

Оглавление

Об этом сборнике	6
I ML & функциональное программирование	7
1 Основы Standard ML © Michael P. Fourman	8
1.1 Введение	8
2 Programming in Standard ML'97	11
An On-line Tutorial © Stephen Gilmore	11
3 Программирование в свободном синтаксисе: FSP	12
3.1 Типичная структура проекта FSP: <i>lexical skeleton</i>	12
3.1.1 Настройки (g)Vim	13
3.1.2 Дополнительные файлы	14
4 Язык <i>bl</i>	15
AST: Абстрактный Символьный Тип /Sym/	15

II	emLinux для встраиваемых систем	18
	Структура встраиваемого микроLinux	19
	Процедура сборки	20
5	clock: коридорные электронные часы = контроллер умного дурдома	21
6	gambox: игровая приставка	22
III	GNU Toolchain и C ₊ ⁺ для встраиваемых систем	23
7	Embedded Systems Programming in C ₊ ⁺ [18]	24
8	Embedded-программирование с использованием GNU Toolchain [19]	25
8.1	Введение	25
8.2	Настройка тестового стенда	26
8.2.1	Qemu ARM	27
8.2.2	Инсталляция Qemu на Debian GNU/Linux	27
8.2.3	Установка кросс-компилятора GNU Toolchain для ARM	27
8.3	Hello ARM	28
8.3.1	Сборка бинарника	29
8.3.2	Выполнение в Qemu	34
8.3.3	Другие команды монитора	37
8.4	4. More Assembler Directives	40
8.4.1	4.1. Sum an Array	40
8.4.2	4.2. String Length	40
8.5	5. Using RAM	40
8.6	6. Linker	40

8.6.1	6.1. Symbol Resolution	40
8.6.2	6.2. Relocation	40
8.7	7. Linker Script File	40
8.7.1	7.1. Linker Script Example	40
8.8	8. Data in RAM, Example	40
8.8.1	8.1. RAM is Volatile!	40
8.8.2	8.2. Specifying Load Address	40
8.8.3	8.3. Copying .data to RAM	40
8.9	9. Exception Handling	40
8.10	10. C Startup	40
8.10.1	10.1. Stack	40
8.10.2	10.2. Global Variables	40
8.10.3	10.3. Read-only Data	40
8.10.4	10.4. Startup Code	40
8.11	11. Using the C Library	40
8.12	12. Inline Assembly	40
8.13	13. Contributing	40
8.14	14. Credits	40
8.14.1	14.1. People	40
8.14.2	14.2. Tools	40
8.15	15. Tutorial Copyright	40
8.16	A. ARM Programmer's Model	40
8.17	B. ARM Instruction Set	40
8.18	C. ARM Stacks	40
9	Сборка кросс-компилятора GNU Toolchain из исходных текстов	41
9.1	Настройка целевой платформы	42

9.1.1	arm-none-eabi: процессоры ARM Cortex-Mx	42
9.1.2	i486-none-elf: ПК и промышленные PC104 без базовой ОС	42
9.1.3	arm-linux-uclibc: SoСи Cortex-A, PXA270,..	42
9.1.4	i686-linux-uclibc: микроLinux для платформы x86	42
9.2	dirs: создание структуры каталогов	42
9.3	gz: загрузка архивов исходных текстов компилятора	42
9.4	cclibs0: сборка библиотек поддержки gcc	42
9.5	binutils0	42
9.6	gcc0	42

IV

Микроконтроллеры Cortex-Mx

43

V

Технологии

44

VI

Сетевое обучения

45

VII

Прочее

46

Ф.И.Атауллаханов об учебниках США и России	47
--	----

10

Настройка редактора/IDE (g)Vim

49

10.1 для вашего собственного скриптового языка	49
--	----

Книги	50
Книги must have любому техническому специалисту	50
Математика, физика, химия	50
Обработка экспериментальных данных и метрология	51
Программирование	51
Разработка языков программирования и компиляторов	52
Lisp/Sheme	55
Haskell	55
ML	55
Электроника и цифровая техника	56
Конструирование и технология	57
Приемы ручной обработки материалов	57
Механообработка	57
Использование OpenSource программного обеспечения	58
L ^A T _E X	58
Математическое ПО: Maxima, Octave, GNUPLOT,..	58
САПР, электроника, проектирование печатных плат	59
Программирование	59
GNU Toolchain	59
Python	59
Разработка операционных систем и низкоуровневого ПО	60
Базовые науки	61
Математика	61
Физика	64
Химия	65
Стандарты и ГОСТы	65

Об этом сборнике

© Dmitry Ponyatov <dponyatov@gmail.com>

В этот сборник (блогбук) я пишу отдельные статьи и переводы, сортированные только по общей тематике, и добавляю их, когда у меня очередной раз зачесется L^AT_EX.

Это сборник черновых материалов, которые мне лень компоновать в отдельные книги, и которые пишутся просто по желанию “чтобы было”. Заказчиков на подготовку учебных материалов подобного типа нет, большая часть только на этапе освоения мной самим, просто хочется иметь некое слабоупорядоченное хранилище наработок, на которое можно дать кому-то ссылку.

Часть I

ML & функциональное программирование

Глава 1

Основы Standard ML © Michael P. Fourman

<http://homepages.inf.ed.ac.uk/mfourman/teaching/mlCourse/notes/L01.pdf>

1.1 Введение

ML обозначает “MetaLanguage”: МетаЯзык. У Robin Milner была идея создания языка программирования, специально адаптированного для написания приложений для обработки логических формул и доказательств. Этот язык должен быть **метаязыком** для манипуляции объектами, представляющими формулы на логическом **объектном языке**.

Первый *ML* был метаязыком вспомогательного пакета автоматических доказательств Edinburgh LCF. Оказалось что метаязык Милнера, с некоторыми дополнениями и уточнениями, стал инновационным и универсальным языком программирования общего назначения. Standard ML (SML) является наиболее близким потомком оригинала, другой — CAML, Haskell является более дальним родственником. В этой статье мы представляем язык SML, и рассмотрим, как он может быть использован для вычисления некоторых интересных результатов с очень небольшим усилием по программированию.

Для начала, вы считаете, что программа представляет собой последовательность команд, которые будут выполняться компьютером. Это неверно! Предоставление последовательности инструкций является лишь одним из способов программирования компьютера. Точнее сказать, что **программа — это текст спецификации вычисления**. Степень, в которой этот текст можно рассматривать как последовательность инструкций, изменяется в разных языках программирования. В этих заметках мы будем писать программы на языке *ML*, который не является столь явно императивным, как такие языки, как Си и Паскаль, в описании мер, необходимых для выполнения требуемого вычисления. Во многих отношениях *ML* **проще** чем Паскаль и Си. Тем не менее, вам может потребоваться некоторое время, чтобы оценить это.

ML в первую очередь функциональный язык: большинство программ на *ML* лучше всего рассматривать как спецификацию **значения**, которое мы хотим вычислить, без явного описания примитивных шагов, необходимых для достижения этой цели. В частности, мы не будем описывать, и вообще беспокоиться о способе, каким значения, хранимые где-то в памяти, изменяются по мере выполнения программы. Это позволит нам сосредоточиться на **организации** данных и вычислений, не втягиваясь в детали внутренней работы самого вычислителя.

В этом программирование на *ML* коренным образом отличается от тех приемов, которыми вы привыкли пользоваться в привычном императивном языке. **Попытки транслировать ваши программистские привычки на *ML* бесплодотворны — сопротивляйтесь этому искушению!**

Мы начнем этот раздел с краткого введения в небольшой фрагмент на *ML*. Затем мы используем этот фрагмент, чтобы исследовать некоторые функции, которые будут полезны в дальнейшем. Наконец, мы сделаем обзор некоторых важных аспектов *ML*.

Крайне важно попробовать эти примеры на компьютере, когда вы читаете этот текст.¹

¹ Пользовательский ввод завершается точкой с запятой “;”. В большинстве систем, “;” должна завершаться нажатием [Enter]/[Return], чтобы сообщить системе, что надо послать строку в *ML*. Эти примеры тестировались на системе Abstract Hardware Limited’s Poly/ML. В **Poly/ML** запрос ввода символ > или, если ввод неполон — #.

Примечание переводчика Для целей обучения очень удобно использовать онлайн среды, не требующие установки программ, и доступные в большинстве браузеров на любых мобильных устройствах. В качестве рекомендуемых online реализаций Standrard ML можно привести следующие:

CloudML <https://cloudml.blechschmidt.saarland/>

описан в [блогпосте В. Blechschmidt](#) как онлайн-интерпретатор диалекта [Moscow ML](#)

TutorialsPoint SML/NJ http://www.tutorialspoint.com/execute_smlnj_online.php

Moscow ML (**offline**) <http://mosml.org/> реализация Standart ML

- Сергей Романенко, Келдышевский институт прикладной математики, РАН, Москва
- Claudio Russo, Niels Kokholm, Ken Friis Larsen, Peter Sestoft
- используется движок и некоторые идеи из Caml Light © Xavier Leroy, Damien Doligez.
- порт на MacOS © Doug Currie.

Глава 2

Programming in Standard ML'97

<http://homepages.inf.ed.ac.uk/stg/NOTES/>

© Stephen Gilmore
Laboratory for Foundations of Computer Science
The University of Edinburgh

Глава 3

Программирование в свободном синтаксисе: FSP

3.1 Типичная структура проекта FSP: *lexical skeleton*

Скелет файловой структуры FSP-проекта = lexical skeleton = skelex

Создаем проект **prog** из командной строки (*Windows*):

```
1 mkdir prog
2 cd prog
3 touch src.src log.log ypp.ypp lpp.lpp hpp.hpp cpp.cpp Makefile bat.bat .gitignore
4 echo gvim -p src.src log.log ... Makefile bat.bat .gitignore >> bat.bat
5 bat
```

Создали каталог проекта, сгенерили набор пустых файлов (см. далее), и запустили батник-hepler который запустит (g)Vim.

Для пользователей GitHub mkdir надо заменить на

```
git clone -o gh git@github.com:yourname/prog.git
cd prog
git gui &
...
```

src.src		исходный текст программы на вашем скриптовом языке
log.log		лог работы ядра <i>bI</i>
ypp.ypp	flex	парсер ??
lpp.lpp	bison	лексер ??
hpp.hpp	C ₊	заголовочные файлы ??
cpp.cpp	C ₊	код ядра ??
Makefile	make	зависимости между файлами и команды сборки (для утилиты make)
bat.bat	Windows	запускалка (g)Vim ??
.gitignore	git	список масок временных и производных файлов ??

3.1.1 Настройки (g)Vim

При использовании редактора/IDE (g)Vim удобно настроить сочетания клавиш и подсветку **синтаксиса вашего скриптового языка** так, как вам удобно. Для этого нужно создать несколько файлов конфигурации .vim: по 2 файла¹ для каждого диалекта скрипт-языка², и привязать их к расширениям через

¹ (1) привязка расширения файла и (2) подсветка синтаксиса

² если вы пользуетесь сильно отличающимся синтаксисом, но скорее всего через какое-то время практики FSP у вас выработается один диалект для всех программ, соответствующий именно вашим вкусам в синтаксисе, и в этом случае его нужно будет описать только в файлах /.vim/(ftdetect|syntax).vim

dot-файлы **(g)Vim** в вашем домашнем каталоге. Подробно конфигурирование **(g)Vim** см. [10](#).

filetype.vim	(g)Vim	привязка расширений файлов (.src .log) к настройкам (g)Vim
syntax.vim	(g)Vim	синтаксическая подсветка для скриптов
/vimrc	<i>Linux</i>	настройки для пользователя
/vimrc	<i>Windows</i>	
/.vim/ftdetect/src.vim	<i>Linux</i>	привязка команд к расширению .src
/vimfiles/ftdetect/src.vim	<i>Windows</i>	
/.vim/syntax/src.vim	<i>Linux</i>	синтаксис к расширению .src
/vimfiles/syntax/src.vim	<i>Windows</i>	

3.1.2 Дополнительные файлы

README.md	github	описание проекта для репозитория github
logo.png	github	логотип
logo.ico	<i>Windows</i>	
rc.rc	<i>Windows</i>	описание ресурсов: логотип, иконки приложения, меню,...

Глава 4

Язык *bI*

AST: Абстрактный Символьный Тип /Sym/

Использование класса **Sym** и виртуально наследованных от него классов, позволяет реализовать на C_+^+ хранение и обработку **любых** данных в виде деревьев¹. Прежде всего этот *символьный тип* применяется при разборе текстовых форматов данных, и текстов программ. **Язык *bI* построен как интерпретатор AST, примерно так же как язык *Lisp* использует списки.**

```
1 // ===== ABSTRACT SYMBOLIC TYPE (AST)
2 struct Sym {
```

тип (класс) и значение элемента данных

```
1 // =====
```

¹ в этом АСТ близок к традиционной аббревиатуре AST: Abstract Syntax Tree

2	string tag;	// data type / class
3	string val;	// symbol value

конструкторы (токен используется в лексере)

1	// _____ constructors	
2	Sym(string, string);	// <T:V>
3	Sym(string);	// token

Хранение вложенных элементов реализовано через указатели на базовый тип **Sym**. Благодаря виртуальному наследованию и использованию RTTI, этими указателями можно пользоваться для работы с любыми другими наследованными типами данных²

AST может хранить (и обрабатывать) вложенные элементы

1	// _____ nest[]ed elements	
2	vector<Sym*> nest;	
3	void push(Sym*);	
4	void pop();	

параметры (и поля класса)

1	// _____ par{}ameters	
2	map<string, Sym*> pars;	
3	void par(Sym*);	// add parameter

вывод дампа объекта в текстовом формате

1	// _____ dumping	
---	------------------	--

² числа, списки, высокоуровневые и скомпилированные функции, элементы GUI,..


```

2  virtual string dump(int depth=0);    // dump symbol object as text
3  virtual string tagval();             // <T:V> header string
4  string tagstr();                     // <T:'V'> Str-like header string
5  string pad(int);                     // padding with tree decorators

```

Операции над *символами* выполняются через использование набора *операторов*:

вычисление объекта

```

1 // ----- compute (evaluate)
2  virtual Sym* eval();

```

операторы

```

1 // ----- operators
2  virtual Sym* str();                 // str(A)    string representation
3  virtual Sym* eq(Sym*);              // A = B    assignment
4  virtual Sym* inher(Sym*);          // A : B    inheritance
5  virtual Sym* member(Sym*);         // A % B,C  named member (class slot)
6  virtual Sym* at(Sym*);              // A @ B    apply
7  virtual Sym* add(Sym*);             // A + B    add
8  virtual Sym* div(Sym*);             // A / B    div
9  virtual Sym* ins(Sym*);             // A += B   insert
10 };

```

Часть II

em*Linux* для встраиваемых систем

Структура встраиваемого микро*Linux*

syslinux Загрузчик

em*Linux* поставляется в виде двух файлов:

1. ядро `(HW)(APP).kernel`
2. сжатый образ корневой файловой системы `(HW)(APP).rootfs`

Загрузчик считывает их с одного из носителей данных, который поддерживается загрузчиком³, распаковывает в память, включив защищенный режим процессора, и передает управление ядру *Linux*.

Для работы em*Linux* не требуются какие-либо носители данных: вся (виртуальная) файловая система располагается в ОЗУ. При необходимости к любому из каталогов корневой ФС может быть *подмонтирована* любая существующая дисковая или сетевая файловая система (FAT,NTFS,Samba,NFS,...), причем можно явно запретить возможность записи на нее, защитив данные от разрушения.

Использование rootfs в ОЗУ позволяет гарантировать защиту базовой ОС и пользовательских исполняемых файлов от внезапных выключений питания и ошибочной записи на диск. Еще большую защиту даст отключение драйверов загрузочного носителя в ядре. Если отключить драйвера SATA/IDE и грузиться с USB флешки, практически невозможно испортить основную установку ОС и пользовательские файлы на чужом компьютере.

kernel Ядро *Linux* 3.13.xx

³ IDE/SATA HDD, CDROM, USB флешка, сетевая загрузка с BOOTP-сервера по Ethernet

ulibc Базовая библиотека языка Си

busybox Ядро командной среды UNIX, базовые сетевые сервера

дополнительные библиотеки

zlib сжатие/распаковка gzip

??? библиотека помехозащищенного кодирования

png библиотека чтения/записи графического формата .png

freetype рендер векторных шрифтов (TTF)

SDL полноэкранная (игровая) графика, аудио, джойстик

кодеки аудио/видео форматов: ogg vorbis, mp3, mpeg, ffmpeg/gsm

К базовой системе добавляются пользовательские программы */usr/bin*
и динамические библиотеки */usr/lib*.

Процедура сборки

Глава 5

clock: коридорные электронные часы =
контроллер умного дурдома

Глава 6

gambox: игровая приставка

Часть III

GNU Toolchain и C_{+}^{+} для встраиваемых систем

Глава 7

Embedded Systems Programming in C_{+}^{+} [18]

1

Глава 8

Embedded-программирование с использованием GNU Toolchain [19]

© Vijay Kumar B.

¹

8.1 Введение

GNU toolchain широко используется при разработки программного обеспечения для встраиваемых систем. Этот тип разработки ПО также называют *низкоуровневым программированием standalone* или *bare metal* программированием (на Си и C_+^+). Написание низкоуровневого кода на C_+^+ добавляет программисту новых проблем, требующих глубокого понимания инструмента разработчика — **GNU Toolchain**. Руководства разработчика **GNU Toolchain** предоставляют отличную информацию по инстру-

¹ © <http://bravegnu.org/gnu-eprog/>

ментария, но с точки зрения самого **GNU Toolchain**, чем с точки зрения решаемой проблемы. Поэтому было написано это руководство, в котором будут описаны типичные проблемы, с которыми сталкивается начинающий разработчик.

Этот учебник стремится занять свое место, объясняя использование **GNU Toolchain** с точки зрения практического использования. Надеемся, что его будет достаточно для разработчиков, собирающихся освоить и использовать **GNU Toolchain** в их embedded проектах.

В иллюстративных целях была выбрана встроенная система на базе процессорного ядра ARM, которая эмулируется в пакете **Qemu**. С таким подходом вы сможете освоить **GNU Toolchain** с комфортом на вашем рабочем компьютере, без необходимости вкладываться в “физическое” железо, и бороться со сложностями с его запуском. Учебник не стремится обучить работе с архитектурой ARM, для этого вам нужно будет воспользоваться дополнительными книгами или онлайн-учебниками типа:

- ARM Assembler <http://www.heyrick.co.uk/assembler/>
- ARM Assembly Language Programming <http://www.arm.com/miscPDFs/9658.pdf>

Но для удобства читателя, некоторое множество часто используемых ARM-инструкций описано в приложении 8.17.

8.2 Настройка тестового стенда

В этом разделе описано, как настроить на вашей рабочей станции простую среду разработки и тестирования ПО для платформы ARM, используя **Qemu** и **GNU Toolchain**. **Qemu** это программный² эмулятор нескольких распространенных аппаратных платформ. Вы можете написать программу на ассемблере и C_+^+ , скомпилировать ее используя **GNU Toolchain** и отладить ее в эмуляторе **Qemu**.

² для i386 — программно-аппаратный, использует средства виртуализации хост-компьютера

8.2.1 Qemu ARM

Будем использовать **Qemu** для эмуляции отладочной платы **Gumstix** на базе процессора PXA255. Для работы с этим учебником у вас должен быть установлен **Qemu** версии не ниже 0.9.1.

Процессор³ PXA255 имеет ядро ARM с набором инструкций ARMv5TE. PXA255 также имеет в своем составе несколько блоков периферии. Некоторая периферия будет описана в этом курсе далее.

8.2.2 Инсталляция Qemu на *Debian GNU/Linux*

Этот учебник требует **Qemu** версии не ниже 0.9.1. Пакет **Qemu** доступный для современных дистрибутивов *Debian GNU/Linux*, вполне удовлетворяет этим условиям, и собирать свежий **Qemu** из исходников совсем не требуется⁴. Установим пакет командой:

```
$ sudo apt install qemu
```

8.2.3 Установка кросс-компилятора GNU Toolchain для ARM

Если вы предпочитаете простые пути, установите пакет кросс-компилятора командной

```
sudo apt install gcc-arm-none-eabi
```

или

1. Годные чуваки из CodeSourcery⁵ уже давно запилили несколько вариантов **GNU Toolchain**ов для разных ходовых архитектур. Скачайте готовую бинарную бесплатную lite-сборку **GNU Toolchain-ARM**

³ Точнее SoC: система-на-кристалле

⁴ хотя может быть и очень хочется

⁵ подразделение Mentor Graphics

2. Распакуйте tar-архив в каталог */toolchains*:

```
$ mkdir ~/toolchains
$ cd ~/toolchains
$ tar -jxf ~/downloads/arm-2008q1-126-arm-none-eabi-i686-pc-linux-gnu.tar.bz2
```

3. Добавьте bin-каталог тулчейна в переменную среды PATH.

```
$ PATH=$HOME/toolchains/arm-2008q1/bin:$PATH
```

4. Чтобы каждый раз не выполнять предыдущую команду, вы можете прописать ее в дот-файл **.bashrc**.

Для совсем упертых подойдет рецепт сборки полного комплекта кросс-компилятора из исходных текстов, описанный в [9](#).

8.3 Hello ARM

В этом разделе вы научитесь пользоваться arm-ассемблером, и тестировать вашу программу на голом железе — эмуляторе платы **connex/Qemu**.

Файл исходника ассемблерной программы состоит из последовательности инструкций, по одной на каждую строку. Каждая инструкция имеет формат (каждый компонент не обязателен):

<метка>: <инструкция> @ <комментарий>

метка — типичный способ пометить адрес инструкции в памяти. Метка может быть использована там, где требуется указать адрес, например как операнд в команде перехода. Метка может состоять из латинских букв, цифр⁶, символов `_` и `$`.

⁶ не может быть первым символом метки

инструкция может быть инструкцией процессора или директивой ассемблера, начинающейся с точки “.”

комментарий начинается с символа @ — все последующие символы игнорируются до конца строки

Вот пример простой ассемблерной программы для процессора ARM, складывающей два числа:

Листинг 1: Сложение чисел

```
1      .text
2 start:      @ Label, not really required
3      mov    r0, #5      @ Load register r0 with the value 5
4      mov    r1, #4      @ Load register r1 with the value 4
5      add    r2, r1, r0   @ Add r0 and r1 and store in r2
6
7 stop:      b stop      @ Infinite loop to stop execution
```

метка необязательна
загрузить в регистр r0 значение 5
загрузить в регистр r1 значение 4
сложить r0+r1 и сохранить в r2
пустой бесконечный цикл для останова выполнения

`.text` ассемблерная директива, указывающая что последующий код должен быть *ассемблирован* в *секцию кода .text* а не в секцию *.data*. *Секции* будут подробно описаны далее.

8.3.1 Сборка бинарника

Сохраните программу в файл **add.s**⁷. Для ассемблирования файла вызовите ассемблер **as**:

⁷ .s или .S стандартное расширение в мире *Linux*, указывает что это файл с программной на ассемблере

```
$ arm-none-eabi-as -o add.o add.s
```

Опция `-o` указывает выходной файл с *объектным кодом*, имеющий стандартное расширение `.o`⁸.

Команды кросс-тулчейна всегда имеют префикс целевой архитектуры (target triplet), для которой они были собраны, чтобы предотвратить конфликт имен с хост-тулчейном для вашего рабочего компьютера. Далее утилиты **GNU Toolchain** будут использоваться без префикса для лучшей читаемости. **не забывайте добавлять `arm-none-eabi-`, иначе получите множество странных ошибок типа “unexpected command”**.

```
$ (arm-none-eabi-)as -o add.o add.s
```

```
$ (arm-none-eabi-)objdump -x add.o
```

Вывод команды **arm-none-eabi-objdump -x**: ELF-заголовки в файле объектного кода

```
1
2 add.o:          file format elf32-littlearm
3 add.o
4 architecture: armv4, flags 0x00000010:
5 HAS_SYMS
6 start address 0x00000000
7 private flags = 50000000: [Version5 EABI]
8
9 Sections:
10 Idx Name          Size      VMA          LMA          File off    Algn
11  0  .text          00000010   00000000   00000000   00000034   2**2
```

⁸ и внутренний формат ELF (как завещал великий *Linux*)

```

12          CONTENTS, ALLOC, LOAD, READONLY, CODE
13 1 .data      00000000 00000000 00000000 00000044 2**0
14          CONTENTS, ALLOC, LOAD, DATA
15 2 .bss      00000000 00000000 00000000 00000044 2**0
16          ALLOC
17 3 .ARM.attributes 00000014 00000000 00000000 00000044 2**0
18          CONTENTS, READONLY
19 SYMBOL TABLE:
20 00000000 1 d .text 00000000 .text
21 00000000 1 d .data 00000000 .data
22 00000000 1 d .bss 00000000 .bss
23 00000000 1 .text 00000000 start
24 00000000 c 1 .text 00000000 stop
25 00000000 1 d .ARM.attributes 00000000 .ARM.attributes

```

сейкция `.text` имеет размер `Size=0x0010 = 16` байт, и содержит **машинный код**:

машинный код из секции `.text`: **objdump -d**

```

1
2 add.o:      file format elf32-littlearm
3
4
5 Disassembly of section .text:
6
7 00000000 <start>:
8   0:   e3a00005    mov r0, #5
9   4:   e3a01004    mov r1, #4
10  8:   e0812000    add r2, r1, r0
11

```

```
12 00000000 c <stop>:
13   c:   e affffffe      b      c <stop>
```

Для генерации **исполняемого файла**⁹ вызовем *линкер* ld:

```
$ arm-none-eabi-ld -Ttext=0x0 -o add.elf add.o
```

Опять, опция `-o` задает выходной файл. `-Ttext=0x0` явно указывает адрес, от которого будут отсчитываться все метки, т.е. секция инструкций начинается с адреса `0x0000`. Для просмотра адресов, назначенных меткам, можно использовать команду `(arm-none-eabi-)nm`¹⁰:

```
ponyator@bs:/tmp$ arm-none-eabi-nm add.elf
...
00000000 t start
0000000c t stop
```

* если вы забудете опцию `-T`, вы получите этот вывод с адресами `00008xxx` — эти адреса были заданы при компиляции **GNU Toolchain-ARM**, и могут не совпадать с необходимыми вам. Проверяйте ваши `.elf`ы с помощью **nm** или **objdump**, если программы не запускаются, или **Qemu** ругается на ошибки (защиты) памяти.

Обратите внимание на *назначение адресов* для меток **start** и **stop**: адреса начинаются с `0x0`. Это адрес первой инструкции. Метка **stop** находится после третьей инструкции. Каждая инструкция занимает

⁹ обычно тот же формат ELF.o, слепленный из одного или нескольких объектных файлов, с некоторыми модификациями см. опцию `-T` далее

¹⁰ NaMes

4 байта¹¹, так что `stop` находится по адресу $0xC_{hex} = 12_{dec}$. *Линковка* с другим *базовым адресом* `-Ttext=nnnn` приведет к сдвигу адресов, назначенных меткам.

Выходной файл, созданный `ld` имеет формат, который называется **ELF**. Существует множество форматов, предназначенных для хранения выполняемого и объектного кода¹². Формат ELF применяется для хранения машинного кода, если вы запускаете его в базовой ОС¹³, но поскольку мы собираемся запускать нашу программу на bare metal¹⁴, мы должны сконвертировать полученный .elf файл в более простой *бинарный формат*.

Файл в *бинарном формате* содержит последовательность байт, начинающуюся с определенного адреса памяти, иногда его называют *образом памяти*. Этот формат типичен для утилит программирования флеш-памяти микроконтроллеров, так как все что требуется сделать — последовательно скопировать каждый байт из файла в FlashROM-память микроконтроллера, начиная с определенного начального адреса в памяти.¹⁵

Команда **GNU Toolchain objcopy** используется для конвертирования машинного кода между разными объектными форматами. Типичное использование:

```
$ objcopy -O <выходной_формат> <входной_файл> <выходной_файл>
```

Конвертируем `add.elf` в бинарный формат:

```
$ objcopy -O binary add.elf add.bin
```

Проверим размер полученного бинарного файла, он должен быть равен тем же 16 байтам¹⁶:

¹¹ в множестве команд ARM-32, если вы компилируете код для микроконтроллера Cortex-Mx в режиме команд Thumb или Thumb2, команды 16-битные, т.е. 2 байта

¹² можно отдельно отметить Microsoft COFF (объектные файлы .obj) и PE (.exe)cutable

¹³ прежде всего “большой” или встраиваемый *Linux*

¹⁴ голом железе

¹⁵ Та же операция выполняется и для SoC-систем с NAND-флешем: записать бинарный образ начиная с некоторого аппаратно фиксированного адреса.

¹⁶ 4 инструкции по 4 байта каждая

```
$ ls -al add.bin
-rw-r--r-- 1 vijaykumar vijaykumar 16 2008-10-03 23:56 add.bin
```

Если вы не доверяете **ls**, можно дизассемблировать бинарный файл:

```
ponyatov@bs:/tmp$ arm-none-eabi-objdump -b binary -m arm -D add.bin
```

```
add.bin:      file format binary
```

```
Disassembly of section .data:
```

```
00000000 <.data>:
   0:   e3a00005      mov     r0, #5
   4:   e3a01004      mov     r1, #4
   8:   e0812000      add     r2, r1, r0
  c:   eaffffff      b       0xc
ponyatov@bs:/tmp$
```

Опция **-b** задает формат файла, опция **-m** (machine) архитектуру процессора, получить полный список сочетаний **-b/-m** можно командой **arm-none-eabi-objdump -i**.

8.3.2 Выполнение в Qemu

Когда ARM-процессор сбрасывается, он начинает выполнять команды с адресе 0x0. На плате Commplex установлен флеш на 16 мегабайт, начинающийся с адрес 0x0. Таким образом, при сбросе будут выполняться инструкции с начала флеша.

Когда **Qemu** эмулирует плату connex, в командной строке должен быть указан файл, который будет считаться образом флеш-памяти. Формат флеша очень прост — это побайтный образ флеша без каких-либо полей или заголовков, т.е. это тот же самый *бинарный формат*, описанный выше.

Для тестирования программы в эмуляторе Gumstix connex, сначала мы создаем 16-мегабайтный файл флеша, копируя 16М нулей из файла **/dev/zero** с помощью команды **dd**. Данные копируются 4Кбайтными блоками¹⁷ (4096 x 4K):

```
$ dd if=/dev/zero of=flash.bin bs=4K count=4K
4096+0 записей получено
4096+0 записей отправлено
скопировано 16777216 байт (17 MB), 0,0153502 с, 1,1 GB/с
```

```
$ du -h flash.bin
16M      flash.bin
```

Затем переписываем начало **flash.bin** копируя в него содержимое **add.bin**:

```
$ dd if=add.bin of=flash.bin bs=4K conv=notrunc
0+1 записей получено
0+1 записей отправлено
скопировано 16 байт (16 B), 0,000173038 с, 92,5 kB/с
```

После сброса процессор выполняет код с адреса 0x0, и будут выполняться инструкции нашей программы. Команда запуска **Qemu**:

```
$ qemu-system-arm -M connex -pflash flash.bin -nographic -serial /dev/null
```

¹⁷ опция bs= (blocksize)

```
QEMU 2.1.2 monitor - type 'help' for more information
(qemu)
```

Опция **-M connex** выбирает режим эмуляции: **Qemu** поддерживает эмуляцию нескольких десятков железа на базе ARM процессоров. Опция **-pflash** указывает файл образа флеша, который должен иметь определенный размер (16M). **-nographic** отключает эмуляцию графического дисплея (в отдельном окне). Самая важная опция **-serial /dev/null** подключает последовательный порт платы на **/dev/null**, при этом в терминале после запуска **Qemu** вы получите **консоль монитора**.

Qemu выполняет инструкции, и останавливается в бесконечном цикле на **stop**, выполняя команду **stop: b stop**. Для просмотра содержимого регистров процессора воспользуемся *монитором*. Монитор имеет интерфейс командной строки, который вы можете использовать для контроля работы эмулируемой системы. Если вы запустите **Qemu** как указано выше, монитор будет доступен через **stdio**.

Для просмотра регистров выполним команду **info registers**:

```
(qemu) info registers
R00=00000005 R01=00000004 R02=00000009 R03=00000000
R04=00000000 R05=00000000 R06=00000000 R07=00000000
R08=00000000 R09=00000000 R10=00000000 R11=00000000
R12=00000000 R13=00000000 R14=00000000 R15=0000000c
PSR=400001d3 -Z-- A svc32
FPSCR: 00000000
```

Обратите внимание на значения в регистрах **r00..r02**: 4, 5 и ожидаемый результат 9. Особое значение для ARM имеет регистр **r15**: он является указателем команд, и содержит адрес текущей выполняемой машинной команды, т.е. **0x000c: b stop**.

8.3.3 Другие команды монитора

Несколько полезных команд:

help	список доступных команд
quit	выход из эмулятора
xp /fmt addr	вывод содержимого физической памяти с адреса addr
system_reset	перезапуск

Команда **xp** требует некоторых пояснений. Аргумент **/fmt** указывает как будет выводиться содержимое памяти, и имеет синтаксис **<счетчик><формат><размер>**:

счетчик число элементов данных

size размер одного элемента в битах: b=8 бит, h=16, w=32, g=64

format определяет формат вывода:

x hex

d десятичные целые со знаком

u десятичные без знака

o 8ричные

c символ (char)

i инструкции ассемблера

Команда **xp** в формате **i** будет дизассемблировать инструкции из памяти. Выведем дамп с адреса 0x0 указав **fmt=4iw**: 4 — 4 , i — инструкции размером w=32 бита:

(qemu) xp /4wi 0x0

0x00000000:	e3a00005	mov	r0, #5 ; 0x5
0x00000004:	e3a01004	mov	r1, #4 ; 0x4
0x00000008:	e0812000	add	r2, r1, r0
0x0000000c:	eafffffe	b	0xc

8.4 4. More Assembler Directives

8.4.1 4.1. Sum an Array

8.4.2 4.2. String Length

8.5 5. Using RAM

8.6 6. Linker

8.6.1 6.1. Symbol Resolution

8.6.2 6.2. Relocation

8.7 7. Linker Script File

8.7.1 7.1. Linker Script Example

8.8 8. Data in RAM, Example

8.8.1 8.1. RAM is Volatile!

8.8.2 8.2. Specifying Load Address

8.8.3 8.3. Copying .data to RAM

8.9 9. Exception Handling

Глава 9

Сборка кросс-компилятора GNU Toolchain из исходных текстов

9.1 Настройка целевой платформы

9.1.1 arm-none-eabi: процессоры ARM Cortex-Mx

9.1.2 i486-none-elf: ПК и промышленные PC104 без базовой ОС

9.1.3 arm-linux-uclibc: SoCи Cortex-A, PXA270,..

9.1.4 i686-linux-uclibc: микроLinux для платформы x86

9.2 dirs: создание структуры каталогов

9.3 gz: загрузка архивов исходных текстов компилятора

Часть IV

Микроконтроллеры Cortex-Mx

Часть V

Технологии

Часть VI

Сетевое обучения

Часть VII

Прочее

Ф.И.Атауллаханов об учебниках США и России

© Доктор биологических наук Фазли Иноятович Атауллаханов.
МГУ им. М. В. Ломоносова, Университет Пенсильвании, США

<http://www.nkj.ru/archive/articles/19054/>

...

У необходимости рекламировать науку есть важная обратная сторона: каждый американский учёный непрерывно, с первых шагов и всегда, учится излагать свои мысли внятно и популярно. В России традиции быть понятными у учёных нет. Как пример я люблю приводить двух великих физиков: русского Ландау и американца Фейнмана. Каждый написал многотомный учебник по физике. Первый — знаменитый “Ландау-Лифшиц”, второй — “Лекции по физике”. Так вот, “Ландау-Лифшиц” прекрасный справочник, но представляет собой полное издевательство над читателем. Это типичный памятник автору, который был, мягко говоря, малоприятным человеком. Он излагает то, что излагает, абсолютно пренебрегая своим читателем и даже издеваясь над ним. А у нас целые поколения выросли на этой книге, и считается, что всё нормально, кто справился, тот молодец. Когда я столкнулся с “Лекциями по физике” Фейнмана, я просто обалдел: оказывается, можно по-человечески разговаривать со своими коллегами, со студентами, с аспирантами. Учебник Ландау — пример того, как устроена у нас вся наука. Берёшь текст русской статьи, читаешь с самого начала и ничего не можешь понять, а иногда сомневаешься, понимает ли автор сам себя. Конечно, крупницы осмысленного и разумного и оттуда можно вынуть. Но автор явно считает, что это твоя работа — их оттуда извлечь. Не потому, что он не хочет быть понятным, а потому, что его не научили правильно писать. Не учат у нас человека ни писать, ни говорить понятно, это считается неважным.

...

Думаю, американская наука в целом устроена именно так: она продаёт не просто себя, а всю свою страну. Сегодня американцы дороги не метут, сапоги не тачают, даже телевизоры не собирают, за них это делает весь остальной мир. А что же делают американцы? Самая богатая страна в мире? Они объяснили,

в первую очередь самим себе, а заодно и всему миру, что они — мозг планеты. Они изобретают. “Мы придумываем продукты, а вы их делайте. В том числе и для нас”. Это прекрасно работает, поэтому они очень ценят науку.

...

Глава 10

Настройка редактора/IDE (g)Vim

При использовании редактора/IDE (g)Vim удобно настроить сочетания клавиш и подсветку синтаксиса языков, которые вы используете так, как вам удобно.

10.1 для вашего собственного скриптового языка

Через какое-то время практики FSP у вас выработается один диалект скриптов для всех программ, соответствующий именно вашим вкусам в синтаксисе, и в этом случае его нужно будет описать только в файлах `/.vim/(ftdetect|syntax).vim`, и привязать их к расширениям через dot-файлы (g)Vim в вашем домашнем каталоге:

filetype.vim	(g)Vim	привязка расширений файлов (.src .log) к настройкам (g)Vim
syntax.vim	(g)Vim	синтаксическая подсветка для скриптов
/vimrc	<i>Linux</i>	настройки для пользователя
/vimrc	<i>Windows</i>	
/vim/ftdetect/src.vim	<i>Linux</i>	привязка команд к расширению .src
/vimfiles/ftdetect/src.vim	<i>Windows</i>	
/vim/syntax/src.vim	<i>Linux</i>	синтаксис к расширению .src
/vimfiles/syntax/src.vim	<i>Windows</i>	

Книги must have любому техническому специалисту

Математика, физика, химия

- Бермант Математический анализ [22]
- Кремер Теория вероятностей и матстатистика [23]
- Смит Цифровая обработка сигналов [24]

Фейнмановские лекции по физике

1. Современная наука о природе. Законы механики. [27]
2. Пространство. Время. Движение. [28]
3. Излучение. Волны. Кванты. [29]
4. Кинетика. Теплота. Звук. [30]
5. Электричество и магнетизм [31]
6. Электродинамика. [32]
7. Физика сплошных сред. [33]
8. Квантовая механика 1. [34]

Обработка экспериментальных данных и метрология

- Князев, Черкасский **Начала обработки экспериментальных данных** [25]

Программирование

- Система контроля версий **Git** и **git-хостинга GitHub**

хранение наработок с полной историей редактирования, правок, релизов для разных заказчиков или вариантов использования

- **Язык Python** [20]

написание простых скриптов обработки данных, автоматизации, графических оболочек и т.п. утилит

- **Язык C_+^+ , утилиты GNU toolchain** [18, 19] (gcc/g++, make, ld)

базовый Си, ООП очень кратко¹, без излишеств профессионального программирования², чисто вспомогательная роль для написания вычислительных блоков и критичных к скорости/памяти секций, использовать в связке с Python.

Знание базового Си **критично при использовании микроконтроллеров**, из C_+^+ необходимо владение особенностями использования ООП и управления крайне ограниченной памятью: пользовательские менеджеры памяти, статические классы.

- Использование утилит **flex/bison**

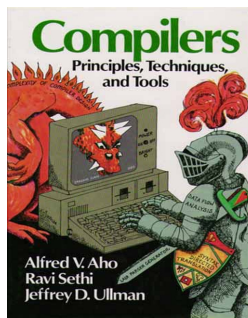
обработка текстовых форматов данных, часто необходимая вещь.

¹ наследование, полиморфизм, операторы для пользовательских типов, использование библиотеки STL

² мегабиблиотека Boost, написание своих библиотек шаблонов и т.п.

Литература

Разработка языков программирования и компиляторов



Dragon Book

Компиляторы. Принципы, технологии, инструменты.

Альфред Ахо, Рави Сети, Джеффри Ульман.

Издательство Вильямс, 2003.

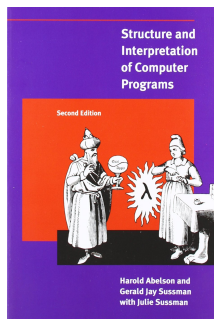
ISBN 5-8459-0189-8

[2] **Compilers: Principles, Techniques, and Tools**

Aho, Sethi, Ullman

Addison-Wesley, 1986.

ISBN 0-201-10088-6



SICP

[3]

Структура и интерпретация компьютерных программ

Харольд Абельсон, Джеральд Сассман

ISBN 5-98227-191-8

EN: web.mit.edu/alexmv/6.037/sicp.pdf



[4]

Функциональное программирование

Филд А., Харрисон П.

М.: Мир, 1993
ISBN 5-03-001870-0



[5]

Функциональное программирование: применение и реализация

П.Хендерсон
М.: Мир, 1983



[6]

LLVM. Инфраструктура для разработки компиляторов

Бруно Кардос Лопес, Рафаэль Аулер

Lisp/Sheme

Haskell

ML

- [7] <http://homepages.inf.ed.ac.uk/mfourman/teaching/mlCourse/notes/L01.pdf>

Basics of Standard ML

© Michael P. Fourman

перевод 1

- [8] <http://www.soc.napier.ac.uk/course-notes/sml/manual.html>

A Gentle Introduction to ML

© Andrew Cumming, Computer Studies, Napier University, Edinburgh

- [9] <http://www.cs.cmu.edu/~rwh/smlbook/book.pdf>

Programming in Standard ML

© Robert Harper, Carnegie Mellon University

Электроника и цифровая техника



[10]

An Introduction to Practical Electronics, Microcontrollers and Software Design

Bill Collis

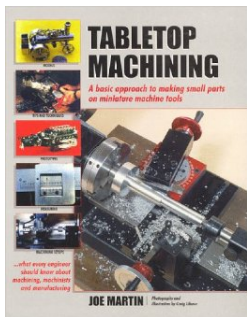
2 edition, May 2014

<http://www.techideas.co.nz/>

Конструирование и технология

Приемы ручной обработки материалов

Механообработка



[11]

Tabletop Machining

Martin, Joe and Libuse, Craig
Sherline Products, 2000

[12]

Home Machinists Handbook

Briney, Doug, 2000

[13]

Маленькие станки

Евгений Васильев
Псков, 2007

<http://www.coilgun.ru/stanki/index.htm>

Использование OpenSource программного обеспечения

Л^AT_EX

- [14] **Набор и вёрстка в системе Л^AT_EX**
С.М. Львовский
3-е издание, исправленное и дополненное, 2003
<http://www.mccme.ru/free-books/llang/newllang.pdf>
- [15] **e-Readers and Л^AT_EX**
Alan Wetmore
<https://www.tug.org/TUGboat/tb32-3/tb102wetmore.pdf>
- [16] **How to cite a standard (ISO, etc.) in BibЛ^AT_EX?**
<http://tex.stackexchange.com/questions/65637/>

Математическое ПО: Maxima, Octave, GNUPLOT, ..

- [17] **Система аналитических вычислений Maxima для физиков-теоретиков**
В.А. Ильина, П.К.Силаев
<http://tex.bog.msu.ru/numtask/max07.ps>

САПР, электроника, проектирование печатных плат

Программирование

GNU Toolchain

- [18] **Embedded Systems Programming in C₊**

© <http://www.bogotobogo.com/>

<http://www.bogotobogo.com/cplusplus/embeddedSystemsProgramming.php>

- [19] **Embedded Programming with the GNU Toolchain**

Vijay Kumar B.

<http://bravegnu.org/gnu-eprog/>

Python

- [20] **Язык программирования Python**

Россум, Г., Дрейк, Ф.Л.Дж., Откидач, Д.С., Задка, М., Левис, М., Монтаро, С., Реймонд, Э.С., Кучлинг, А.М., Лембург, М.-А., Йи, К.-П., Ксиллаг, Д., Петрилли, Х.Г., Варсав, Б.А., Ахлстром, Дж.К., Роскинд, Дж., Шеменор, Н., Мулендер, С.

© Stichting Mathematisch Centrum, 1990–1995 and Corporation for National Research Initiatives, 1995–2000 and BeOpen.com, 2000 and Откидач, Д.С., 2001

<http://rus-linux.net/MyLDP/BOOKS/python.pdf>

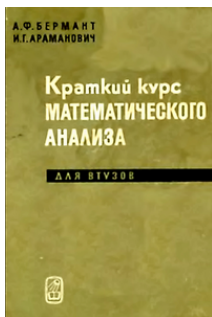
Python является простым и, в то же время, мощным интерпретируемым объектно-ориентированным языком программирования. Он предоставляет структуры данных высокого уровня, имеет изящный синтаксис и использует динамический контроль типов, что делает его идеальным языком для быстрого написания различных приложений, работающих на большинстве распространенных платформ. Книга содержит вводное руководство, которое может служить учебником для начинающих, и справочный материал с подробным описанием грамматики языка, встроенных возможностей и возможностей, предоставляемых модулями стандартной библиотеки. Описание охватывает наиболее распространенные версии Python: от 1.5.2 до 2.0.

Разработка операционных систем и низкоуровневого ПО

[21] OSDev Wiki
<http://wiki.osdev.org>

Базовые науки

Математика



[22]

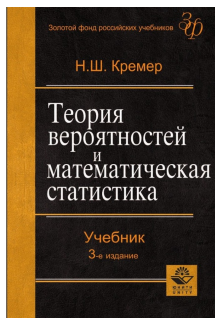
Краткий курс математического анализа для ВТУЗов

Бермант А.Ф., Араманович И.Г.

М.: Наука, 1967

<https://drive.google.com/file/d/0B0u4WeMj0894U1Y1dEJ6cncxU28/view?usp=sharing>

Пятое издание известного учебника, охватывает большинство вопросов программы по высшей математике для инженерно-технических специальностей вузов, в том числе дифференциальное исчисление функций одной переменной и его применение к исследованию функций; дифференциальное исчисление функций нескольких переменных; интегральное исчисление; двойные, тройные и криволинейные интегралы; теорию поля; дифференциальные уравнения; степенные ряды и ряды Фурье. Разобрано много примеров и задач из различных разделов механики и физики. **Отличается крайней доходчивостью и отсутствием филолианов и “легко догадаться”.**



Теория вероятностей и математическая статистика

[23]

Наум Кремер
М.: Юнити, 2010



[24]

Цифровая обработка сигналов. Практическое руководство для инженеров и научных работников

Стивен Смит
Додэка XXI, 2008
ISBN 978-5-94120-145-7

В книге изложены основы теории цифровой обработки сигналов. Акцент сделан на доступности изложения материала и объяснении методов и алгоритмов так, как они понимаются при практическом

использовании. Цель книги - практический подход к цифровой обработке сигналов, позволяющий преодолеть барьер сложной математики и абстрактной теории, характерных для традиционных учебников. Изложение материала сопровождается большим количеством примеров, иллюстраций и текстов программ

[25] **Начала обработки экспериментальных данных**

Б.А.Князев, В.С.Черкасский

Новосибирский государственный университет, кафедра общей физики, Новосибирск, 1996

http://www.phys.nsu.ru/cherk/Methodizm_old.PDF

Учебное пособие предназначено для студентов естественно-научных специальностей, выполняющих лабораторные работы в учебных практикумах. Для его чтения достаточно знаний математики в объеме средней школы, но оно может быть полезно и тем, кто уже изучил математическую статистику, поскольку исходным моментом в нем является не математика, а эксперимент. Во второй части пособия подробно описан реальный эксперимент — от появления идеи и проблем постановки эксперимента до получения результатов и обработки данных, что позволяет получить менее формализованное представление о применении математической статистики. Пособие дополнено обучающей программой, которая позволяет как углубить и уточнить знания, полученные в методическом пособии, так и проводить собственно обработку результатов лабораторных работ. Приведен список литературы для желающих углубить свои знания в области математической статистики и обработки данных.

Физика



[26]

Савельев И.В.



Фейнмановские лекции по физике

Ричард Фейнман, Роберт Лейтон, Мэттью Сэндс

[27] Современная наука о природе. Законы механики.

[28] Пространство. Время. Движение.

[29] Излучение. Волны. Кванты.

- [30] Кинетика. Теплота. Звук.
- [31] Электричество и магнетизм.
- [32] Электродинамика.
- [33] Физика сплошных сред.
- [34] Квантовая механика 1.
- [35] Квантовая механика 2.

Химия

Стандарты и ГОСТы

- [36] 2.701-2008 Схемы. Виды и типы. Общие требования к выполнению
http://rtu.samgtu.ru/sites/rtu.samgtu.ru/files/GOST_ESKD_2.701-2008.pdf