительная

Джозеф Ю. Ядро Cortex-M3 компании ARM. Полное руководство / Джозеф Ю; пер. с англ. А.В.Евстифеева. — М.: Додэка-XXI, 2012. — 552 с.

Trevor Martin. The Insider's Guide to the STM32 ARM-based Microcontroller // Hitex (UK) Ltd (www.hitex.com), 2009. Перевод: Ознакомительное руководство по ARM-микроконтроллерам Cortex-M3 [Электронный ресурс]. — Режим доступа: http://www.gaw.ru/html.cgi/txt/doc/micros/arm/cortex_arh/index.htm.

Carminе Noviello. Mastering STM32 [Электронный ресурс], 2018.
<https://leanpub.com/mastering-stm32>. Перевод: Освоение STM32, 2021.
https://vk.com/wall-58310134_11903.

Иванов Р. Лидер по производительности среди ядер Cortex-M4 – STM32F4xx. — Новости электроники, 2012, № 2, с. 17-22.

Керниган Б., Ритчи Д. Язык программирования C. : Пер. с англ. — М.: Издательский дом "Вильямс", 2009. — 304 с.

Методическая

Богаченков А.Н. Цифровые устройства и микропроцессоры [Электронный ресурс]: Методические указания по выполнению лабораторных работ. — М.: МИРЭА – Российский технологический университет, 2019.

Богаченков А.Н. Цифровые устройства и микропроцессоры [Электронный ресурс]: Методические указания по выполнению курсовой работы. — М.: МИРЭА – Российский технологический университет, 2022.

Введение в микропроцессорную технику

Микропроцессор — цифровое устройство, выполненное в виде микросхемы и производящее по загруженной в него программе различные операции (арифметические, логические, ввод-вывод и др.)

Микроконтроллер — содержит процессор (обычно с ограниченными вычислительными возможностями) и различные периферийные модули. Основное предназначение — выполнение задач управления объектами.



В настоящем курсе сделан упор на применение процессоров во встраиваемых системах (примеры подобных устройств приведены выше). Наиболее универсальным, востребованным и распространенным вычислительным ядром является ARM Cortex, ставшее современным промышленным стандартом. 32-разрядный микроконтроллер, по стоимости практически сравнявшийся с применяемыми ранее 8- и 16-разрядными устройствами, обладает по сравнению с ними значительно большими вычислительными и функциональными возможностями.

Аббревиатура ARM (**A**dvanced **R**ISC **M**achine, RISC — Reduced Instruction Set Computer) является как названием компании — разработчика вычислительного ядра, так и обозначением микропроцессорной архитектуры. На 32- и 64-разрядных процессорных ядрах от компании ARM Limited базируется практически вся современная мобильная аппаратура. Семейство Cortex на сегодняшний момент является самой последней модификацией ядра ARM. Существуют 3 профиля этого ядра:

Профиль **Cortex-A** (ARMv7-A) — прикладные процессоры, предназначенные для поддержки сложных приложений во встраиваемых операционных системах (Linux, Windows, Android, iOS и др.). Основные потребители — современные смартфоны, планшеты.

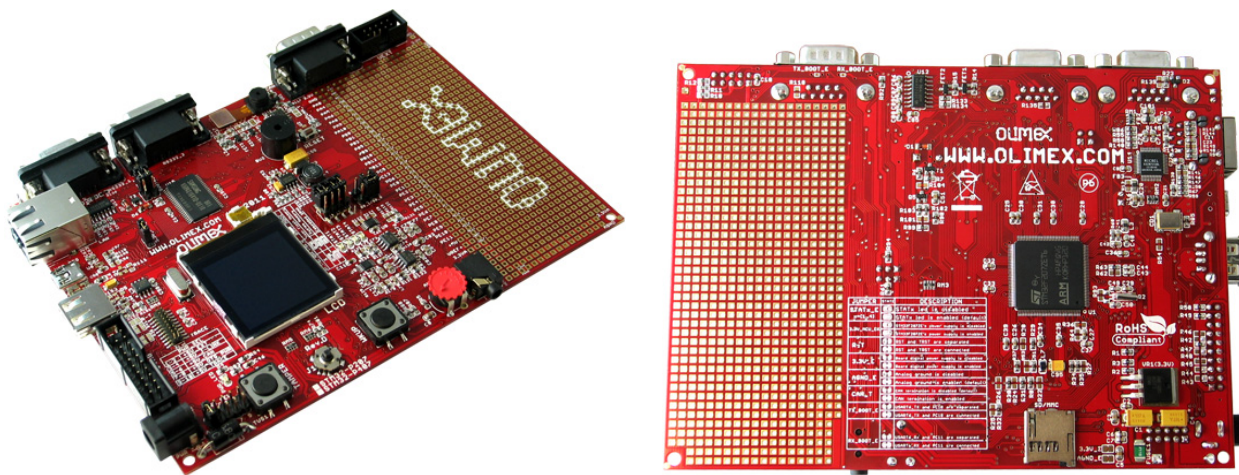
Профиль **Cortex-R** (ARMv7-R) — высокопроизводительные процессоры, предназначенные для создания устройств, работающих в условиях жёсткого реального времени (автомобили, накопители, технологические установки и т.п.).

Профиль **Cortex-M** (ARMv7-M) — процессоры для бюджетных приложений, в которых помимо высокой производительности критичными являются стоимость, энергопотребление, время реакции, простота использования. Профиль, в свою очередь, имеет модификации: M0, M1, M3, M4, M7, различающиеся дополнительными наборами команд и производительностью.

Отличительными признаками ядра ARM являются наилучшие соотношения параметров: производительность / энергопотребление, производительность / стоимость.

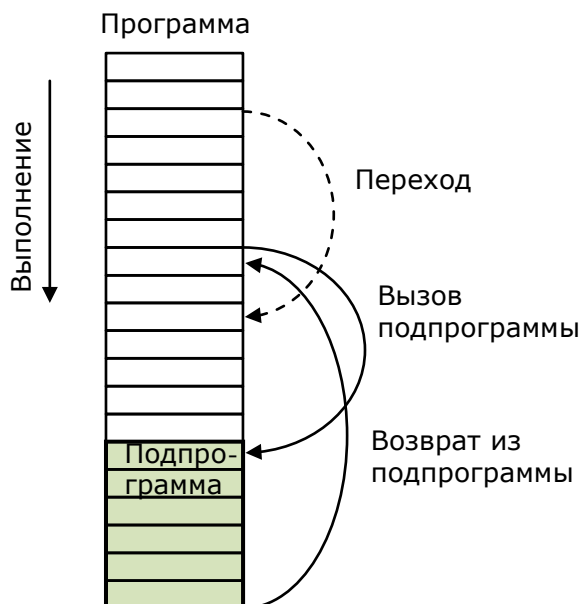
Освоение микропроцессорной техники в настоящем курсе базируется на платформе ARM Cortex-M4. Основные производители микроконтроллеров с подобным ядром: STMicroelectronics (выпускает наибольшее количество различных модификаций), NXP, Texas Instruments, Microchip (Atmel), Cypress, Silicon Laboratories, GigaDevice, Holtek, Infineon, Nuvoton, Toshiba, Renesas и др. (есть также и российские фирмы).

В лабораторном практикуме используется отладочная плата, построенная на процессоре фирмы STMicroelectronics (микросхема имеет 144 вывода, размещена на нижней стороне платы):



Введение в разработку и отладку программ

Программа, исполняемая процессором — последовательность команд, размещенная в памяти. Команды представляются в виде машинного двоичного кода различной разрядности. Выборка, дешифрация, исполнение команд осуществляется как последовательно, так и с использованием переходов-ветвлений.



Основные типы команд:

- чтение из памяти/регистра, ввод из порта (направление передачи — в процессор);
- запись в память/регистр, вывод в порт (направление передачи — из процессора);
- арифметические, логические и некоторые специальные операции с многоразрядными данными и отдельными битами — обработка данных;
- проверка условий ($=$, $<$, $>$, состояние бит и др.);
- ветвления: переходы и вызовы подпрограмм, могут быть безусловными и условными.

Пользовательская программа, содержащая подобные низкоуровневые операции, использует язык ассемблера (язык низкого уровня). Почти каждая команда ассемблера эквивалентна команде на машинном языке процессора.

Языки высокого уровня используют более абстрактное представление структур данных и операций над ними.

При наличии исходного текста на одном из языков программирования получение исполняемого кода требует как минимум следующих двух стадий:

Компиляция. Перевод конструкций языка в машинный код, часто с промежуточным преобразованием в язык ассемблера. Для каждого исходного модуля создается свой так называемый объектный модуль. В таком модуле помимо кода содержится таблица перекрестных ссылок на другие модули и библиотеки.

Компоновка. Отдельная программа — компоновщик (другие названия: редактор связей, "линковщик", "линкер") — собирает объектные модули, в том числе из стандартных библиотек, в единый исполняемый модуль, заменяя перекрестные ссылки реальными адресами.

На данных этапах выявляются и устраняются синтаксические (формальные) ошибки.

Для поиска логических ошибок (обычно связаны с "человеческим" фактором) используются средства отладки. Наиболее простое — симуляция, когда компьютерная программа моделирует работу реального процессора. При этом возможно пошаговое (пооператорное или покомандное) выполнение программы, остановка в заданных точках, просмотр содержимого переменных, регистров, памяти и др. Главные недостатки — сложность или даже невозможность моделирования других модулей системы, кроме вычислительного

ядра, сложность имитации внешних воздействий/сигналов, низкая скорость выполнения программы и другие неудобства. Данный вид отладки используется в ряде проектов при проведении лабораторного практикума. Принципы отладки реальных устройств будут рассмотрены позднее.

Фрагмент текста программы на языке ассемблера

```
label
LDRH  R3,[R1],#0x02    //Чтение из памяти с адресом в R1 в регистр R3 с автоувеличением R1
STR   R3,[R0,#0x0C]    //Пересылка из регистра R3 по адресу (R0 + 0x0C)
SUBS  R2,R2,#1         //Уменьшение содержимого регистра R2 на 1 с установкой признаков
BNE   label            //Если не 0 (в предыд.операции), переход на команду с меткой label
BX    LR               //Возврат в вызывающую программу по адресу в регистре связи LR
```

Фрагмент листинга после компиляции и дизассемблирования

Адрес	Машинный код	Команда	Операнды
0x08000100	F8313B02	LDRH	r3,[r1],#0x02
0x08000104	60C3	STR	r3,[r0,#0x0C]
0x08000106	1E52	SUBS	r2,r2,#1
0x08000108	D1FA	BNE	(0x08000100)
0x0800010A	4770	BX	lr

Сравнительная оценка языков программирования

Самостоятельно предлагается заполнить следующую таблицу:

Характеристика	Язык ассемблера	Язык высокого уровня
Трудоемкость программирования		
Наглядность представления данных		
Доступность ресурсов процессора		
Контроль за распределением памяти		
Автовывявление ошибок		
Размер откомпилированной программы		
Быстродействие		
Зависимость от типа процессора		

Введение в программирование микропроцессорных систем**Форматы чисел в микроконтроллерах и сигнальных процессорах**

Десятичный целочисленный			Десятичный дробный 1.15	16-ричный	Двоичный
Без знака	Со знаком, 1 байт	Со знаком, 2 байта			
0	0	0	0	0	0000
1	1	1	0.0000305	1	0001
2	2	2	0.0000610	2	0010
3	3	3	0.0000916	3	0011
4	4	4	0.0001221	4	0100
5	5	5	0.0001526	5	0101
6	6	6	0.0001831	6	0110
7	7	7	0.0002136	7	0111
8	8	8	0.0002441	8	1000
9	9	9	0.0002747	9	1001
10	10	10	0.0003052	A	1010
11	11	11	0.0003357	B	1011
12	12	12	0.0003662	C	1100
13	13	13	0.0003967	D	1101
14	14	14	0.0004272	E	1110
15	15	15	0.0004578	F	1111
16	16	16	0.0004883	10	1 0000
126	126	126	0.0038452	7E	0111 1110
127	127	127	0.0038757	7F	0111 1111
128	-128	128	0.0039063	80	1000 0000
129	-127	129	0.0039368	81	1000 0001
254	-2	254	0.0077515	FE	1111 1110
255	-1	255	0.0077820	FF	1111 1111
256		256	0.0078125	100	0000 0001 0000 0000
257		257	0.0078430	101	0000 0001 0000 0001
4096		4096	0.1250000	1000	0001 0000 0000 0000
8192		8192	0.2500000	2000	0010 0000 0000 0000
16384		16384	0.5000000	4000	0100 0000 0000 0000
32766		32766	0.9999390	7FFE	0111 1111 1111 1110
32767		32767	0.9999695	7FFF	0111 1111 1111 1111
32768		-32768	-1	8000	1000 0000 0000 0000
32769		-32767	-0.9999695	8001	1000 0000 0000 0001
49152		-16384	-0.5000000	C000	1100 0000 0000 0000
65534		-2	-0.0000610	FFFE	1111 1111 1111 1110
65535		-1	-0.0000305	FFFF	1111 1111 1111 1111

Основные типы данных в языке Си

Тип данных, диапазон	Обозначение	
	Стандарт C/C++	Альтернат. варианты
8-разрядное целое без знака 0...255	unsigned char	uint8_t, uint8, u8, uchar, byte и др.
8-разрядное целое со знаком -128...127	signed char, char	int8_t, int8, _int8, s8, i8, sbyte
16-разрядное целое без знака 0...65535 ($0...2^{16}-1$)	unsigned short	uint16_t, uint16, u16, USHORT, WORD, WCHAR
16-разрядное целое со знаком -32768...32767 ($-2^{15}...2^{15}-1$)	signed short, short, short int	int16_t, int16, _int16, s16, i16
32-разрядное целое без знака 0...4294967295 ($0...2^{32}-1$)	unsigned long	uint32_t, uint32, u32, ULONG, DWORD, WCHAR32
32-разрядное целое со знаком -2147483648...2147483647	signed long, long	int32_t, int32, _int32, s32, i32, longint
64-разрядное целое со знаком $-2^{63}...2^{63}-1$	long long	int64_t, int64, _int64
Целое со знаком	int	integer
Целое без знака	size_t	
32-разрядное с плавающей точкой $-3.4 \times 10^{38}...+3.4 \times 10^{38}$	float	float32_t, float32, single
64-разрядное с плавающей точкой $\pm 5.0 \times 10^{-324}... \pm 1.7 \times 10^{308}$	double	float64_t, float64
16-разрядное дробное в формате 1.15 -1.0...+1.0 с дискретностью изм. 2^{-15}	–	q15_t
32-разрядное дробное в формате 1.31 -1.0...+1.0 с дискретностью изм. 2^{-31}	–	q31_t
Булев [false, true] или [0, 1] или [0, не 0]	bool	Boolean, BOOL

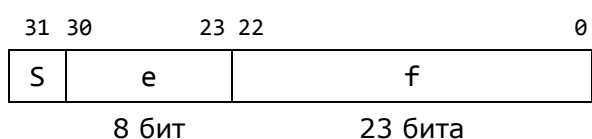
Разрядность типа данных `int` зависит от разрядности процессора и компилятора. Для ядра ARM Cortex-M тип `int` является 32-разрядным и эквивалентен типу `long`. Для 8- и 16-разрядных процессоров тип `int`, как правило, имеет размер 16 бит.

Обычно ключевое слово `signed` опускают, так как по умолчанию целые числа считаются знаковыми.

Дробные форматы в стандартном языке Си отсутствуют, аппаратно поддерживаются только в сигнальных процессорах, в других случаях объявляются как целочисленные (`q15_t` соответствует `short`, `q31_t` – `long`). С такими дробными данными процессор работает как с целыми, а особенности вычислительных операций и приведение к диапазону $-1.0...+1.0$ реализуются в библиотечных подпрограммах.

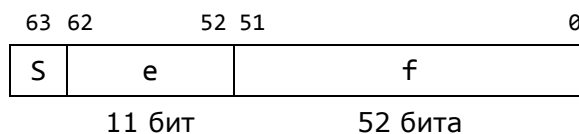
Число с плавающей точкой (вещественное) по стандарту IEEE 754 определяется в виде $C = (-1)^S 2^e 1.f$, где S – знак числа, e – смещенный порядок (целое положительное число), 1 – целая часть мантииссы (присутствует неявно), f – мантиисса.

Одинарная точность:



$e = 1...254$ — соответствует реальному диапазону порядков $-126...127$

Двойная точность:



$e = 1...2046$ — соответствует реальному диапазону порядков $-1022...1023$

При операциях с плавающей точкой результат всегда является приближенным. *Не рекомендуется* использование вещественных данных при целочисленных операциях.

Булев тип `bool` физически может быть представлен одним битом, однако в большинстве случаев компиляторы определяют такой тип как однобайтовое или просто целое число (доступ к отдельным битам обычно более медленный, чем к словам).

В языке имеется также стандартный тип `void`, используемый для обозначения пустого списка аргументов, отсутствия возвращаемого функцией значения, а также указателя на произвольный тип, например:

```
bool func1 (void);
bool func2 (void* argument);
```

Типы данных могут переопределяться в пользовательских проектах, например, часто используются такие обозначения, как `int32`, `uint32`, `s32`, `u8`, `u16`, `u32` и т.п., например, типы данных, используемые в программах для процессоров с ядром ARM, объявлены в заголовочных файлах следующим образом:

```
typedef signed short int16_t;
typedef unsigned int uint32_t;
```

Непосредственно ядром Cortex-M поддерживаются операции с 8-/16-/32-разрядными данными, а также с 64-разрядными целочисленными операндами в командах умножения-накопления (как правило, эти операции выполняются за 1 такт). Блок обработки с плавающей точкой в Cortex-M4 оперирует только с 32-разрядными данными (`float`) и не во всех модификациях процессоров имеется в наличии. Для других форматов используются библиотечные подпрограммы.

Примеры записи констант:

<code>123, -45,</code>	– целочисленная со знаком (обычно имеет тип <code>int</code>)
<code>678u</code>	– целочисленная без знака
<code>0x9ABC</code>	– целочисленная в 16-ричной форме
<code>2.34, -.005f, 1.9e-4</code>	– вещественная (суффикс <code>f</code> для типа <code>float</code>)
<code>'a', '\n'</code>	– символьная (обычно типа <code>char</code>)
<code>"good"</code>	– строковая
<code>enum my_mode { constA, constB, constE }</code>	– перечислимый тип

Примеры объявления массивов:

```
int ab[100];           // Массив из 100 элементов типа int
uint8_t cd[4][16];     // Двумерный массив 8-разрядных элементов
```

Пример объявления структуры (объединения разнородных данных в один объект):

```
struct my_object
{ int N;
  char name[16];
  float param;
};
```

Примеры объявления указателей:

```
float *ef;             // Указатель на элементы типа float
my_object *gh;         // Указатель на объект (например, структуру)
Void *pointer;         /* Указатель на произвольный объект — при операциях
                        записи-чтения необходимо приведение к указателю
                        на конкретный объект */
```

Символам всегда соответствуют числовые коды — одно- или двухбайтовые. Распространенным стандартом является таблица ASCII, ее фрагменты представлены ниже (для символов кириллицы используется 8-битовая кодировка Windows-1251).

	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xA	0xB	0xC	0xD	0xE	0xF
0x00 (0)								\a		\t	\n			\r		
0x10 (16)																
0x20 (32)		!	“	#	\$	%	&	‘	()	*	+	,	-	.	/
0x30 (48)	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0x40 (64)	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0x50 (80)	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
0x60 (96)	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0x70 (112)	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL
...																
0xC0	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
0xD0	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
0xE0	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
0xF0	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я

Пример записи строки "МИРЭА" в виде 6-элементного массива (с завершающим 0):

```
char name[] = {0xCC, 0xC8, 0xD0, 0xDD, 0xC0, 0x00};
```

Приведенная строка эквивалентна следующему строковому литералу:

```
char* name = "МИРЭА";
```

В обоих примерах имя name является указателем (адресом) на начало последовательности кодов символов.

В современных операционных системах общепринятым является двухбайтовое представление символов, например, в кодировке Unicode:

0x0000...0x007F – соответствует символам с кодами 0x00...0x7F таблицы ASCII;

0x0410...0x044F – символы А...Я (без Ё), а...я (без ё);

0x2200...0x22FF – символы математических операций и т.д.

Для работы с однобайтовыми символами обычно используется тип char, с двухбайтовыми — wchar_t (является стандартным типом в C++, соответствует unsigned short).

Приведение (преобразование) типов данных

Арифметические и логические операции всегда выполняются с данными одного типа. Если в операции участвуют различные типы, то компилятор осуществляет их неявное преобразование к более сложному типу (к большей разрядности, целого к плавающей точке и др.). Присвоение значений одного типа другому также сопровождается неявным приведением к конечному типу, причем в ряде случаев, например, если двухбайтовое значение присваивается однобайтовой переменной или вещественное число преобразуется в целое, возможна потеря части данных. В подобных случаях компилятор может выдавать предупреждения и сообщения об ошибках.

Всегда предпочтительно явное приведение типов при написании операторов программы. В языке Си новый тип указывается в скобках перед переменной или выражением:

```
int x = 1;
float result1 = x/2;           // Результат целочисленного деления равен 0 !
float result2 = (float)x/2;    // Здесь результат равен 0.5
```

Пример преобразования типа указателя:

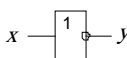
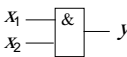
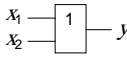
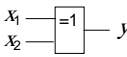
```
int8_t * array;           // Объявление указателя на байт
y = *array                // Чтение байта из памяти
y = *(int16_t*)array;     // Чтение двухбайтового слова из памяти
```

Основные операции в языке Си

Операция	Направление приоритета	Назначение
::		Разрешение области видимости
() [] . -> ++ --	→	Группировка, индекс массива, прямое и косвенное членство, инкремент и декремент постфиксные
! ~ + - ++ -- & * (mun) sizeof	←	Логическое отрицание, инверсия, унарный +, унарный -, инкремент и декремент префиксные, адрес, разыменование, приведение типа, размер в байтах
. * -> *	→	Выбор члена класса
* / %	→	Умножение, деление, остаток от деления
+ -	→	Сложение, вычитание
<< >>	→	Сдвиг влево, сдвиг вправо
< <= >= >	→	Меньше, меньше или равно, больше или равно, больше
== !=	→	Равно, не равно
&	→	Поразрядное "И" (лог. умножение, конъюнкция)
^	→	Поразрядное исключающее "ИЛИ" (неравнозначность)
	→	Поразрядное "ИЛИ" (лог. сложение, дизъюнкция)
&&	→	Логическое "И"
	→	Логическое "ИЛИ"
?:	←	Условная операция
= *= /= %= += -= &= ^= = <<= >>=	←	Простое присваивание и присваивание с операцией
,	→	Последовательное вычисление

В верхней строке таблицы приведены операции с наивысшим приоритетом, далее вниз по строкам приоритетность убывает. Если в выражении несколько операций имеют один и тот же уровень приоритета, они выполняются в порядке, указанном направлением стрелки в таблице (слева направо или справа налево).

Логические поразрядные (побитовые) операции

Функция	Математич. запись	Таблица истинности	Логический элемент	Команда или оператор	Пример	Назначение															
Инверсия , операция "НЕ"	$y = \overline{x}$	<table><tr><td>x</td><td>y</td></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	x	y	0	1	1	0		ARM: MVN Язык Си: ~	$\begin{array}{r} 0011\ 1100 \\ \hline 1100\ 0011 \end{array}$	Инверсия всех разрядов									
x	y																				
0	1																				
1	0																				
Логическое умножение , конъюнкция, операция "И"	$y = x_1 \wedge x_2$ или $y = x_1 \& x_2$	<table><tr><td>x_1</td><td>x_2</td><td>y</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x_1	x_2	y	0	0	0	0	1	0	1	0	0	1	1	1		ARM: AND Язык Си: &	$\begin{array}{r} 0011\ 1100 \\ \wedge\ 1100\ 1100 \\ \hline 0000\ 1100 \end{array}$	Обнуление отдельных разрядов (маскирование)
x_1	x_2	y																			
0	0	0																			
0	1	0																			
1	0	0																			
1	1	1																			
Логическое сложение , дизъюнкция, операция "ИЛИ"	$y = x_1 \vee x_2$	<table><tr><td>x_1</td><td>x_2</td><td>y</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x_1	x_2	y	0	0	0	0	1	1	1	0	1	1	1	1		ARM: ORR Язык Си: 	$\begin{array}{r} 0011\ 1100 \\ \vee\ 1100\ 1100 \\ \hline 1111\ 1100 \end{array}$	Установка в состояние лог. "1" отдельных разрядов
x_1	x_2	y																			
0	0	0																			
0	1	1																			
1	0	1																			
1	1	1																			
Неравнозначность , сложение по модулю 2, операция "исключающее ИЛИ"	$y = x_1 \oplus x_2$ или $y = x_1 \nabla x_2$	<table><tr><td>x_1</td><td>x_2</td><td>y</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x_1	x_2	y	0	0	0	0	1	1	1	0	1	1	1	0		ARM: EOR Язык Си: ^	$\begin{array}{r} 0011\ 1100 \\ \oplus\ 1100\ 1100 \\ \hline 1111\ 0000 \end{array}$	1. Сравнение кодов. 2. Инверсия отдельных разрядов
x_1	x_2	y																			
0	0	0																			
0	1	1																			
1	0	1																			
1	1	0																			

Условный оператор

```
if (условие) оператор 1;
else оператор 2;
```

При выполнении условия выполняется оператор 1, иначе — оператор 2. Ветвь else может отсутствовать.

Условный оператор

```
переменная = условие ? выражение 1 : выражение 2;
```

При выполнении условия переменной присваивается значение выражения 1, иначе — значение выражения 2.

Оператор-переключатель

```
switch (выражение)
{
  case константа 1: операторы 1; break;
  case константа 2: операторы 2; break;
  ...
  case константа N: операторы N; break;
  default: операторы;
}
```

Используется для выбора одного из нескольких вариантов ветвления в зависимости от того, какой константе равно значение выражения. Блок default выполняется, если не найдено ни одного соответствия в ветвях case (допускается его отсутствие).

Оператор цикла с предусловием

while (*условие*) *оператор-тело*;

Вычисления оператора (тела цикла) повторяются, пока условие является истинным (проверяется перед каждой итерацией). Если условие не выполняется изначально, цикл не выполняется ни разу.

Оператор цикла с постусловием

do *оператор-тело* **while** (*условие*);

Оператор (тело цикла) выполняется однократно всегда, затем вычисления повторяются, пока условие является истинным.

Оператор пошагового цикла

for (*нач.выражение*; *условие*; *выражение-модификация*)
оператор-тело;

Работает аналогично **while** — пока выполняется условие. Позволяет дополнительно использовать выражения, например, для задания точного числа повторений. Начальное выражение вычисляется один раз перед входом в цикл, условие — перед каждым началом выполнения оператора-тела цикла, выражение-модификация — каждый раз по завершении оператора-тела цикла,

Оператор пропуска тела цикла

continue;

Осуществляет пропуск оставшейся части тела цикла и переходит к следующей итерации цикла.

Оператор выхода из цикла / блока

break;

Осуществляет принудительный выход из циклов **while**, **do-while**, **for**, а также выход из ветвей переключателя **switch**.

Оператор возврата из подпрограммы

return; или **return** (*выражение*);

Завершает выполнение подпрограммы (функции) и передает управление оператору, следующему после оператора вызова подпрограммы. Выражение присутствует, если функция возвращает значение.

Оператор перехода

goto *метка*;

Передаёт управление оператору, помеченному именем *метка*:

метка: *оператор*;

Переход осуществляется только внутри функции. Удобен, например, для принудительного выхода из вложенных циклов, перехода в определенную точку из разных мест. Но оператор не рекомендуется для использования в программах. Он нарушает принципы структурного и модульного программирования, является потенциальным источником ошибок при наличии локальных переменных.

Оператор определения размера типа данных или объекта

`sizeof` (объект);

Возвращает длину в байтах типа данных, переменной или другого объекта, указанного в скобках. Тип возвращаемого значения: `size_t`. Примеры использования:

```
x = sizeof (float);           // Возвращаемое значение = 4

int16_t buffer[30];          // Объект: массив из 30 двухбайтовых элементов
r = sizeof (buffer);          // Возвращаемое значение = 60
```

Некоторые стандартные функции библиотеки языка Си

Для использования функций в программе необходимо подключение заголовочных файлов, в которых эти функции объявлены, например:

```
#include <math.h>
```

Функция `printf`

Используется для форматированного вывода текстовой информации на стандартное устройство вывода, в зависимости от устройства это может быть дисплей, последовательный порт, накопитель (для микроконтроллеров по умолчанию вывод осуществляется в последовательный порт). Прототип функции (определена в `stdio.h`):

`printf` (строка формата, список вывода)

Строка формата помимо произвольных выводимых символов содержит спецификации формата, используемые для представления данных. В упрощенном виде спецификация имеет вид:

%ширина_поля.точность.символ-спецификатор

Ширина поля и точность могут быть опущены (заданы по умолчанию). Некоторые символы-спецификаторы:

d	Знаковое десятичное целое
u	Беззнаковое десятичное целое
x, X	Беззнаковое целое в 16-ричной форме
f	Вещественное с десятичной точкой без экспоненты
e, E	Вещественное с экспоненциальной частью
g, G	Знаковое число в формате e или f, исходя из более компактной записи
c	Одиночный символ
s	Строка символов

Поле "ширина", если задано, содержит минимальное количество выводимых символов (если у числа меньше символов, оно дополняется пробелами или нулями).

Точность для `e`, `f` — количество символов после десятичной точки, для `g` — максимальное число значащих цифр.

Примеры использования функции:

```
int a = 7890;
float b = 12.3456;

printf("a=%d, a=%6d, a=%06d", a, a, a);
//Будет выведено: a=7890, a= 7890, a=007890

printf("b=%f, b=%6.1f, b=%e, b=%g", b, b, b, b);
//Будет выведено: b=12.345600, b= 12.3, b=1.234560e+01, b=12.3456
```

Функция `sprintf`

Аналогична функции `printf`, но первым аргументом является указатель на область памяти, куда и производится вывод. Прототип функции (определена в `stdio.h`):

`sprintf` (указатель на буфер, строка формата, список вывода)

Пример использования функции:

```
char str[128];           // Резервирование строкового буфера в памяти
short X = 2, Y = 2;      // Начальные значения параметров
sprintf(str, "Результат %d x %d = %d", X, Y, X*Y); // Вывод
```

Функция `scanf`

Используется для форматированного ввода текстовой информации из стандартного устройства и сохранение в переменных, указанных в списке (определена в `stdio.h`):

`scanf` (строка формата, список ввода)

Строка формата аналогична функции `printf`.

Функция `sin`

Вычисляет синус от угла в радианах. Прототипы функции (определены в `math.h`):

```
double sin (double argument);
float  sinf (float argument);
```

Функция `pow`

Осуществляет возведение числа-основания `basis` в степень `exponent`. Прототипы функции (определены в `math.h`):

```
double pow (double basis, double exponent);
float  powf (float basis, float exponent);
```

На аргументы накладываются некоторые ограничения, например, для отрицательного основания степень должна быть только целым числом, для нулевого основания степень не может быть отрицательной.

Функции `fabs`, `abs`

Определяют абсолютное значение аргумента. Прототипы:

```
double fabs (double x);           // Определена в math.h
float  fabsf (float x);           // Определена в math.h
int     abs  (int x);              // Определена в stdlib.h
```


Функция rand

Генерирует случайные числа. Прототип функции:

```
int rand (void);           // Определена в stdlib.h
```

При каждом вызове функция возвращает псевдослучайное число в диапазоне от 0 до RAND_MAX (в большинстве случаев константа RAND_MAX определена как 0x7FFFFFFF).

Использование функций

Использование функций в языке Си связано с тремя моментами:

- объявление функции (другие термины: декларация, прототип) — описание формы обращения к функции: типа возвращаемого значения, имени функции, типов формальных аргументов;
- определение (реализация) функции — описание выполняемых действий;
- вызов функции с подстановкой фактических аргументов.

Пример объявления:

```
bool func (short* , int);
```

Пример реализации:

```
bool func (short* data, int size)
{
    for (int i = 0; i < size; i++)
        if (!PortWrite(*data++)) return false;
    return true;
}
```

Пример вызова:

```
result = func (Array, Length);
```

Вызову функции всегда должно предшествовать либо объявление, либо реализация. Если по тексту программы реализация размещена раньше вызова, объявление может быть опущено. Использование библиотечных (а также и пользовательских) функций, определенных в отдельных файлах или уже скомпилированных библиотеках, требует наличия объявлений, которые обычно располагаются в подключаемых заголовочных файлах.

Функция возвращает только один параметр, но этот параметр, как и входные параметры, могут быть указателями на объекты любых размеров.

Особенности обработки данных в микропроцессорах

Операции над целыми числами

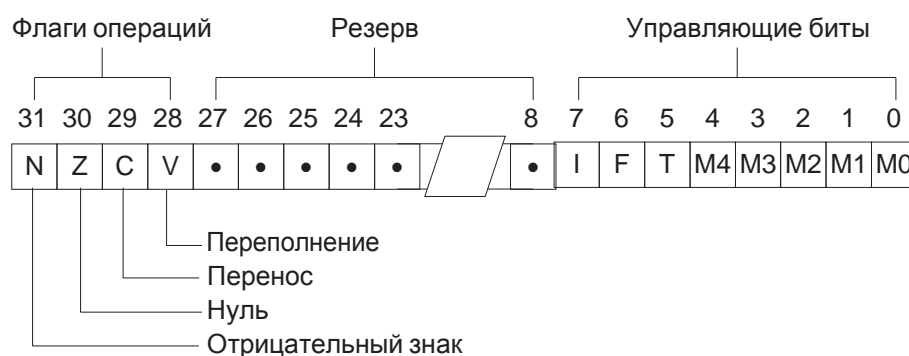
Примеры сложения/вычитания 16-разрядных чисел без знака и со знаком:

$$\begin{array}{r}
 + \quad 65530 = \quad 1111 \ 1111 \ 1111 \ 1010 \\
 \quad \quad 8 = \quad \quad \quad \quad \quad 1000 \\
 \hline
 \quad \quad 2 = \quad \mathbf{1} \ 0000 \ 0000 \ 0010 \quad - \text{перенос}
 \end{array}$$

$$\begin{array}{r}
 + \quad 30000 = \quad \mathbf{0}111 \ 0101 \ 0011 \ 0000 \\
 \quad \quad 30000 = \quad \mathbf{0}111 \ 0101 \ 0011 \ 0000 \\
 \hline
 \quad - 5536 = \quad \mathbf{1}110 \ 1010 \ 0110 \ 0000 \quad - \text{переполнение}
 \end{array}$$

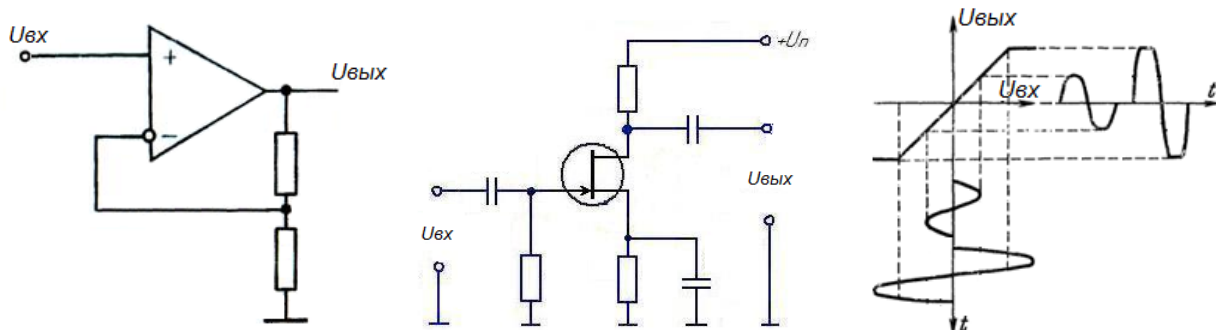
Кроме признаков (или флагов) переноса и переполнения в любом процессоре используются признаки знака (точнее: отрицательного знака) и нуля. В более сложных вычислителях имеется ряд и других признаков. Все эти особенности могут возникать в процессе проведения арифметических и логических операций и записываться в виде отдельных битов в специальный регистр состояний/признаков. Регистр обновляется после каждой операции, поэтому для идентификации указанных особенностей необходима проверка требуемых разрядов сразу после операции.

Структура регистра состояний CPSR ядра ARM Cortex-M:

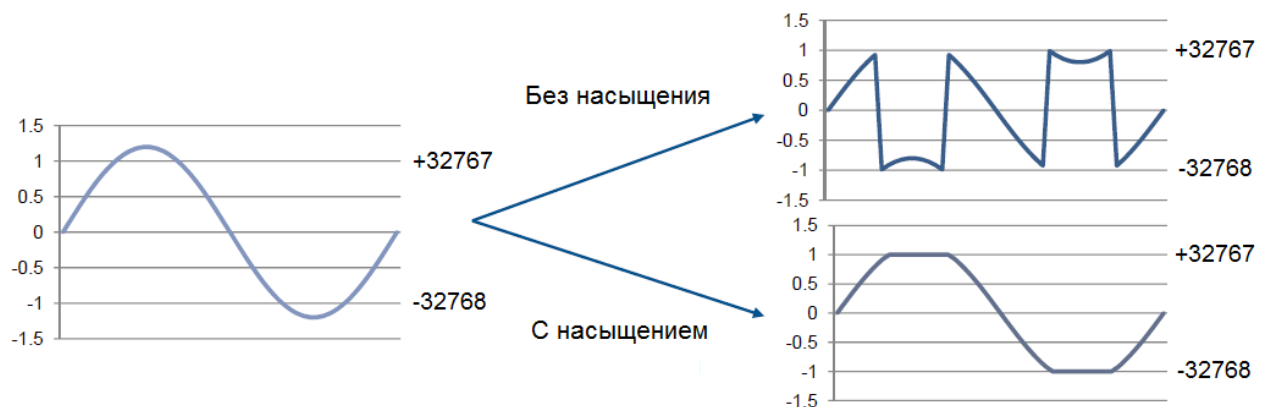


Прохождение сигнала через аналоговое и цифровое устройство

На следующем рисунке демонстрируется работа аналогового устройства при малом и большом сигнале. В последнем случае возможно естественное ограничение (насыщение) сигнала, зависящее как от особенностей выходных цепей устройства, так и напряжения питания.



При обработке цифровых сигналов обычно максимальной положительной и отрицательной амплитудам аналогового сигнала ставят в соответствие предельные значения кода заданной разрядности, например, для знакового 8-разрядного кода: -128...127, для 16-разрядного: -32768...32767. При еще большем увеличении амплитуды, как было показано на числовых примерах выше, возникает переполнение, сигнал не ограничивается, а меняет свой знак. Для реализации такого же поведения как в аналоговом устройстве, при вычислениях используют специальный режим насыщения:



Подобное действие относится к операциям цифровой обработки сигналов и на уровне команд реализовано не во всех вычислительных ядрах. В ядре ARM Cortex-M4 имеются команды ассемблера SSAT (для операций со знаком), USAT (без знака), эквивалентные функции на языке Си: `__ssat(<операция>, <разрядность>)`, `__usat(<операция>, <разрядность>)`. В функциях на языке Си в аргумент `<операция>` рекомендуется подставлять не результат ранее выполненных действий, а непосредственно включать формулу самой операции.

Рассмотренные эффекты исследуются в лабораторной работе № 1.