**Ассемблер в целом: достоинства и недостатки**

Ассемблер (от англ. assembler — сборщик) — транслятор исходного текста программы, написанной на языке ассемблера, в программу на машинном языке.

Ассемблирование может быть не первым и не последним этапом на пути получения исполнимого модуля программы. Так, многие компиляторы с языков программирования высокого уровня выдают результат в виде программы на языке ассемблера, которую в дальнейшем обрабатывает ассемблер.

Также результатом ассемблирования может быть не исполняемый, а объектный модуль, содержащий разрозненные блоки машинного кода и данных программы, из которого (или из нескольких объектных модулей) в дальнейшем с помощью редактора связей может быть получен исполнимый файл.

**Достоинства**

* Искусный программист, как правило, способен написать более эффективную программу на ассемблере, чем те, что генерируются трансляторами с языков программирования высокого уровня, то есть для программ на ассемблере характерно использование меньшего количества команд и обращений в память, что позволяет увеличить скорость и уменьшить размер программы.
* Обеспечение максимального использования специфических возможностей конкретной платформы, что также позволяет создавать более эффективные программы с меньшими затратами ресурсов.
* При программировании на ассемблере возможен непосредственный доступ к аппаратуре, в том числе портам ввода-вывода, регистрам процессора, и др.
* Язык ассемблера применяется для создания драйверов оборудования и ядра операционной системы.
* Язык ассемблера используется для создания «прошивок» BIOS.
* С помощью языка ассемблера создаются компиляторы и интерпретаторы языков высокого уровня, а также реализуется совместимость платформ.
* Существует возможность исследования других программ с отсутствующим исходным кодом с помощью дизассемблера (Дизассемблер — транслятор, преобразующий машинный код, объектный файл или библиотечные модули в текст программы на языке ассемблера.).

**Недостатки**

* Главное преимущество ассемблера практически полностью нивелируется хорошей оптимизацией в современных компиляторах языков высокого уровня.
* В силу своей машинной ориентации («низкого» уровня) человеку по сравнению с языками программирования высокого уровня сложнее читать и понимать программу, она состоит из слишком «мелких» элементов — машинных команд, соответственно усложняются программирование и отладка, растет трудоемкость, велика вероятность внесения ошибок. В значительной степени возрастает сложность совместной разработки.
* Как правило, меньшее количество доступных библиотек по сравнению с современными индустриальными языками программирования.
* Отсутствует переносимость программ на компьютеры с другой архитектурой и системой команд (кроме двоично-совместимых).

**ARM (32bit; A32) Assembler**

Для процессоров ARM существует достаточно широкий выбор компиляторов, внутренняя реализация которых зависит от непосредственно производителя данного ARM-процессора либо разработчика IDE для работы с ARM-процессорами. Официальным компилятором ARM, непосредственно от компании ARM, является ARM Compiler 6, который поддерживает компиляцию программ на языках Си и С++.

Сейчас большая часть мобильных устройств, планшетов разработаны именно на этой архитектуре процессоров. Основным преимуществом данного семейства является низкое энергопотребление, благодаря чему он часто используется в различных встроенных системах.

Каждый ARM процессор создан из следующих блоков:

* 37 регистров (из которых видимых при разработке только 17)
* Арифметико-логическое устройство (АЛУ) — выполняет арифметические и логические задачи
* Barrel shifter — устройство, созданное для перемещения блоков данных на определенное количество бит
* The CP15 — специальная система, контролирующая ARM сопроцессоры
* Декодер инструкций — занимается преобразованием инструкции в последовательность микроопераций

Разрешены следующие имена регистров:

* общие регистры r0–r12;
* указатель стека SP, регистр r13;
* регистр связи LR, регистр r14;
* счётчик команд PC, регистр r15.

Инструкция создаётся таким образом:

{метка} {инструкция|операнды} {@ комментарий}

Метка — необязательный параметр. Инструкция — непосредственно мнемоника инструкции процессору. Основные инструкции и их использование будет разобрано далее. Операнды — константы, адреса регистров, адреса в оперативной памяти. Комментарий — необязательный параметр, который не влияет на исполнение программы.

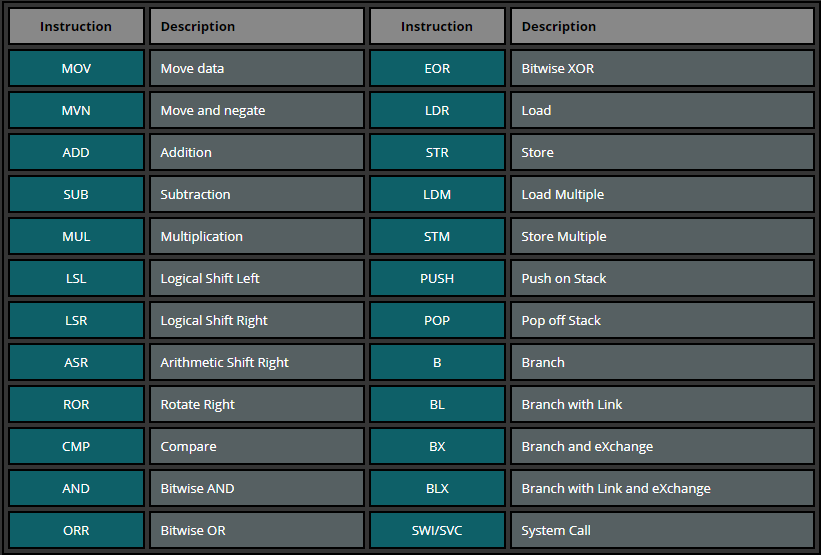


Рисунок 1 – Основные инструкции в системе ARM(A32)

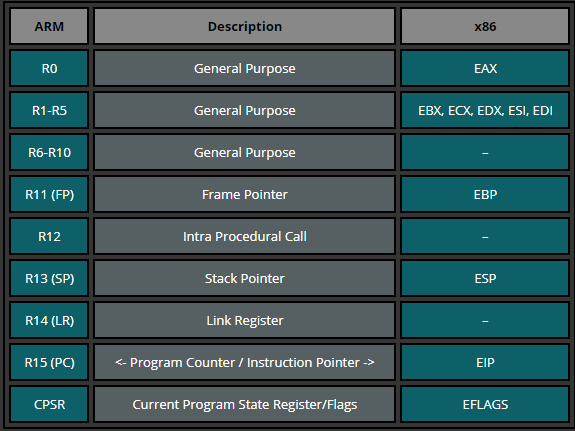


Рисунок 2 – Аналоги регистров ARM в системе x86

Для работы со стеком (SP(r13)) используются команды LDM(загрузка) и STM(сохранение).

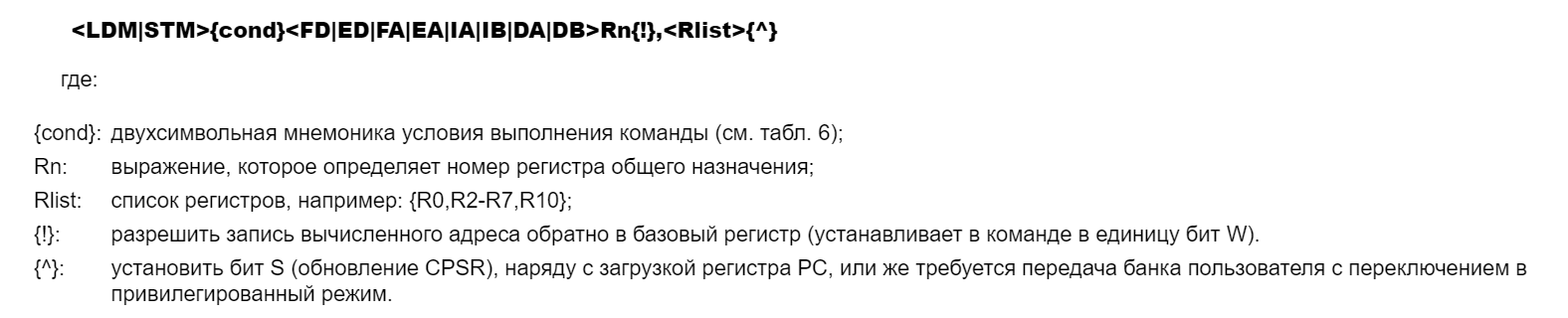


Рисунок 3 – Синтаксис инструкций LDM|STM



Рисунок 4 – Названия режимов адресации

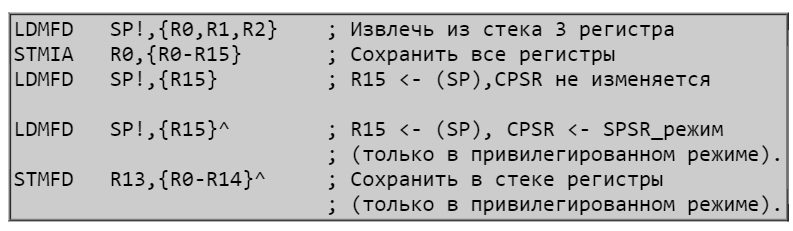


Рисунок 5 – Примеры использования инструкций LDM|STM

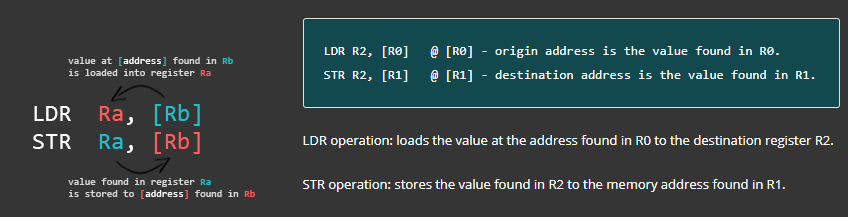


Рисунок 6 – Описание инструкций LDR|STR

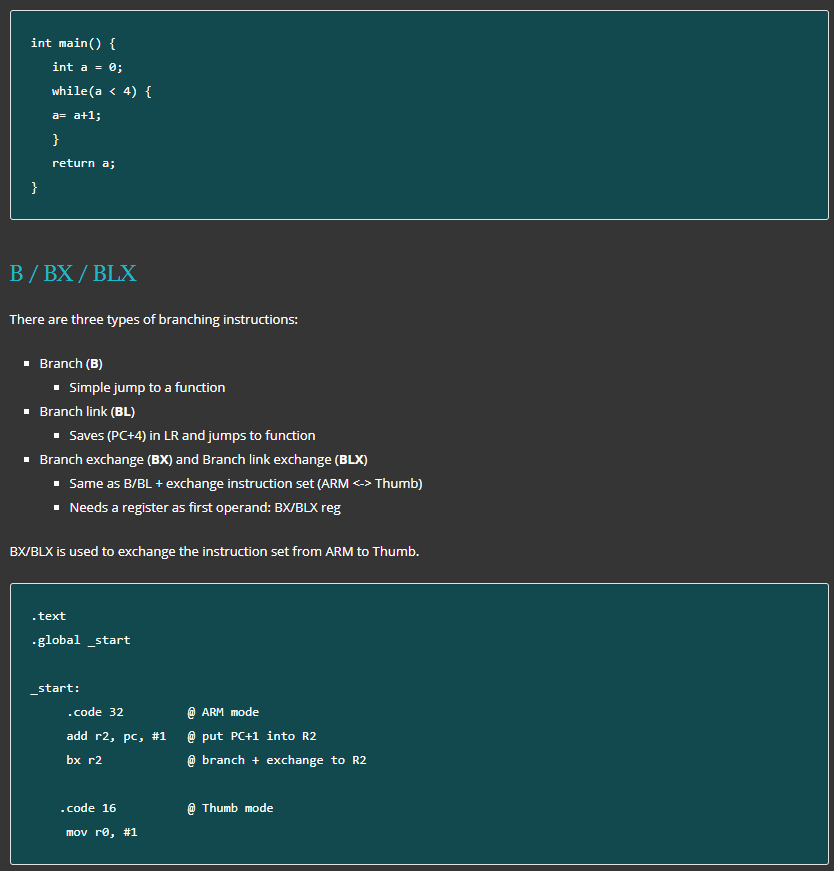


Рисунок 7 – Ветвление в ARM

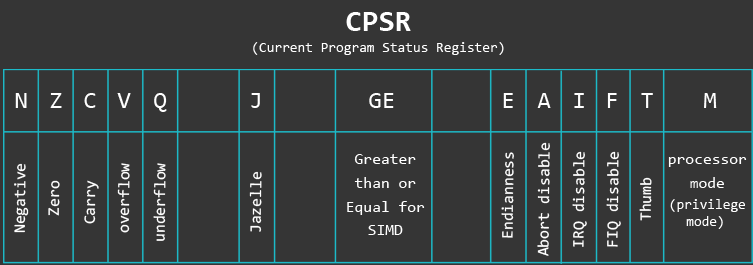


Рисунок 8 - Регистр cpsr, показывающий значение регистра текущего состояния программы



Рисунок 9 - Коды условий условного исполнения, их значения и состояние флагов (регистра cpsr)

Для ARM стандартное соглашение о вызовах следующее:

* Используется full-descending стек, то есть при занесении в стек значение регистра sp уменьшается, регистр sp указывает на первое занятое значение в стеке.
* При вызове подпрограмм значение указателя стека должно быть выровнено по 8 байтам. Поэтому, общий размер параметров, передаваемых через стек должен быть кратен 8 байтам. Общий размер локальных переменных и сохраненных регистров в стеке также должен быть кратен 8 байтам.
* Параметры в подпрограмму передаются в регистрах r0, ..., r3 и далее в стеке. Параметры заносятся в стек в обратном порядке, то есть параметр с меньшим номером находится в памяти по меньшим адресам.
* Переменная часть параметров (обозначаемая в Си ...) всегда передается на стеке.
* Значения типов long long и double должны быть выровнены по 8 байтам. Все остальные значения выравниваются по 4 байтам (даже char).
* При передаче параметров в регистрах значения типов long long и double могут размещаться только в паре регистров r0, r1 или r2, r3. В регистре с меньшим номером размещается младшая половина значения. Это может приводить к тому, что регистр r1 не будет использоваться.
* Подпрограммы должны сохранять значения регистров r4 ... r11. То есть вызывающая подпрограмма может расчитывать на то, что после возврата из подпрограммы значения регистров r4, ... r11 сохранятся.
* Регистры r0 ... r3, r12, r14 (lr) могут произвольным образом меняться.
* При входе в подпрограмму адрес возврата в вызывающую подпрограмму находится в регистр lr. Сама подпрограмма должна сохранить адрес возврата (например, в стек), если будут вызываться вложенные подпрограммы.
* Результат работы подпрограммы возвращается в регистрах r0 ... r3. То есть, если результат имеет размер не более 4 байт, он возвращается в r0. Если результат имеет размер 8 байт, он возвращается в регистрах r0 и r1. Если результат имеет размер более 16 байт, возврат значения должен выполняться другими средствами.