# Русское описание AAPCS

Что хочется в первую очередь описать.

### Общая часть

AAPCS – сокращение от (ARM Architecture Procedure Call Standard).

В названии стандарта используется термин «Procedure». Далее в русском контексте я буду использовать не только русский эквивалент «процедура», но также и термин «подпрограмма» ‑ который испольуется в терминологии, принятой среди программистов на языках Ассемблера и который в данном описании является синонимом термина «процедура». В том случае, если подпрограмма возвращает значение – результат работы, в качестве синонима может использоваться термин «функция».

AAPCS Procedure Call Standard for the ARM Architecture (this standard).

APCS ARM Procedure Call Standard (obsolete).

TPCS Thumb Procedure Call Standard (obsolete).

ATPCS ARM-Thumb Procedure Call Standard (precursor to this standard).

PIC, PID Position-independent code, position-independent data.

Routine, subroutine Подпрограмма ‑ Фрагмент программы, которую можно вызвать (передать ей управление), и по окончании выполнения подпрограммы управление вернется на инструкцию, следующую за вызовом. Routine – вызывающая часть, subroutine ‑ вызываемая подпрограмма. В англоязычных текстах для вызывающего кода иногда используют слово *caller*, а для вызываемого кода (подпрограммы) – термин *callee*

Argument, Parameter будут использоваться оба термина, как синонимы.

Variadic routine Подпрограмма с переменным числом параметров (???).

Scratch register, temporary register Регистр, используемый для хранения промежуточных результатов (обычно таким значениям не соответствует имя, упоминаемое в исходном тексте программы, а решение об использовании регистра принимает компилятор.)

Variable register, v-register – регистр, используемый для хранения значения, которому соответствует имя (переменной, может быть, и локальной для подпрограммы).

### Область действия стандарта.

Стандарт определяет, как программистам следует а) писать, б) транслировать (под)программы, чтобы они были способны работать работать совместно. Он определяет соглашение, которое должно соблюдаться вызывающей и вызываемой частями кода.

* Обязанность для *вызывающего* создать состояние программы, в котором вызываемый может успешно начать исполняться.
* Обязанность для *вызываемого* после вызова сохранять программное состояние, которое позволит *вызывающему* успешно продолжиться после возврата из вызываемого.
* Права для вызываемого изменять состояние программы в некоторых пределах (рамках).

Данный стандарт {определяет/устанавливает/создает} основу для семейства соглашений о вызовах подпрограмм. {Экземпляры/представители} семейства могут способствовать достижению различных (часто альтернативных/взаимно-исключающих) целей, среди которых основные:  
‑ минимизация размера кода;  
‑ повышение производительности (скорости работы);  
‑ функциональность (в частности, легкость отладки, проверки во время исполнения и т.п.)

Некоторые свойства каждого из вариантов – например, допустимое использование регистра R9 – могут определяться/определяются средой исполнения. Таким образом:

1. It is possible for code complying strictly with the base standard to be PCS compatible with each of the variants.
2. It is unusual for code complying with a variant to be compatible with code complying with any other variant.
3. Code complying with a variant, or with the base standard, is not guaranteed to be compatible with an execution environment that requires those standards. An execution environment may make further demands beyond the scope of the procedure call standard.

* Код может строго следовать стандарту (Соглашению о Вызовах Процедур), и, как следствие, быть совместимым с любым из вариантов.
* Если код следует требованиям, реализованным в одном из вариантов, это, как правило, не означает полного соответствия всем требованиям стандарта.
* Полное удовлетворение требованиям стандарта еще не гарантирует работоспособности в конкретной среде исполнения, т.е. среда исполнения может накладывать дополнительные требования (ограничения) сверх тех, что оговорены стандартом.

Стандарт содержит четыре секции (части), в которых оговаривается:

* Требования к размещению элементов данных (???)
* Структура стека (желательно более узкий термин: что-то вроде «структура и назначение областей/элементов стека, а может быть, стекового кадра/кадров)
* Варианты/вариации, которые могут быть использованы при разработке архитектурных расширений процессоров (???), или случаи, когда среда исполнения накладывает ограничения на модель адресации.
* The C and C++ language bindings for plain data types.

This specification does not standardize the representation of publicly visible C++-language entities that are not also C language entities (these are described in CPPABI) and it places no requirements on the representation of language entities that are not visible across public interfaces.

### Цели разработки стандарта

Стандарт разрабатывался, чтобы

* в равной степени обеспечить поддержку режимов процессора Thumb-state и ARM-state
* обеспечить взаимодействие этих режимов
* обеспечить эффективное исполнение кода на высокопроизводительных реализациях архитектурыARM
* четко разграничить обязательные требования от вариаций, которые допустимы при конкретных реализациях архитектуры.
* минимизировать (возможную) несовместимость на уровне двоичного кода (т.е. результата трансляции). (Почему «минимизировать», а не «свести к нулю»???).

Совместимость / или соответствие (Conformance )

The AAPCS defines how separately compiled and separately assembled routines can work together. There is an externally visible interface between such routines. It is common that not all the externally visible interfaces to software are intended to be publicly visible or open to arbitrary use. In effect, there is a mismatch between the machine-level concept of external visibility—defined rigorously by an object code format—and a higher level, application-oriented concept of external visibility—which is system-specific or application-specific.

…….

Соответствие стандарту требует, чтобы

* в любой момент границы стека и выравнивание элементов стека {были наблюдаемы /или/ соблюдались} (are observed )
* соблюдались правила использования IP (??? IP это The Intra-Procedure-call scratch register R12 )

### Регистры процессора

В таблице указаны регистры процессора архитектур ARM7, ARM9, ARM11, обозначения для этих регистров и их назначение, предписываемое стандартом.



Обозначения регистров могут использоваться не только в поясняющих текстах, но также и в инструкциях языка Ассемблера. Там допускается использовать как верхний регистр (прописные буквы) так и нижний (строчные буквы).

В данном стандарте обозначения большими буквами (например, PC, LR, SP, IP) используются в том случае, если регистру предписана стандартом определенная (фиксированная роль/предназначение) в ходе вызова процедуры. Кроме того, программе доступен еще регистр состояний CPSR (that is available for use in conforming code).

Первые четыре регистра r0-r3 (синоним a1-a4) используются для передачи в процедуру значений параметров и для возврата результата, формируемого функцией.

Регистр r12 (IP ‑ Intra-Procedure-call scratch register) может быть использован в качестве регистра промежуточного хранения (scratch register)

В любом из двух состояний (States) команда вызова BL не способна совершить переход в любую точку адресного пространства, в этой команде используется относительная адресация со 26-битовым смещением. Для такого перехода может быть использована связка команды BL и команды BX, например, так:

LDR r12, =#Proc ; загрузить стартовый адрес процедуры в r12

BL L1 ; команда вызова, расстояние до L1 не больше 226

; на эту команду будет возврат

… …

L1: BX [r12] ; дополнительная команда (veneer)

… …

Proc: ; стартовый адрес процедуры

Назначение регистра r9 зависит от «платформы» (что тут имеется в виду? возможно, версия процессора, а, может быть, среда исполнения). В стандарте приводится такой пример:

A virtual platform may assign any role to this register and must document this usage. For example, it may designate it as the static base (SB) in a position-independent data model, or it may designate it as the thread register (TR) in an environment with thread-local storage. The usage of this register may require that the value held is persistent across all calls. A virtual platform that has no need for such a special register may designate r9 as an additional callee-saved variable register, v6.

Виртуальная платформа (что имеется в виду? среда исполнения, операционная система? ) может назначить регистру r9 некую роль, но использование этого регистра должно быть документировано. Например, в этом регистре может содержаться «статический базовый адрес», позволяющий организовать позиционную независимость кода. Или в этом регистре (TR – thread register) может содержаться базовый адрес области данных, локальных для определенного потока исполнения. Использование этого регистра может требовать сохранения его содержимого неизменным при всех вызовах процедур. Если же среда исполнения такова, что этого не требуется, она (среда) может использовать r9 для других целей, например держать в нем локальную переменную для вызываемой процедуры.

Регистры r4-r8, r10 и r11 обычно используются для хранения локальных переменных программы. Поэтому вызванная процедура должна сохранять их содержимое (а, может быть и содержимое r9) перед тем, как их использовать для своих целей.

Во всех вариантах стандарта регистры r12-r15 имеют специальное предназначение и не могут использоваться для иных целей.

##### Поля регистра состояний CPSR

Стандарт предписывает свойства отдельных полей регистра состояния (CPSR Current Program Status Register). Некоторые поля присутствуют и используются только в некоторых архитектурных версиях.

##### Работа с величинами, размер которых превышает 32 бита

Базовые типы с длиной более 32 бит, могут передаваться как параметры в процедуру, а также возвращаться при выходе из процедуры. Если для этого используются регистры процессора, то необходимо руководствоваться следующими правилами:

64-битовые величины должны передаваться в парах регистров (r0, r1) или (r2, r3) так, как если бы эти регистры были загружены одной командой LDM.

128-битовая величина может передаваться через четверку регистров (r0,…r3) так, как если бы эти регистры были загружены из памяти одной командой LDM.

### 5.2 Процессы, память и стек.

Стандарт вызова процедур применим к одному единственному потоку исполняемых команд(говоря о таком потоке команд далее будет использоваться термин «процесс»). Процесс характеризуется текущим состоянием (контекстом), который определяется содержимым регистров процессора и содержимым областей памяти, к которым имеет доступ код процесса. Размер и положение доступных областей памяти (при доступе к которым не возникает исключительной ситуации, т.е. не срабатывают аппаратные механизмы защиты) могут изменяться в процессе исполнения кода, принадлежащего процессу.

Области памяти, доступные процессу, подразделяют на следующие пять категорий/разновидностей:

1. Область кода, которая должна быть доступна по чтению (на этапе выборки команды), но не обязательно – по записи. Даже наоборот, недоступность по записи предотвращает случайное повреждение программы.
2. Статических (константных) данных, которая доступна только по чтению. Такие элементы данных обычно помещают в исполняемый файл, который формируется во время трансляции. При загрузке код и константные данные загружаются в память. Некоторые из этих константных данных представляют собой инициализирующие значения переменных.
3. Данные, доступные программе, как по чтению, так и по записи (т.е. обычные переменные). Если программист инициализировал некоторые из переменных одновременно с определением, то компилятор помещает инициализирующие значения в область (2) и в «прологе» (это код, который выполняется до main) копирует эти значения из области (2) в область (3) – т.е. присваивает переменным начальные значения.
4. Область динамически распределяемой памяти (heap).
5. Область стека

Доступность областей кода и константных данных «только по чтению» может быть обеспечена либо использованием «постоянной» памяти, либо использованием аппаратных механизмов ограничения доступа.

#### 5.2.1 Стек

Стек должен занимать обязательно непрерывную область в пространстве адресов. Стек используется для передачи параметров в процедуру (но не для возврата результата). Кроме того, в стеке выделяется место для локальных переменных процедуры (имеющих ограниченное время существования).

В архитектуре ARM стек растет в сторону уменьшения адресов, а указатель стека R13 содержит адрес последнего занятого элемента (full-descending stack). Для стека (должны быть) определены его *база* (*stack-base*) ‑ начальный адрес или *дно стека*, и *верхняя граница области стека* (*stack-limit*), хотя прикладная программа может не иметь возможности определить каждую из этих величин.

Стек может иметь фиксированный размер, либо быть динамически расширяемым, путем уменьшения значения *stack-limit*.

Правила обращения со стеком включают две части:   
а) универсальные (действующие в любых обстоятельствах)  
б) которые должны соблюдаться при разработке/использовании *внешних интерфейсов (public interface)*

##### 5.2.1.2 Универсальные ограничения на параметры стека

В любой момент выполнения:

* *stack-limit* < SP <= *stack-base* , т.е. указатель стека должен указывать внутрь области стека.
* SP mod 4 = 0 . Содержимое указателя стека должно быть кратно 4 (длине слова)
* Процесс может осуществлять доступ (по записи или по чтению) только к области адресов в интервале [SP, *stack-base*-1]. Здесь SP это содержимое регистра r13.

##### 5.2.1.2. Ограничения для внешних интерфейсов

* SP mod 8 = 0 . Содержимое указателя стека должно быть кратно 8.

#### 5.3. Вызов подпрограммы

В соглашении о вызовах написано гораздо больше, чем в этом коротком резюме на русском. Это потому, что в источнике рассматривается гораздо больше тонкостей и особенностей для нескольких разных архитектурных версий. Когда-нибудь, может быть, поговорим и о тонкостях.

Для вызова подпрограммы с запоминанием адреса возврата в системе команд есть команда BL (смотрите описание в справочнике и не забудьте, что наша архитектурная версия – 4Т). Выполняя эту команду процессор ARM автоматически запоминает адрес возврата в регистре LR (r14), после чего загружает в PC адрес первой команды подпрограммы (этот адрес получается в результате коррекции содержимого счетчика команд при выполнении команды вызова ПП).

Для передачи параметров в ПП используются регистры r0 – r3 (может быть, только некоторые из них). Это означает, что к моменту вызова ПП те из перечисленных регистров, которые используются для передачи параметров, должны быть освобождены.

В этих же регистрах (или в некоторых из них принято возвращать из ПП результат). Если параметров много (не помещаются в указанных четырех регистрах), то оставшаяся часть параметров должна передаваться через стек.

Для параметров различных типов указывается, где именно и как именно эти параметры должны размещаться в регистрах, или, каким образом (в каком регистре или регистрах и в каком порядке) должно размещаться возвращаемое подпрограммой значение.