Азбука ARMатурщика

учебный курс по микроконтроллерам Cortex-Mx:
Миландр 1986BE, STM32F, LPC21xx

©
Понятов Д.А. <dponyatov@gmail.com>, ИКП СГАУ,
Недяк С.П. <fvs@fet.tusur.ru>, ТУСУР

3 июля 2014 г.

Оглавление

Лабораторные работы

Часть I

Обзор семейства микроконтроллеров Cortex-Mx

Глава 1

Архитектура ARM

[1]

Архитектура ARM (Advanced RISC Machine, Acorn RISC Machine, усовершенствованная RISC-машина) — семейство лицензируемых 32-битных и 64-битных микропроцессорных ядер разработки компании ARM Limited.

Среди лицензиатов: практически все заметные разработчики цифровых электронных компонентов. Многие лицензиаты делают собственные версии ядер на базе ARM.

Значимые семейства процессоров: ARM7, ARM9, ARM11 и Cortex.

В 2007 году около 98% из более чем миллиарда мобильных телефонов, продаваемых ежегодно, были оснащены по крайней мере одним процессором ARM. По состоянию на 2009 на процессоры ARM приходилось до 90% всех встроенных 32-разрядных процессоров. Процессоры ARM широко используются в потребительской электронике — в том числе КПК, мобильных телефонах, цифровых носителях и плеерах, портативных игровых консолях, калькуляторах и компьютерных п ериферийных устройствах, таких как жесткие диски или маршрутизаторы.

Эти процессоры имеют низкое энергопотребление, поэтому находят широкое применение во встраиваемых системах и преобладают на рынке мобильных устройств, для которых данный фактор немаловажен.

В настоящее время значимыми являются несколько семейств процессоров ARM:

- ARM7 (с тактовой частотой до 60-72 МГц), предназначенные, например, для недорогих мобильных телефонов и встраиваемых решений средней производительности. В настоящее время активно вытесняется новым семейством Cortex.
- ARM9, ARM11 (с частотами до 1 ГГц) для продвинутых телефонов, карманных компьютеров и встраиваемых решений высокой производительности.
- Cortex A — новое семейство процессоров на смену ARM9 и ARM11.
- Cortex M — новое семейство процессоров на смену ARM7, также призванное занять новую для ARM нишу встраиваемых решений низкой производительности. В семействе присутствуют три значимых ядра: Cortex-M0, Cortex-M3 и Cortex-M4.

[1] копипаста: http://ru.wikipedia.org/wiki/ARM

Архитектура Существует спецификация архитектуры ARM Cortex, которая разграничивает все типы опций, которые поддерживает ARM, так как детали реализации каждого типа процессора м огут отличаться. Архитектура развивалась с течением времени, и начиная с ARMv7 были определены 3 профиля:

- A (application) для устройств, требующих высокой производительности (смартфоны, планшеты)
- R (real time) для приложений, работающих в реальном времени,
- M (microcontroller) для микроконтроллеров и недорогих встраиваемых устройств.

Режимы процессора Процессор может находиться в одном из следующих рабочих режимов:

- User mode — обычный режим выполнения программ. В этом режиме выполняется большинство программ.
- Fast Interrupt (FIQ) — режим быстрого прерывания (меньшее время с рабатывания)
- Interrupt (IRQ) — основной режим прерывания.
- System mode — защищённый режим для использования операционной системой.
- Abort mode — режим, в который процессор переходит при возникновении ошибки доступа к памяти (доступ к данным или к инструкции на этапе prefetch конвейера).
- Supervisor mode — привилегированный пользовательский режим.
- Undefined mode — режим, в который процессор входит при попытке выполнить неизвестную ему инструкцию.

Переключение режима процессора происходит при возникновении соответствующего исключения, или же модификацией регистра статуса.

Набор команд ARM Режим, в котором исполняется 32-битный набор команд.

Набор команд Thumb Для улучшения плотности кода процессоры, начиная с ARM7TDMI, снабжены режимом Thumb. В этом режиме процессор выполняет альтернативный набор 16-битных команд. Большинство из этих 16-разрядных команд переводятся в нормальные команды ARM. Уменьшение длины команды достигается за счет сокрытия некоторых операндов и ограничения возможностей адресации по сравнению с режимом полного набора команд ARM.

В режиме Thumb меньшие коды операций обладают меньшей функциональностью. Например, только ветвления могут быть условными, и многие коды операций имеют ограничение на доступ только к половине главных регистров процессора. Более короткие коды операций в целом дают большую плотность кода, хотя некоторые операции требуют дополнительных команд. В ситуациях, когда порт памяти или ширина шины ограничены 16 битами, более короткие коды операций режима Thumb становятся гораздо производительнее по сравнению с обычным 32-битным ARM кодом, так как меньший программный код придется загружать в процессор при ограниченной пропускной способности памяти.

Аппаратные средства типа Game Boy Advance, как правило, имеют небольшой объём оперативной памяти доступной с полным 32-битным информационным каналом. Но большинство операций выполняется через 16-битный или более узкий информационный канал. В этом случае имеет смысл использовать Thumb код и вручную оптимизировать некоторые тяжелые участки кода, используя переключение в режим ARM.

Набор команд Thumb2 Thumb2 — технология, стартовавшая с ARM1156 core, анонсированного в 2003 году. Он расширяет ограниченный 16-битный набор команд Thumb дополнительными 32-битными командами, чтобы задать набору команд дополнительную ширину. Цель Thumb2 — достичь плотности кода как у Thumb, и производительности как у набора команд ARM на 32 битах. Можно сказать, что в ARMv7 эта цель была достигнута.

Thumb2 расширяет как команды ARM, так и команды Thumb ещё большим количеством команд, включая управление битовым полем, табличное ветвление, условное исполнение. Новый язык «Unified Assembly Language» (UAL) поддерживает создание команд как для ARM, так и для Thumb из одного и того же исходного кода. Версии Thumb на ARMv7 выглядят как код ARM. Это требует осторожности и использования новой команды if-then, которая поддерживает исполнение до 4 последовательных команд испытываемого состояния. Во время компиляции в ARM код она игнорируется, но во время компиляции в код Thumb2 генерирует команды.

Набор команд Jazelle Jazelle — это технология, которая позволяет байткоду Java исполняться прямо в архитектуре ARM в качестве 3-го состояния исполнения (и набора команд) наряду с обычными командами ARM и режимом Thumb. Поддержка технологии Jazelle обозначается буквой «J» в названии процессора — например, ARMv5TEJ. Данная технология поддерживается начиная с архитектуры ARMv6, хотя новые ядра содержат лишь ограниченные реализации, которые не поддерживают аппаратного ускорения.

ARMv8 и набор команд ARM 64 бит В конце 2011 года была опубликована новая версия архитектуры, ARMv8. В ней появилось определение архитектуры AArch64, в которой исполняется 64-битный набор команд A64. Поддержка 32-битных команд получила название A32 и исполняется на архитектурах AArch32. Инструкции Thumb поддерживаются в режиме T32, только при использовании 32-битных архитектур. Допускается исполнение 32-битных приложений в 64-битной ОС, и запуск виртуализованной 32-битной ОС при помощи 64-битного гипервизора.[47] Applied Micro, AMD, Broadcom, Calxeda, HiSilicon, Samsung, STM и другие заявили о планах по использованию ARMv8. Ядра Cortex-A53 и Cortex-A57, поддерживающие ARMv8, были представлены компанией ARM 30 октября 2012 года.[48]

Как AArch32, так и AArch64, поддерживают VFPv3, VFPv4 и advanced SIMD (NEON). Также добавлены криптографические инструкции для работы с AES, SHA-1 и SHA-256.

Условное исполнение Одним из существенных отличий архитектуры ARM от других архитектур ЦПУ является так называемая предикация — возможность условного исполнения команд. Под «условным исполнением» здесь понимается то, что команда будет выполнена или проигнорирована в зависимости от текущего состояния флагов состояния процессора.

В то время как для других архитектур таким свойством, как правило, обладают только команды условных переходов, в архитектуру ARM была заложена возможность условного исполнения практически любой команды. Это было достигнуто добавлением в коды их инструкций особого 4-битового поля (предиката). Одно из его значений зарезервировано на то, что инструкция должна быть выполнена безусловно, а остальные кодируют то или иное сочетание условий (флагов). С одной стороны, с учётом ограниченности общей длины инструкции, это сократило число бит, доступных для кодирования смещения в командах обращения к памяти, но с другой — позволило избавляться от инструкций ветвления при генерации кода для небольших if-блоков.

Пример, обычно рассматриваемый для иллюстрации — основанный на вычитании алгоритм Евклида. В языке C он выглядит так:

алгоритм Евклида

```
while (i != j) {
        if (i > j)
            i -= j;
        else
            j -= i;
}
```

А на ассемблере ARM — так:

```
loop CMP Ri, Rj; set condition "NE" if (i != j),
                        ; "GT" if (i > j),
                        ; or "LT" if (i < j)
    SUBGT  Ri, Ri, Rj   ; if "GT" (greater than), i = i-j;
    SUBLT  Rj, Rj, Ri   ; if "LT" (less than), j = j-i;
    BNE    loop         ; if "NE" (not equal), then loop
```

Из кода видно, что использование предикации позволило полностью избежать ветвления в операторах else и then. Заметим, что если Ri и Rj равны, то ни одна из SUB инструкций не будет выполнена, полностью убирая необходимость в ветке, реализующей проверку while при каждом начале цикла, что могло быть реализовано, например, при помощи инструкции SUBLE (меньше либо равно).

Один из способов, которым уплотнённый (Thumb) код достигает большей экономии объёма — это именно удаление 4-битового предиката из всех инструкций, кроме ветвлений.

Другие особенности Другая особенность набора команд это возможность соединять сдвиги и вращения в инструкции «обработки информации» (арифметическую, логическую, движение регистр-регистр) так, что, например выражение C:

```
a += (j << 2);
```

может быть преобразовано в команду из одного слова и одного цикла в ARM:

```
ADD Ra, Ra, Rj, LSL #2
```

Это приводит к тому, что типичные программы ARM становятся плотнее, чем обычно, с меньшим доступом к памяти. Таким образом, конвейер используется гораздо более эффективно. Даже несмотря на то, что ARM работает на скоростях, которые многие бы сочли низкими, он довольно-таки легко конкурирует с многими более сложными архитектурами ЦПУ.

ARM процессор также имеет некоторые особенности, редко встречающиеся в других архитектурах RISC — такие, как адресация относительно счетчика команд (на самом деле счетчик команд ARM является одним из 16 регистров), а также пре- и пост-инкрементные режимы адресации.

Другая особенность, которую стоит отметить, это то, что некоторые ранние ARM процессоры (до ARM7TDMI), например, не имеют команд для хранения 2-байтных чисел. Таким образом, строго говоря, для них невозможно сгенерировать эффективный код, который бы вел себя так, как ожидается от объектов C, типа `volatile int16_t`.

Сопроцессоры Архитектура предоставляет способ расширения набора команд, используя сопроцессоры, которые могут быть адресованы, используя MCR, MRC, MRRC, MCRR и похожие команды. Пространство сопроцессора логически разбито на 16 сопроцессоров с номерами от 0 до 15, причем 15-й зарезервирован для некоторых типичных функций управления, типа управления кэш-памятью и операции блока управления памятью (на процессорах, в которых они есть).

В машинах на основе ARM периферийные устройства обычно подсоединяются к процессору путем сопоставления их физических регистров в памяти ARM или в памяти сопроцессора, или путем присоединения к шинам, которые в свою очередь подсоединяются к процессору. Доступ к сопроцессорам имеет большее время ожидания, поэтому некоторые периферийные устройства проектируются для доступа в обоих направлениях. В остальных случаях разработчики чипов лишь пользуются механизмом интеграции сопроцессора. Например, движок обработки изображений должен состоять из малого ядра ARM7TDMI, совмещенного с сопроцессором, который поддерживает примитивные операции по обработке элементарных кодировок HDTV.

Усовершенствованный SIMD (NEON) Расширение усовершенствованного SIMD, также называемое технологией NEON — это комбинированный 64- и 128-битный набор команд SIMD (single instruction multiple data), который обеспечивает стандартизованное ускорение для медиа приложений и приложений обработки сигнала. NEON может выполнять декодирование аудио формата mp3 на частоте процессора в 10 МГц, и может работать с речевым кодеком GSM AMR (adaptive multi-rate) на частоте более 13МГц. Он обладает внушительным набором команд, отдельными регистровыми файлами, и независимой системой исполнения на аппаратном уровне. NEON поддерживает 8-, 16-, 32-, 64-битную информацию целого типа, одинарной точности и с плавающей запятой, и работает в операциях SIMD по обработке аудио и видео (графика и игры). В NEON SIMD поддерживает до 16 операций единовременно. VFP

Технология VFP (Vector Floating Point, вектора чисел с плавающей запятой) — расширение сопроцессора в архитектуре ARM. Она производит низкозатратные вычисления над числами с плавающей запятой одинарной/двойной точности, в полной мере соответствующие стандарту ANSI/IEEE Std 754—1985 Standard for Binary Floating-Point Arithmetic. VFP производит вычисления с плавающей запятой, подходящие для широкого спектра приложений — например, для КПК, смартфонов, сжатие звука, трёхмерной графики и цифрового звука, а также принтеров и телеприставок. Архитектура VFP также поддерживает исполнение коротких векторных команд. Но, поскольку процессор выполняет операции последовательно над каждым элементом вектора, то VFP нельзя назвать истинным SIMD набором инструкций. Этот режим может быть полезен в графике и приложениях обработки сигнала, так как он позволяет уменьшить размер кода и выработку команд.

Другие сопроцессоры с плавающей запятой и/или SIMD, находящиеся в ARM процессорах включают в себя FPA, FPE, iwMMXt. Они обеспечивают ту же функциональность, что и VFP, но не совместимы с ним на уровне опкодов.

Отладка Все современные процессоры ARM включают аппаратные средства отладки, так как без них отладчики ПО не смогли бы выполнить самые базовые операции типа остановки, отступа, установка контрольных точек после перезагрузки.

Архитектура ARMv7 определяет базовые средства отладки на архитектурном уровне. К ним относятся точки останова, точки просмотра и выполнение команд в режиме отладки. Такие средства были также доступны с модулем отладки EmbeddedICE. Поддерживаются оба режима —

остановки и обзора. Реальный транспортный механизм, который используется для доступа к средствам отладки, не специфицирован архитектурно, но реализация, как правило, включает поддержку `JTAG`.

Существует отдельная архитектура отладки «с обзором ядра», которая не требуется архитектурно процессорами ARMv7.

Регистры ARM предоставляет 31 регистр общего назначения разрядностью 32 бит. В зависимости от режима и состояния процессора пользователь имеет доступ только к строго определённому набору регистров. В ARM state разработчику постоянно доступны 17 регистров:

- 13 регистров общего назначения (**R0**..**R12**).
- Stack Pointer (**R13**) — содержит указатель стека выполняемой программы.
- Link register (**R14**) — содержит адрес возврата в инструкциях ветвления.
- Program Counter (**R15**) — биты [31:1] содержат адрес выполняемой инструкции.
- Current Program Status Register (**RCPSR**) — содержит флаги, о писывающие текущее состояние процессора. Модифицируется при выполнении многих инструкций: логических, арифметических, и др.

Во всех режимах, кроме User mode и System mode, доступен также Saved Program Status Register (**SPSR**). После возникновения исключения регистр **CPSR** сохраняется в **SPSR**. Тем самым фиксируется состояние процессора (режим, состояние; флаги арифметических, логических операций, разрешения прерываний) на момент непосредственно перед прерыванием.

1.1 Cortex-Mx

Глава 2

Периферия

2.1 Таймеры

2.2 DMA

2.3 UART

2.4 CAN

2.5 USB

2.6 Ethernet

Глава 3

Производители

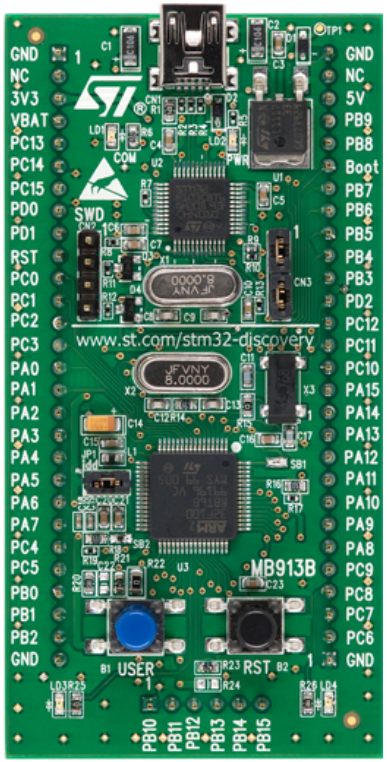
3.1 ST Microelectronics

3.1.1 STM32F0xx /Cortex-M0/

3.1.2 STM32F1xx /Cortex-M1/

Демо-плата STM32VLDISCOVERY /MB913C/

Отладочный комплект для линейки МК STM32F100 Value Line — с контроллером STM32F100RB(T6B).



www.st.com/stm32-discovery

STM32VLDISCOVERY — дешевый и быстрый способ освоение линейки МК STM32 value line. Плата содержит необходимый минимум для начинающих и опытных пользователей для быстрого старта. STM32VLDISCOVERY включает МК STM32F100RB в корпусе LQFP64, внутрисхемный отладчик/программатор ST-Link (обрезанный v.1) для отладки прошивок. В комплекте предоставляется большое количество готовых примеров прошивок в исходных текстах, для обеспечения быстрого запуска и изучения, с примерами использования светодиодов, кнопок и доступа к контактным гребенкам для подключения к другим платам или устройствам.

- МК STM32F100RB, 128 KB Flash, 8 KB RAM в корпусе LQFP64
- Наплатный ST-Link с джамперами для переключения в режим независимого программатора (с интерфейсом SWD)

- Питание от USB или внешнего источника 5 V или 3.3 V
- Может обеспечить питанием целевую схему 5 V или 3 V
- Два светодиода (зеленый и синий)
- Одна пользовательская кнопка, вторая — сброс
- Контактные гребенки для всех выводов QFP64 для быстрого подключения при прототипировании и подключения логического анализатора

3.1.3 STM32F4xx /Cortex-M4/

STM32F4DISCOVERY

3.2 ЗАО «ПКК Миландр»

Компания «Миландр» выпускает специализированные МК для спецприменений, и (относительно) дешевый вариант МК в пластиковом корпусе для обучения и неответственных применений.

Телефон: (495) 981-54-33 (8.30-17.00, отдел маркетинга и продаж *)
Факс: (495) 981-54-36
E-mail: <**info@milandr.ru**>
Сайт: http://www.milandr.ru
Форум: http://forum.milandr.ru
Адрес: 124498, г. Москва, Зеленоград, проезд 4806, дом 6

3.2.1 KP1986BE91T /Cortex-M1/

3.2.2 MDR1986Q1 /Cortex-M1/

3.3 NXP

3.3.1 LPC210x

Часть II

Программное обеспечение

Глава 4

Рабочая среда разработчика встраиваемых систем

4.1 Выбор и установка операционной системы

4.1.1 MacOS

Этот странный пока случай не рассматриваем — у меня нет под рукой Мака ☺.

4.1.2 Windows

Самый распространенный вариант. Вам придется ограничиться этим вариантом если вам не повезло с поставщиком контроллера: windows-only ПО поддержки, например софт для программатора, или внезапно библиотеки только для коммерческих компиляторов. Аналогичная ситуация будет в случае покупки какого-нибудь специфичного windows-only оборудования (лог.анализатор, измерительное оборудование или просто принтер).

Установку ОС не рассматриваем.

 Секции текста книги, зависящие от ОС, будет выделены вот так.

Для разработки встраиваемого ПО нужно поставить несколько пакетов, обеспечивающих совместимость с UNIX средами 2.

4.1.3 Linux

1. Linux удобен для разработчика,
2. если он вам не удобен, см п.1

ЛР1: Установка Debian GNU/Linux

Секции текста книги, зависящие от ОС, будет выделены вот так.

4.2 Установка инструментального программного обеспечения

ЛР2: Установка инструментального ПО для Windows

Создадим рабочий каталог, установим систему контроля версий Git5.1 и получим локальную копию проекта этой книги, содержащий кроме текста для издательской системы LaTeX еще и исходные коды библиотек, примеры кода и т.п., которые вы заходите использовать в своих проектах:

 + R http://git-scm.com/download/win

Запуститься закачка установочного пакета scm-git (`Git-1.9.4-preview20140611.exe`), после его загрузки запустите установщик,

Welcome » Next,

GNU GPL » Next,

Select components » Windows Explorer Integration » Simple Context Menu » Git GUI here » Next

Use Git and optional Unix tools from the Command Prompt » Next

Use OpenSSH » Next

Checkout Windows-style » Next

Extracting files...

Completing Setup » [ ] View ReleaseNotes » Finish

Проверим что Git правильно установился:

 + R cmd

```
C:\Documents and Settings\pda>git --version
git version 1.9.4.msysgit.0
```

Создаем каталог D:▸ARM и выгружаем текущую копию этой книги из репозитория https://github.com/ponyatov/CortexMx

```
C:\Documents and Settings\ponyatov>D:
D:\>mkdir \ARM
D:\>cd \ARM
D:\ARM>git clone --depth=1 https://github.com/ponyatov/CortexMx.git book
```

1.

ЛР3: Установка инструментального ПО для Linux

Глава 5

Управление версиями при написании ПО

5.1 Git

Глава 6

Eclipse / GNU toolchain

6.1 Утилита Make

6.2 binutils

6.2.1 Ассемблер GNU AS

6.2.2 Линкер LD

6.2.3 Утилиты работа с файлами формата ELF

6.3 Компилятор GCC

6.4 IDE Eclipse

Глава 7

Keil MDK-ARM

Глава 8

IAR Embeded Workbench

Часть III

Встраиваемый $C^{+^{+}}$

Часть IV

Литература