Международный союз электросвязи

МСЭ-Т J.1013

(04/2020)

СЕКТОР СТАНДАРТИЗАЦИИ
ЭЛЕКТРОСВЯЗИ МСЭ

СЕРИЯ J: КАБЕЛЬНЫЕ СЕТИ И ПЕРЕДАЧА СИГНАЛОВ ТЕЛЕВИЗИОННЫХ И ЗВУКОВЫХ ПРОГРАММ И ДРУГИХ МУЛЬТИМЕДИЙНЫХ СИГНАЛОВ

Условный доступ и защита – Заменяемые встроенные решения для обеспечения условного доступа и управления цифровыми правами

Встроенный общий интерфейс для заменяемых решений CA/DRM; виртуальная машина

Рекомендация МСЭ-Т J.1013

ITU-T

Международный союз электросвязи

Рекомендация МСЭ-Т J.1013

Встроенный общий интерфейс для заменяемых решений CA/DRM; виртуальная машина

Резюме

Рекомендация МСЭ-Т J.1013 – это часть состоящей из нескольких частей документации, охватывающая спецификацию виртуальной машины встроенного общего интерфейса (ECI) для заменяемых решений для обеспечения условного доступа и управления цифровыми правами (CA/DRM).

Настоящая Рекомендация МСЭ-Т является переложением стандарта ETSI GS ECI 001-4 и представляет собой результат сотрудничества ИК9 МСЭ-Т и ETSI ISG ECI. Внесены незначительные изменения в пункт 7.3.7.1.

Хронологическая справка

Издание	Рекомендация	Утверждение	Исследовательская комиссия	Уникальный идентификатор*
1.0	МСЭ-Т J.1013	23.04.2020 года	9-я	11.1002/1000/13574

Ключевые слова

CA, DRM, замена.

* Для получения доступа к Рекомендации наберите в адресном поле вашего браузера URL http://handle.itu.int/, после которого следует уникальный идентификатор Рекомендации. Например, http://handle.itu.int/11.1002/1000/11830-en.

ПРЕДИСЛОВИЕ

Международный союз электросвязи (МСЭ) является специализированным учреждением Организации Объединенных Наций в области электросвязи и информационно-коммуникационных технологий (ИКТ). Сектор стандартизации электросвязи МСЭ (МСЭ-Т) – постоянный орган МСЭ. МСЭ-Т отвечает за изучение технических, эксплуатационных и тарифных вопросов и за выпуск Рекомендаций по ним с целью стандартизации электросвязи на всемирной основе.

На Всемирной ассамблее по стандартизации электросвязи (ВАСЭ), которая проводится каждые четыре года, определяются темы для изучения исследовательскими комиссиями МСЭ-Т, которые, в свою очередь, вырабатывают Рекомендации по этим темам.

Утверждение Рекомендаций МСЭ-Т осуществляется в соответствии с процедурой, изложенной в Резолюции 1 ВАСЭ.

В некоторых областях информационных технологий, которые входят в компетенцию МСЭ-Т, необходимые стандарты разрабатываются на основе сотрудничества с ИСО и МЭК.

ПРИМЕЧАНИЕ

В настоящей Рекомендации термин "администрация" используется для краткости и обозначает как администрацию электросвязи, так и признанную эксплуатационную организацию.

Соблюдение положений данной Рекомендации осуществляется на добровольной основе. Однако данная Рекомендация может содержать некоторые обязательные положения (например, для обеспечения функциональной совместимости или возможности применения), и в таком случае соблюдение Рекомендации достигается при выполнении всех указанных положений. Для выражения требований используются слова "следует", "должен" (shall) или некоторые другие обязывающие выражения, такие как "обязан" (must), а также их отрицательные формы. Употребление таких слов не означает, что от какой-либо стороны требуется соблюдение положений данной Рекомендации.

ПРАВА ИНТЕЛЛЕКТУАЛЬНОЙ СОБСТВЕННОСТИ

МСЭ обращает внимание на вероятность того, что практическое применение или выполнение настоящей Рекомендации может включать использование заявленного права интеллектуальной собственности. МСЭ не занимает какую бы то ни было позицию относительно подтверждения, действительности или применимости заявленных прав интеллектуальной собственности независимо от того, доказываются ли такие права членами МСЭ или другими сторонами, не относящимися к процессу разработки Рекомендации.

На момент утверждения настоящей Рекомендации МСЭ получил извещения об интеллектуальной собственности, защищенной патентами, которые могут потребоваться для выполнения настоящей Рекомендации. Однако те, кто будет применять Рекомендацию, должны иметь в виду, что вышесказанное может не отражать самую последнюю информацию, и поэтому им настоятельно рекомендуется обращаться к патентной базе данных БСЭ по адресу: http://www.itu.int/ITU-T/ipr/.

СОДЕРЖАНИЕ

Стр.

Введение

Настоящая Рекомендация МСЭ-Т[1] является переложением стандарта ETCИ [b-ETSI GS ECI 001-4] и представляет собой результат сотрудничества ИК9 МСЭ-Т и ETSI ISG ECI. Внесены незначительные изменения в пункт 7.3.7.1.

Настоящая Рекомендация призвана способствовать повышению функциональной совместимости и развитию конкуренции в сфере услуг электронной связи – в частности, на рынке радиовещательных и аудиовизуальных устройств. В зависимости от ситуации в Государствах – Членах это может относиться и к другим существующим технологиям.

В настоящем документе дается описание концепции виртуальной машины (ВМ), которая выполняется в изолированной программной среде и предоставляет для использования ряд команд и функций системных вызовов. ВМ предназначена для работы в разнообразных средах. Она взаимодействует с другими приложениями, работающими на том же физическом компьютере посредством известных интерфейсов, а также поддерживает собственную систему команд и модульный механизм для выполнения программных элементов, написанных в **собственном коде**[2] центрального процессора **хоста ECI** и взаимодействующих с аппаратным обеспечением и другими элементами среды **хоста ECI**. Это позволяет ВМ выполнять легко обновляемый код, который может обеспечить широкий круг потенциальных защищенных приложений, включая практическую реализацию клиентов CA/DRM.

---

1 В Дополнении II определен ряд тематических областей для дальнейшей разработки.

2 Полужирным шрифтом в тексте настоящей Рекомендации выделены термины, определения которых в контексте встроенного общего интерфейса могут отличаться от общеупотребительных.

Рекомендация МСЭ-Т J.1013

Встроенный общий интерфейс для заменяемых решений CA/DRM; виртуальная машина

1 Сфера применения

В настоящей Рекомендации определяется виртуальная машина, предназначенная для включения в реализации цифровых телевизионных приемников и абонентских приставок, которая способна обеспечить защищенную среду для запуска приложений ядра системы условного доступа или клиента системы управления цифровыми правами. Цель заключается в том, чтобы обеспечить унифицированную среду выполнения, в которой такие клиенты могли бы работать, зная о соблюдении минимальных требований к производительности **хоста ECI**, наличии стандартного API для извлечения основных данных, касающихся безопасности, непосредственно из контента (то есть инкапсулированных вместе с контентом данных или через внешние сети (например, интернет), а также о возможности стандартизированного доступа к ресурсам из среды **хоста ECI**. См. также [b-ITU-T J.1010] и [b-ITU-T J.1011].

Присутствие и использование ВМ позволяют произвольно менять клиентов CA/DRM и поддерживать множество одновременно существующих экземпляров таких клиентов на **хостах ECI**. Благодаря этому пользователи и операторы не привязаны к конкретному поставщику систем защиты контента (СР), что облегчает использование различных типов решений в области обеспечения безопасности, наиболее подходящих для конкретного типа контента. Для поставщиков систем защиты контента ВМ обеспечивает известную платформу выполнения, не требующую отдельной интеграции с системами каждого конкретного поставщика устройств **хоста ECI**.

2 Справочные документы

Указанные ниже Рекомендации МСЭ-Т и другие справочные документы содержат положения, которые путем ссылки на них в данном тексте составляют положения настоящей Рекомендации. На момент публикации указанные издания были действующими. Все Рекомендации и другие источники могут подвергаться пересмотру; поэтому всем пользователям данной Рекомендации предлагается изучить возможность применения последнего издания Рекомендаций и других справочных документов, перечисленных ниже. Список действующих в настоящее время Рекомендаций МСЭ-Т регулярно публикуется. Ссылка на документ в данной Рекомендации не придает ему как отдельному документу статус Рекомендации.

[ITU-T J.1012] Recommendation ITU-T J.1012 (2020), *Embedded common interface for exchangeable CA/DRM solutions; CA/DRM container, loader, interfaces, revocation.*

[ETSI GS ECI 001-4] ETSI GS ECI 001-4 V1.1.1 (2017), *Embedded Common Interface (ECI) for exchangeable CA/DRM solutions; Part 4: The Virtual Machine.*

3 Определения

3.1 Термины, определенные в других документах

Отсутствуют.

3.2 Термины, определенные в настоящей Рекомендации

В настоящей Рекомендации определяются следующие термины.

3.2.1 байт-код (bytecode) – код **клиента ECI** (обычно ядра системы условного доступа или клиента системы управления цифровыми правами), который выполняется виртуальной машиной (ВМ).

3.2.2 оборудование в помещении пользователя (Customer Premises Equipment (CPE)) – устройство пользователя, которое предоставляет определенные в спецификации встроенного общего интерфейса (**ECI**) функции шифрования и расшифровки.

3.2.3 ECI (встроенный CI) (ECI (embedded CI)) – архитектура и система, определяемая в ETSI ISG "Embedded CI", которая позволяет разрабатывать и внедрять программно заменяемые **клиенты ECI** в комплекты оборудования в помещении пользователя (**CPE**), тем самым обеспечивая совместимость устройств **CPE** с интерфейсом **ECI**.

3.2.4 клиент ECI (клиент встроенного CI) (ECI client (embedded CI client)) – практическая реализация клиента условного доступа/управления цифровыми правами (CA/DRM), соответствующая спецификациям встроенного общего интерфейса (**ECI**).

3.2.5 хост ECI (ECI host) – аппаратная и программная система в составе **CPE**, которая реализует ряд относящихся к **ECI** функций и связана с **клиентом ECI** специальными интерфейсами.

3.2.6 собственный код (native Code) – программный код в собственной системе исполняемых команд процессора **хоста ECI**.

3.2.7 экземпляр виртуальной машины (VM instance) – созданный **хостом ECI** экземпляр виртуальной машины (ВМ), выступающий для **клиента ECI** в качестве среды выполнения, в которой он работает.

4 Сокращения и акронимы

В настоящей Рекомендации используются следующие сокращения и акронимы.

API	Application Programming Interface		Интерфейс прикладного программирования
CA	Conditional Access		Условный доступ
CAS	Conditional Access System		Система условного доступа
CI	Common Interface		Общий интерфейс
CP	Content Protection		Защита контента
CPE	Customer Premises Equipment		Оборудование в помещении пользователя
CPU	Central Processing Unit	ЦП	Центральный процессор
DRM	Digital Rights Management		Управление цифровыми правами
ECI	Embedded Common Interface		Встроенный общий интерфейс
ECP	Enhanced Content Protection		Система расширенной защиты контента
ELF	Executable and Linkable Format		Формат исполняемых и компонуемых файлов
EPG	Electronic Programme Guide		Электронная программа телепередач
ID	Identification/Identity/Identifier		Идентификация/идентификатор
OS	Operating System	ОС	Операционная система
OTT	Over The Top		Технология "Over The Top"
PC	Program Counter		Счетчик команд
POSIX	Portable Operating System Interface		Интерфейс переносимых операционных систем
RISC	Reduced Instruction Set Computer		Компьютер с сокращенным набором команд
VM	Virtual Machine	ВМ	Виртуальная машина

5 Соглашения

Использование в настоящей Рекомендации терминов, выделенных полужирным шрифтом и начинающихся с прописной буквы*, указывает на то, что данные термины имеют особое значение, связанное с ECI, которое может отличаться от общеупотребительного.

6 Концептуальные принципы

6.1 Виртуальная машина как центральный процессор

По существу, виртуальная машина (ВМ) состоит из виртуального центрального процессора (ЦП), имеющего собственную память для хранения кода и данных, и набора системных интерфейсов, обеспечивающих доступ к аппаратным функциям **хоста ECI**. Эмулируемый ЦП выполняет код подобно виртуальному 32-битовому ЦП, и этот код называется в настоящей Рекомендации **байт-кодом**. Поскольку ВМ эмулирует процессор RISC ("компьютер с сокращенным набором команд") общего назначения, она способна выполнять самые разнообразные приложения.

6.2 Характеристики виртуальной машины

Виртуальная машина обеспечивает однопроцессную, однопоточную среду.

Интерфейс к аппаратному обеспечению **хоста ECI** и другим его функциям предоставляется в форме стандартной библиотеки вызовов, называемых системными вызовами (SYSCALL). Команда SYSCALL – одна из модифицируемых команд ВМ, которая обычно исполняется после подготовки требуемых параметров для библиотечной подпрограммы (то есть их передачи в регистры ВМ).

С помощью этой команды обеспечивается все взаимодействие между **клиентом ECI** и **хостом ECI**. **Прерывания** в архитектуре не предусмотрены, и после запуска **клиента ECI** программа выполняется до конца. Таким образом, вызовы к самой виртуальной машине невозможны. Это несколько ограничивает гибкость, зато обеспечивает усовершенствованное управление процессом исполнения виртуальной машины (устойчивость ее работы), устранение состояний гонки, вмешательство в критичные по времени операции и т. д.

Соответственно, передавать данные или сообщения **клиенту ECI**, выполняемому в виртуальной машине, можно только по запросам, которые он делает путем активизации соответствующих SYSCALL.

6.3 Изоляция отдельных клиентов ECI

Клиент ECI выполняется в виртуальной машине, которая существует в виде приложения, работающего в прошивке **хоста ECI**. Должна быть обеспечена возможность запуска нескольких экземпляров виртуальной машины, в каждой из которых может работать свой **клиент ECI**. Отсюда вытекают три основных требования к среде выполнения **хоста ECI**.

1) Изоляция отдельных клиентов ECI: операционная система (ОС) должна выделять каждому экземпляру виртуальной машины достаточное количество ресурсов, чтобы требования к производительности выполнялись всеми одновременно работающими экземплярами. Предлагаемые значения требуемых параметров производительности даны в [b-ITU-T J-Suppl.7].
2) Библиотеки, определенные в пункте 8 и Приложении C, должны быть полностью реентерабельными или реализовываться независимо в каждом экземпляре виртуальной машины.
3) Операционная система и ВМ должны обеспечить невозможность обмена информацией между работающими **клиентами ECI** и внешним миром (включая других **клиентов ECI**) иными способами, кроме тех, которые прямо предусмотрены для этой цели в интерфейсе SYSCALL. В частности, это подразумевает необходимость заблаговременно очищать от предыдущего контента всю память, отображаемую в пространство данных экземпляра **виртуальной машины**, и предотвращать любые попытки использования исключительных ситуаций в ВМ

* В русской версии данные термины только выделяются полужирным шрифтом – прим. перев.

для запуска несанкционированного поведения. Это также подразумевает невозможность для **клиента ECI** заменять собственный **байт-код**. Наконец, это подразумевает, что **хост ECI** и ВМ должны производить все необходимые проверки, чтобы не позволять **клиенту ECI** инициировать в реализациях **хоста ECI** или ВМ непредусмотренное поведение, которое может, например, прямо или косвенно давать **клиенту ECI** возможность манипулировать **хостом ECI**.

6.4 Определение виртуальной машины

В последующих пунктах настоящей Рекомендации явным образом определяются следующие аспекты виртуальной машины:

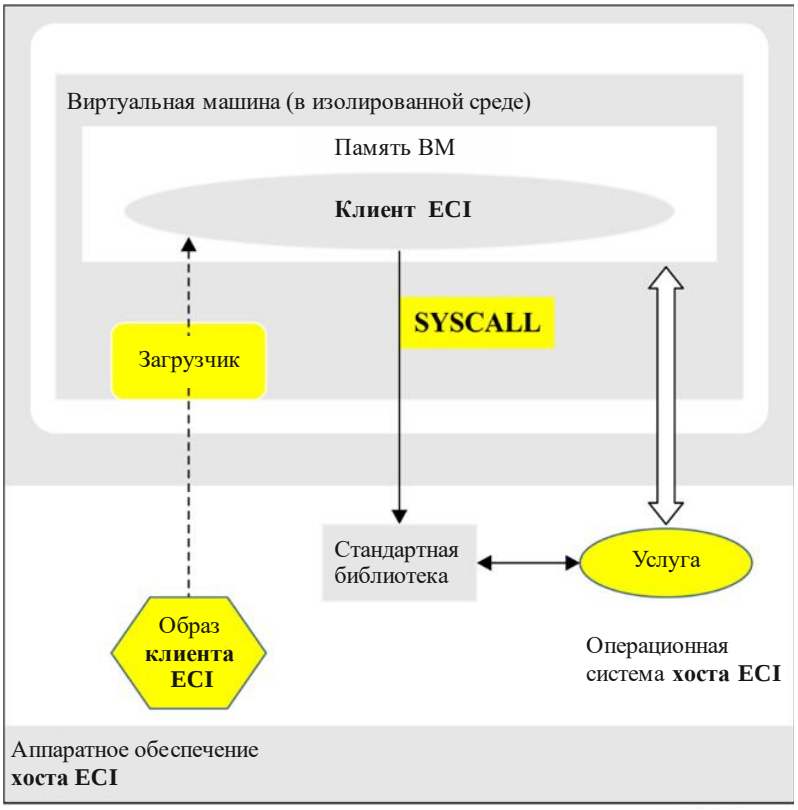
1) техническая архитектура ВМ;
2) система инструкций ВМ;
3) интерфейс **хоста ECI**.

6.5 Загрузчик клиента ECI

Для запуска **клиента ECI** необходимо сначала загрузить **байт-код** в кодовое пространство памяти ВМ и инициализировать ее пространство данных. В разделе 9 рассматриваются некоторые конкретные аспекты формата контейнера **клиента ECI** и инициализация ВМ.

7 Виртуальная машина

7.1 Среда выполнения



J.1013(20)_F01

Рисунок 1 – Среда хоста виртуальной машины

Как показано на рисунке 1, ВМ выполняется в программной среде, обеспечивающей изоляцию от операционной системы **хоста ECI**, других экземпляров виртуальной машины и любых других приложений, выполняемых **хостом ECI**.

ВМ состоит из собственного приложения **хоста ECI** и выделенной ему памяти, а также интерфейсной библиотеки и загрузчика для установки **байт-кода клиента ECI**. Интерфейсная библиотека предоставляет **клиенту ECI** доступ к функциям операционной системы и аппаратного обеспечения **хоста ECI**, а также к другим работающим на **хосте ECI** приложениям, с которыми **клиенту ECI** может понадобиться взаимодействовать. В качестве типичного примера можно привести взаимодействие с приложением электронной программы телепередач, при котором требуется статус авторизации для показа определенного контента пользователю.

7.2 Архитектура виртуальной машины

7.2.1 Архитектура ЦП

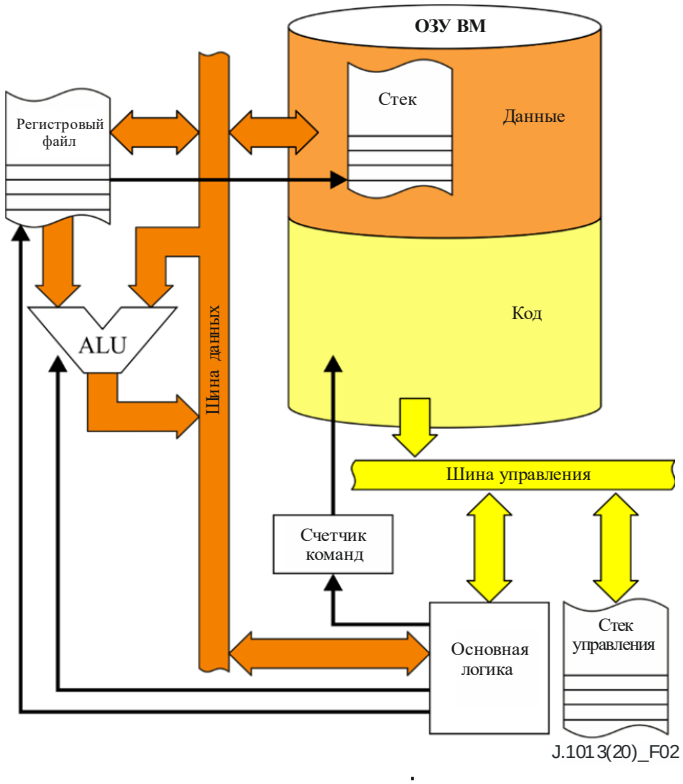


Рисунок 2 – Архитектура виртуального процессора

На рисунке 2 показана архитектура ЦП виртуальной машины. ВМ представляет собой регистровую вычислительную машину со следующими характеристиками.

– Регистровый файл с 32-битовыми регистрами общего назначения. Вся совокупность регистров поделена на регистровые окна. В каждом регистровом окне 32 регистра. Последние 16 регистров каждого окна перекрываются с первыми 16 регистрами следующего окна. Два таких регистра в каждом окне служат указателем стека и указателем кадра. Общее число регистров в файле равняется REGISTER_FILE_SIZE (определение см. в Приложении A).

– Гарвардская архитектура ЦП. Данные хранятся в области памяти со сплошной 32-битовой адресацией. Код хранится в неадресуемой области памяти, доступной только для чтения.

– Отдельный стек управления обеспечивает хранение адресов возврата. Содержимое этого стека недоступно для **байт-кода** или внешних приложений. Максимальное число адресов возврата, которое может хранить стек, равняется CONTROL_STACK_SIZE (см. Приложение A).

– Команды загрузки и хранения для таких типов данных, как знаковые и беззнаковые байты, полуслова и слова (8, 16 и 32 байта соответственно).

– Система команд с обширным набором команд обработки данных, ориентированных на конкретную область применения.

– Собственный порядок байтов для эффективной загрузки и хранения независимо от порядка следования байтов используемой платформы. Для простых типов данных применяется естественное выравнивание (по размеру соответствующего типа), чтобы обеспечить максимальную переносимость **байт-кода**. Другими словами, адрес полуслова в памяти всегда четный, а адрес слова всегда кратен четырем.

– Команда системного вызова SYSCALL, которую можно использовать для реализации системных услуг. Это также позволяет дополнять виртуальную машину встроенными функциями, например, для выполнения часто встречающихся операций обработки данных в собственных кодах.

– Страничная память с поддержкой фрагментированных областей памяти. Позволяет отображать память хоста в область памяти ВМ.

7.2.2 Регистры

В каждом регистровом окне видны 32 регистра – от R0 до R31. Два регистра зарезервированы для особых целей: R0 – указатель кадра, а R16 – указатель стека. Использование этих регистров более подробно описывается ниже.

При входе в функцию команда ENTER сдвигает регистровое окно на шестнадцать регистров вверх. В результате старый указатель стека становится новым указателем кадра, а также появляются новый указатель стека и еще пятнадцать регистров. Для инициализации нового указателя стека из значения указателя кадра вычитается размер кадра, предоставленный командой ENTER.

По команде RETURN происходит обратный процесс. Эта команда сдвигает регистровое окно на шестнадцать регистров вниз, в результате чего восстанавливаются старые указатель кадра и указатель стека.

Поскольку исходные регистры R0–R15 недоступны из вызываемой подпрограммы, она автоматически сохраняет их. В связи с тем, что адрес возврата сохраняется в отдельном стеке управления, для сохраняемых вызываемой подпрограммой регистров и адресов возврата не используется стек данных.

Истинное число регистров ограниченно, поэтому установлено максимальное значение глубины стека управления **клиента ECI** – CONTROL_STACK_SIZE. При превышении этого значения работа программы ВМ принудительно завершается. Число регистров и соответствующая глубина стека управления может быть задана при создании процесса ВМ.

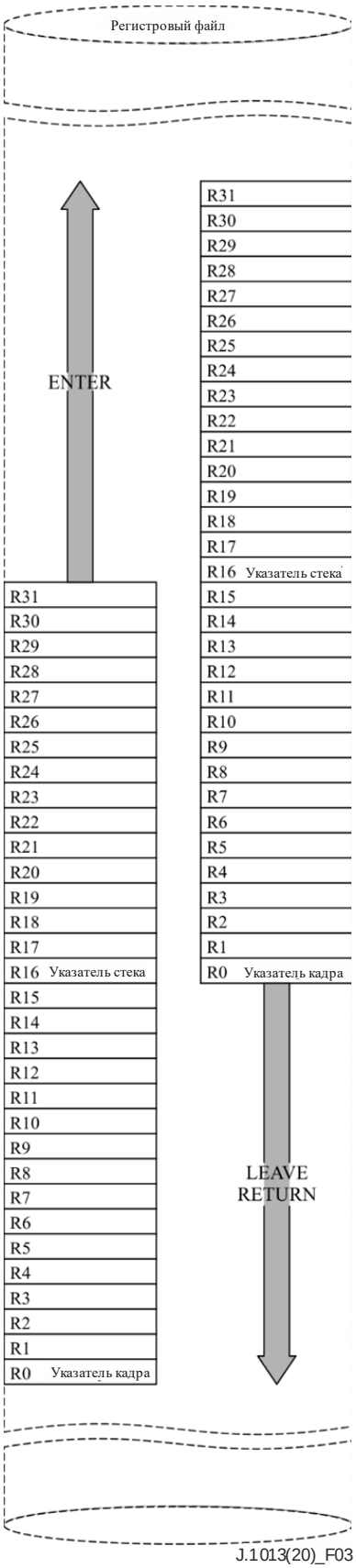


Рисунок 3 – Архитектура регистрового файла

7.2.3 Пространство данных

Базовый адрес данных ВМ, определенный как DATA_BASE_ADDRESS (см. Приложение А), должен равняться 0x1000000 (16 МБ). Наименьший адрес, расположенный выше адресуемой области памяти, равняется DATA_BASE_ADDRESS + ADDRESSABLE_DATA_SIZE (см. Приложение А). Базовый адрес стека определяется реализацией виртуальной машины, но должен находиться в верхней части адресного пространства. Допускается резервирование виртуальной машиной для собственных целей участка памяти размером не более VM_RESEVED_SIZE (см. Приложение А) в адресном пространстве **клиента ECI** "ниже" дна стека (по большему адресу). При инициализации ВМ указатель стека должен указывать на первую свободную ячейку стека. **Клиент ECI** может допустить, что верхняя граница (пустой) динамической памяти при инициализации равна сумме размеров сегментов инициализированных и неинициализированных данных, округленной до значения, кратного 4.

Распределение памяти данных схематически изображено на рисунке 4.

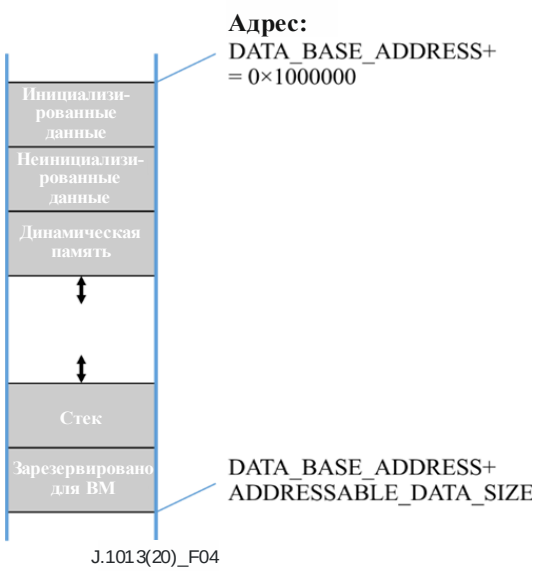


Рисунок 4 – Распределение памяти данных виртуальной машины

При инициализации **клиента ECI** загрузчик **клиента ECI** загружает сегменты инициализированных и неинициализированных данных, начиная с адреса DATA_BASE_ADDRESS. Всем байтам сегмента неинициализированных данных присваивается нулевое значение. Сегмент инициализированных данных не защищается от записи.

ПРИМЕЧАНИЕ 1. – Размер стека изначально ограничен. Поскольку локальные структуры данных, определяемые в функциях на языке С, обычно размещаются в стеке, **клиент ECI** должен установить надлежащий размер сегмента стека на случай, если в С-коде будут использоваться локальные переменные большого размера.

ПРИМЕЧАНИЕ 2. – **Хост ECI** может отображать буферы сообщений в зарезервированную для ВМ память ниже базового адреса стека.

В будущих версиях ВМ для **клиентов ECI** может резервироваться больший объем адресуемой памяти, то есть значение ADDRESSABLE_DATA_SIZE у них может быть больше. В целях обратной совместимости **клиенты ECI** не должны зависеть от конкретного значения или диапазона значений указателя стека, определенных в настоящий момент, а просто использовать то значение указателя стека, которое было передано при инициализации.

Загрузчик **клиента ECI** не должен загружать файлы образов, которые не соответствуют приведенной выше схеме распределения памяти для сегментов инициализированных и неинициализированных сегментов данных.

7.2.4 Кодовое пространство

Прямой доступ к коду из программы невозможен. Программа может получать 32-битовые непрозрачные ссылки на статические объекты кода (например, точка входа в подпрограмму, адрес

перехода), называемые *кодовыми ссылками* (code references) (см. команду MOVF). *Кодовые ссылки* могут использоваться только с командами косвенного управления ходом выполнения (JMPR и CALLR). Эти ссылки не являются указателями на кодовое пространство памяти, и с ними не должны выполняться операции адресной арифметики.

Начальный адрес сегмента кода в адресном пространстве программ должен равняться 0x00000000. Максимальный размер сегмента кода определен как CODE_SIZE (см. Приложение A).

7.2.5 Стек

По соглашению стек располагается в памяти данных, оперирует только словами и увеличивается в сторону уменьшения адресов, а на его верхушку (последнее помещенное в стек слово) всегда указывает регистр R16 (*указатель стека*) текущего регистрового окна.

Указатель кадра R0 используется в качестве указателя на стек вызываемой подпрограммы, с тем чтобы у нее имелась возможность доступа к помещенным в стек параметрам или другим данным (см. пункт 7.2.8).

7.2.6 Порядок следования байтов

Многобайтовые данные (полуслова и слова) представляются в системной памяти в формате с порядком байтов от младшего к старшему. Программное обеспечение **клиента ECI** должно использовать этот же порядок байтов.

7.2.7 Исключения

Центральный процессор ВМ не генерирует исключений во время выполнения. В случае выполнения команды в условиях, отличных от тех, которые изложены в настоящей Рекомендации (например, доступ без выравнивания к полуслову или слову в памяти, доступ к адресу, по которому отсутствует память, ветвление по неизвестной кодовой ссылке), поведение является неопределенным. По выбору виртуальной машины работа ядра может быть прервана. ВМ должна обеспечить, чтобы **клиент ECI**, работающий вне сферы применения настоящей Рекомендации, ни при каких обстоятельствах не мог получить несанкционированный доступ к данным или повлиять на работу другого приложения.

7.2.8 Соглашение о вызовах

Первые семь скалярных параметров (указатели и целочисленные значения) передаются через регистры R17–R23. Для вызываемой подпрограммы это будут регистры R1–R7 соответственно.

Скалярные параметры, начиная с восьмого, передаются через стек в порядке справа налево. Ввиду принципа работы механизма регистрового окна на восьмой параметр (если он есть) в вызываемой подпрограмме всегда будет указывать регистр R0. Поэтому R0 служит *указателем кадра*. Параметры-структуры всегда передаются через стек или по ссылке. Указатели всегда ссылаются на область памяти ВМ.

ПРИМЕЧАНИЕ. – Все команды SYSCALL передают структуры и массивы только по ссылке. Этот подход следует применять и для других вызовов.

Вызываемая подпрограмма помещает возвращаемое значение в регистр R1, то есть R17 с точки зрения вызывающей стороны. Типы данных размером менее 32 битов передаются и возвращаются как 32-битовые значения.

Возврат структур реализуется путем передачи неявного первого параметра, представляющего собой указатель на область памяти, куда предполагается записать возвращаемую структуру (передача по ссылке). Вызываемая подпрограмма записывает результат в то место, куда указывает данный параметр. Этот указатель на возвращаемое значение обрабатывается как обычный аргумент (с передачей через регистр R17 → R1), а обычные аргументы функции, возвращающей структуру, сдвигаются в другие регистры, предусмотренные соглашением о вызовах (R18...R23 → R2...R7), или передаются через стек.

7.3 Система команд виртуальной машины

7.3.1 Обозначения

Применяются следующие обозначения.

```
rx       Регистр x.
uimm5    5-битовый беззнаковый непосредственный операнд.
uimms9   9-битовый беззнаковый непосредственный операнд. Всегда кратен двум.
uimms10  10-битовый беззнаковый непосредственный операнд. Всегда кратен четырем.
simm11   11-битовый знаковый непосредственный операнд.
simm16   16-битовый знаковый непосредственный операнд.
uimm16   16-битовый беззнаковый непосредственный операнд.
pcr16    16-битовый знаковый операнд, заданный относительно регистра PC.
pcr24    24-битовый знаковый операнд, заданный относительно регистра PC.
imm32    32-битовый непосредственный операнд.
low8(x)  8 младших значащих битов x.
```

В описаниях функций для 32-битовых целочисленных типов данных применяется семантика языка C. О поддержке знаковых и беззнаковых типов данных сообщается в комментариях. Доступ в память обозначается функциями MEM1(), MEM2() и MEM4(), что соответствует адресации 1, 2 или 4 байтов памяти. Операндом этих функций служит смещение в сегменте данных. Там, где это уместно, к мнемонике MEM добавляется префикс U для беззнаковых операций или S для операций со знаковым расширением.

7.3.2 Арифметические команды

7.3.2.1 Команды с регистровыми операндами

```
ADD     r1,r2,rd        ; rd = r1 + r2;
SUB     r1,r2,rd        ; rd = r1 - r2;
OR      r1,r2,rd        ; rd = r1 | r2;
AND     r1,r2,rd        ; rd = r1 & r2;
XOR     r1,r2,rd        ; rd = r1 ^ r2;
SRA     r1,r2,rd        ; rd = r1 >> r2;     signed shift right
SRL     r1,r2,rd        ; rd = r1 >> r2;     logic shift right
SLL     r1,r2,rd        ; rd = r1 << r2;
MUL     r1,r2,rd        ; rd = r1 * r2;
SDIV    r1,r2,rd        ; rd = r1 / r2;      signed divide
SMOD    r1,r2,rd        ; rd = r1 % r2;      signed remainder
UDIV    r1,r2,rd        ; rd = r1 / r2;      unsigned divide
UMOD    r1,r2,rd        ; rd = r1 % r2;      unsigned remainder
EQ      r1,r2,rd        ; rd = r1 == r2;
NE      r1,r2,rd        ; rd = r1 != r2;
LT      r1,r2,rd        ; rd = r1 < r2;      signed less than
GE      r1,r2,rd        ; rd = r1 >= r2;     signed greater or equal
LTU     r1,r2,rd        ; rd = r1 < r2;      unsigned less than
GEU     r1,r2,rd        ; rd = r1 >= r2;     unsigned greater or equal
NOT     r1,rd           ; rd = ~r1;
NEG     r1,rd           ; rd = -r1;
ABS     r1,rd           ; rd = abs(r1);
MOV     r1,rd           ; rd = r1;
EXTB    r1,rd           ; rd = (int8_t) r1;   sign-extend from 8 bits
EXTH    r1,rd           ; rd = (int16_t) r1;  sign-extend from 16 bits
ZEXTB   r1,rd           ; rd = (uint8_t) r1;  zero-extend from 8 bits
ZEXTH   r1,rd           ; rd = (uint16_t) r1; zero-extend from 16 bits
MASKHI  r1,rd           ; rd = ~(-1) >> r1;   logic shift right
```

7.3.2.2 Команды с регистровыми и непосредственными операндами

```
ADDI    r1,imm32,rd       ; rd = r1 + imm32;
RSUBI   r1,imm32,rd       ; rd = imm32 - r1;
ORI     r1,imm32,rd       ; rd = r1 | imm32;
NORI    r1,imm32,rd       ; rd = ~(r1 | imm32);
ANDI    r1,imm32,rd       ; rd = r1 & imm32;
NANDI   r1,imm32,rd       ; rd = ~(r1 & imm32);
XORI    r1,imm32,rd       ; rd = r1 ^ imm32;
XNORI   r1,imm32,rd       ; rd = ~(r1 ^ imm32);
SRAI    r1,uimm5,rd       ; rd = r1 >> uimm5;     signed
SRLI    r1,uimm5,rd       ; rd = r1 >> uimm5;     logic
SLLI    r1,uimm5,rd       ; rd = r1 << uimm5;
MULI    r1,imm32,rd       ; rd = r1 * imm32;
MACI    r1,imm32,rd       ; rd += r1 * imm32;
SMODI   r1,imm32,rd       ; rd = r1 % imm32;      signed
SDIVI   r1,imm32,rd       ; rd = r1 / imm32;      signed
UMODI   r1,imm32,rd       ; rd = r1 % imm32;      unsigned
UDIVI   r1,imm32,rd       ; rd = r1 / imm32;      unsigned
EQI     r1,imm32,rd       ; rd = r1 == imm32;
NEI     r1,imm32,rd       ; rd = r1 != imm32;
LTI     r1,imm32,rd       ; rd = r1 < imm32;      signed
GTI     r1,imm32,rd       ; rd = r1 > imm32;      signed
```

```
GEI     r1,imm32,rd       ; rd = r1 >= imm32;     signed
LEI     r1,imm32,rd       ; rd = r1 <= imm32;     signed
LTUI    r1,imm32,rd       ; rd = r1 < imm32;      unsigned
GTUI    r1,imm32,rd       ; rd = r1 > imm32;      unsigned
GEUI    r1,imm32,rd       ; rd = r1 >= imm32;     unsigned
LEUI    r1,imm32,rd       ; rd = r1 <= imm32;     unsigned
ADDMXI  r1,imm32,rd       ; rd = (r1 + imm32) % 0x7fffffff;
MOVC    simm16,rd         ; rd = simm16;
MOVI    imm32,rd          ; rd = imm32;
MOVF    caddr,rd          ; rd = caddr;           load code reference
CLR     rd                ; rd = 0;
INC     rd                ; rd = rd + 1;
DEC     rd                ; rd = rd - 1;
```

Знаковые операции деления и взятия остатка выполняются в соответствии с определением в стандарте C99: при делении дробная часть математического результата отбрасывается, а остаток удовлетворяет соотношению

$$\frac{a}{b} \times b + a\%b = a,$$

где % представляет операцию взятия остатка (деления по модулю).

Правый операнд в командах сдвига должен находиться в диапазоне [0, 31], иначе поведение будет неопределенным. При знаковом сдвиге вправо старший значащий бит исходного значения копируется в освободившиеся позиции. Арифметически это соответствует делению на степень числа два с округлением математического результата в направлении минус бесконечность (операция FLOOR).

7.3.3 Сокращенные формы

Во многих случаях в командах с тремя операндами один из операндов одновременно служит результатом. Поскольку такие команды можно закодировать компактнее, для них предусмотрены специальные коды операций:

```
ADD2    r1,rd                       ; rd += r1;
SUB2    r1,rd                       ; rd -= r1;
MUL2    r1,rd                       ; rd *= r1;
AND2    r1,rd                       ; rd &= r1;
OR2     r1,rd                       ; rd |= r1;
XOR2    r1,rd                       ; rd ^= r1;
XNOR2   r1,rd                       ; rd = ~(rd ^ r1);
NE2     r1,rd                       ; rd = r1 != rd;
EQ2     r1,rd                       ; rd = r1 == rd;
SLL2    r1,rd                       ; rd <<= r1;
SRA2    r1,rd                       ; rd >>= r1;          signed
SRL2    r1,rd                       ; rd >>= r1;          logical
```

При битовых операциях с непосредственными операндами проверяется или изменяется отдельный бит. Эти непосредственные операнды могут кодироваться 5-битовым значением, задающим положение соответствующего бита.

```
ANDB   r1,uimm5,rd                  ; rd = r1 & (1 << uimm5);
ORB    r1,uimm5,rd                  ; rd = r1 | (1 << uimm5);
XORB   r1,uimm5,rd                  ; rd = r1 ^ (1 << uimm5);
TESTB  r1,uimm5,rd                  ; rd = (r1 >> uimm5) & 1;
TESTBC r1,uimm5,rd                  ; rd = ! ((r1 >> uimm5) & 1);
```

Часто приходится делать сравнение с нулем. В этом случае можно сэкономить на непосредственном операнде, что потребует меньше ресурсов при эмуляции.

```
EQZ     r1,rd                       ; rd = r1 == 0;
NEZ     r1,rd                       ; rd = r1 != 0;
LTZ     r1,rd                       ; rd = r1 < 0;
GTZ     r1,rd                       ; rd = r1 > 0;
LEZ     r1,rd                       ; rd = r1 <= 0;
GEZ     r1,rd                       ; rd = r1 >= 0;
```

Беззнаковые версии этих команд не имеют смысла. Они тождественно истинны или ложны либо могут быть выражены с помощью `EQZ` или `NEZ`.

7.3.4 Управление ходом выполнения

7.3.4.1 Общие правила

В командах управления ходом выполнения с прямыми операндами адресация операндов осуществляется относительно конечного адреса самой команды. В случае регистровых команд регистр содержит указатель на функцию.

7.3.4.2 Безусловное ветвление и вызовы функций

```
JMP     pcr24               ; goto PC+pcr24;
JMPR    rd                  ; goto rd (shall be code reference);
CALL    pcr24               ; push PC; goto PC+pcr24;
CALLR   rd                  ; push program counter; goto rd (code reference);
ENTER   uimm16              ; shift register file by 16 (new r0 is old r16);
                            ; r16 = r0 - 4 * uimm16;
ENTER0                      ; equivalent to ENTER 0
ENTERC  uimms10             ; equivalent to ENTER uimms10
LEAVE                       ; unshift register file
RETURN                      ; unshift register file;
                            ; goto popped program counter;
RETURNL                     ; goto popped program counter;

SWITCH  r1,uimm16           ; goto PC + MIN(r1, uimm16)
                          ; advance to r1th CASE statement below
CASE    pcr24               ; goto PC + pcr24
                          ; add a case in the previous SWITCH. The first
                            ; entry is case value zero, each next one adds
                            ; one to the case value.
```

7.3.4.3 Условное ветвление

```
JEQ     r1,r2,pcr16             ; if (r1 == r2) goto PC+pcr16;
JNE     r1,r2,pcr16             ; if (r1 != r2) goto PC+pcr16;
JLT     r1,r2,pcr16             ; if (r1 < r2) goto PC+pcr16;
JGE     r1,r2,pcr16             ; if (r1 >= r2) goto PC+pcr16;
JLTU    r1,r2,pcr16             ; if ((unsigned)r1 < (unsigned)r2) goto PC+pcr16;
JGEU    r1,r2,pcr16             ; if ((unsigned)r1 >= (unsigned)r2) goto PC+pcr16;

JEQC    r1,simm11,pcr16         ; if (r1 == simm11) goto PC+pcr16;
JNEC    r1,simm11,pcr16         ; if (r1 != simm11) goto PC+pcr16;
JLTC    r1,simm11,pcr16         ; if (r1 < simm11) goto PC+pcr16;
JGEC    r1,simm11,pcr16         ; if (r1 >= simm11) goto PC+pcr16;
JLTUC   r1,uimm11,pcr16         ; if ((unsigned) r1 < uimm11) goto PC+pcr16;
JGEUC   r1,uimm11,pcr16         ; if ((unsigned) r1 >= uimm11) goto PC+pcr16;
JGTC    r1,simm11,pcr16         ; if (r1 > simm11) goto PC+pcr16;
JLEC    r1,simm11,pcr16         ; if (r1 <= simm11) goto PC+pcr16;
JGTUC   r1,uimm11,pcr16         ; if ((unsigned) r1 > uimm11) goto PC+pcr16;
JLEUC   r1,uimm11,pcr16         ; if ((unsigned) r1 <= uimm11) goto PC+pcr16;
```

7.3.4.4 Условное ветвление по результатам сравнения значения из памяти с константой

```
JWEQC   r1,simm11,pcr16     ; if (MEM4(r1) == simm11) goto PC+pcr16;
JWNEC   r1,simm11,pcr16     ; if (MEM4(r1) != simm11) goto PC+pcr16;
```

Эти команды считывают значение из памяти и сравнивают его с константой.

7.3.4.5 Дальнее условное ветвление

Для каждой из описанных выше команд условного ветвления существует *дальняя* версия с 24-битовым смещением. Ассемблер должен выбирать кратчайшую подходящую версию.

7.3.5 Команды загрузки и хранения

7.3.5.1 Значение регистра + смещение

```
LDSBI    r1,imm32,rd                    ; rd = SMEM1(r1 + imm32);
LDUBI    r1,imm32,rd                    ; rd = UMEM1(r1 + imm32);
LDSHI    r1,imm32,rd                    ; rd = SMEM2(r1 + imm32);
LDUHI    r1,imm32,rd                    ; rd = UMEM2(r1 + imm32);
LDWI     r1,imm32,rd                    ; rd = MEM4 (r1 + imm32);
```

```
STBI     rd,r1,imm32                    ; MEM1(r1 + imm32) = low8(rd);
STHI     rd,r1,imm32                    ; MEM2(r1 + imm32) = low16(rd);
STWI     rd,r1,imm32                    ; MEM4(r1 + imm32) = rd;
```

7.3.5.2 Значение регистра + короткое смещение

```
LDSBC    r1,uimm8,rd                    ; rd = SMEM1(r1 + uimm8);
LDUBC    r1,uimm8,rd                    ; rd = UMEM1(r1 + uimm8);
LDSHC    r1,uimms9,rd                   ; rd = SMEM2(r1 + uimms9);
LDUHC    r1,uimms9,rd                   ; rd = UMEM2(r1 + uimms9);
LDWC     r1,uimms10,rd                  ; rd = MEM4 (r1 + uimms10);

STBC     rd,r1,uimm8                    ; MEM1(r1 + uimm8) = low8(rd);
STHC     rd,r1,uimms9                   ; MEM2(r1 + uimms9) = low16(rd);
STWC     rd,r1,uimms10                  ; MEM4(r1 + uimms10) = rd;
```

7.3.5.3 Регистровая индексация

```
LDUB     r1,r2,rd                       ; rd = UMEM1(r1 + r2);
LDSB     r1,r2,rd                       ; rd = SMEM1(r1 + r2);
LDUH     r1,r2,rd                       ; rd = UMEM2(r1 + 2 * r2);
LDSH     r1,r2,rd                       ; rd = SMEM2(r1 + 2 * r2);
LDW      r1,r2,rd                       ; rd = MEM4(r1 + 4 * r2);

STB      rd,r1,r2                       ; MEM1(r1 + r2) = rd;
STH      rd,r1,r2                       ; MEM2(r1 + 2 * r2) = rd;
STW      rd,r1,r2                       ; MEM4(r1 + 4 * r2) = rd;

LDW1     r1,r2,rd                           ; rd = MEM4(r1 + r2);
STW1     rd,r1,r2                           ; MEM4(r1 + r2) = rd;
```

7.3.5.4 Абсолютная индексация

```
LDSHAX   imm32,r1,rd                        ; rd = SMEM2(imm32 + 2 * r1);
LDUHAX   imm32,r1,rd                        ; rd = UMEM2(imm32 + 2 * r1);
LDWAX    imm32,r1,rd                        ; rd = MEM4(imm32 + 4 * r1);
STHAX    rd,imm32,r1                        ; MEM2(imm32 + 2 * r1)= rd;
STWAX    rd,imm32,r1                        ; MEM4(imm32 + 4 * r1)= rd;
```

Следует отметить, что в загрузке байтов с абсолютной индексацией нет необходимости. Например, `LDSBAX` эквивалентно `LDSBI`.

7.3.5.5 Специальные команды для доступа к стеку

Эти команды производят загрузку и хранение слов с неявным использованием указателя кадров.

```
LDFP     simm16,r1                          ; r1 = MEM4(FP + simm16);
STFP     r1,simm16                      ; MEM4(FP + simm16) = r1;
```

7.3.5.6 Передача данных памяти

Команда блочного копирования, используемая для получения таких копий, генерируемых компилятором.

```
COPY     r1,s:uimm32,r2,o:uimm32        ; copy s bytes from r1 to r2+o
```

7.3.6 Сложные команды

Это команды, выполняющие комбинированные операции, как правило, с непосредственными операндами. В этой сводке команд каждый операнд, обозначенный как `i1`, `i2` и т. д., представляет собой 32-битовый непосредственный операнд (imm32).

```
ADDANDI2    r1,i1,i2,rd                 ; rd = (r1 + i1) & i2;
ADDMULI2    r1,i1,i2,rd                 ; rd = (r1 + i1) * i2;
ADDORI2 r1,i1,i2,rd                     ; rd = (r1 + i1) | i2;
ADDXORI2    r1,i1,i2,rd                 ; rd = (r1 + i1) ^ i2;

MULADDI2    r1,i1,i2,rd                 ; rd = (r1 * i1) + i2;
MULANDI2    r1,i1,i2,rd                 ; rd = (r1 * i1) & i2;
MULORI2 r1,i1,i2,rd                     ; rd = (r1 * i1) | i2;
```

```
MULXORI2    r1,i1,i2rd                    ; rd = (r1 * i1) ^ i2;

RSUBANDI2   r1,i1,i2,rd                   ; rd = (i1 - r1) & i2;
RSUBORI2    r1,i1,i2,rd                   ; rd = (i1 - r1) | i2;
RSUBXORI2   r1,i1,i2,rd                   ; rd = (i1 - r1) ^ i2;

ORADDI2 r1,i1,i2,rd                       ; rd = (r1 | i1) + i2;
ORMULI2 r1,i1,i2,rd                       ; rd = (r1 | i1) * i2;

SLLADDI2    r1,s1:uimm5,i2,rd             ; rd = (r1 << s1) + i2;
SLLANDI2    r1,s1:uimm5,i2,rd             ; rd = (r1 << s1) & i2;
SLLORI2 r1,s1:uimm5,i2,rd                 ; rd = (r1 << s1) | i2;
SLLRSUBI2   r1,s1:uimm5,i2,rd             ; rd = i2 - (r1 << s1);

ANDSLLI2    r1,i1,s2:uimm5,rd             ; rd = (r1 & i1) << s2;

MAMI3       r1,i1,i2,i3,rd                ; rd = ((r1 * i1) & i2) * i3;
MPMI3       r1,i1,i2,i3,rd                ; rd = ((r1 * i1) + i2) * i3;
MOMI3       r1,i1,i2,i3,rd                ; rd = ((r1 * i1) | i2) * i3;

MPAI3       r1,i1,i2,i3,rd                ; rd = ((r1 * i1) + i2) & i3;
MPOI3       r1,i1,i2,i3,rd                ; rd = ((r1 * i1) + i2) | i3;
RORI3       r1,i1,i2,i3,rd                ; rd = i3 - ((i1 - r1) | i2);
AMPI3       r1,i1,i2,i3,rd                ; rd = ((r1 & i1) * i2) + i3;

LPAI3       r1,s1:uimm5,i2,i3,rd          ; rd = ((r1 << s1) + i2) & i3;

MPMPI4  r1,i1,i2,i3,i4,rd                 ; rd = (((r1 * i1) + i2) * i3) + i4;
MPOMI4  r1,i1,i2,i3,i4,rd                 ; rd = (((r1 * i1) + i2) | i3) * i4;
```

7.3.7 Разное

7.3.7.1 Системные вызовы

С помощью системных вызовов реализованы различные услуги.

```
SYSCALL  uimm16                           ; system service uimm16
```

Реализован минимальный набор системных вызовов стандарта POSIX (интерфейс переносимых операционных систем), непосредственно транслированных в вызовы базовой ОС. Возможно добавление других услуг, ориентированных на конкретные приложения.

7.3.7.2 Псевдокоманды

Некоторые операции могут быть выражены через другие. Предусмотрены следующие коды псевдокоманд.

```
SUBI    r1,imm32,rd    = ADDI  r1,-imm32,rd
GT      r1,r2,rd       = LT    r2,r1,rd
LE      r1,r2,rd       = GE    r2,r1,rd
GTU     r1,r2,rd       = LTU   r2,r1,rd
LEU     r1,r2,rd       = GEU   r2,r1,rd
```

8 Интерфейс между клиентом ECI и хостом ECI

8.1 Общие принципы

Системные вызовы совершаются с помощью команды SYSCALL. У этой команды есть непосредственный операнд, идентифицирующий системный вызов. Системные вызовы – это, по сути, вызовы к стандартной библиотеке с передачей параметров методом, описанным в пункте 7.2.8.

Первые 7 параметров (слова или указатели) передаются через регистры R1–R8. Если по фактическому типу эти параметры являются 8- или 16-битовыми скалярными параметрами, они расширяются со знаком до 32-битового значения. Возвращаемые значения (слова или указатели) помещаются в регистр R1.

Если не указано иное, все адреса памяти относятся к адресному пространству памяти ВМ.

В целях будущей совместимости **клиент ECI** должен устанавливать на нуль все регистры R1–R8, которые не используются для передачи параметров. Содержимое всех регистров может быть очищено от "мусора" библиотечной функцией.

Библиотечные системные вызовы, которые должны в обязательном порядке поддерживаться всеми реализациями для соответствия требованиям настоящей Рекомендации, перечислены ниже. Приводятся следующие данные:

– идентификатор системного вызова, указываемый в качестве непосредственного операнда (SYSCALL imm32);

– описание библиотечной функции;

– объявление в синтаксисе C;

– описание параметров и возвращаемого значения;

– любые дополнительные примечания.

Типы параметров и возвращаемых значений обозначаются следующим образом:

– **uint*nn*** – *nn*-битовое целое без знака (*nn* = 8, 16 или 32). Значения менее 32 битов при помещении в регистры дополняются нулями до 32-битовых;

– **int*nn*** – целое со знаком. Значения менее 32 битов при помещении в регистры расширяются со знаком до 32-битовых;

– **void *** – указатель общего вида;

– **[u]int*nn* *** – указатель на одиночное значение типа [u]int*nn* или массив таких значений;

– **struct struct_type *** – указатель на структуру (или массив структур) в памяти; структуры всегда передаются по ссылке таким способом.

8.2 Код ошибки

Большинство системных вызовов возвращают слово с отрицательным значением, указывающее на обнаружение состояния ошибки. В таблице 1 перечислены коды ошибок.

Таблица 1 – Коды ошибок

Значение	Символическое наименование	Смысл
–49	EPERM	Несуществующий системный вызов или библиотечная функция
–50	EINVAL	Неверное значение одного из параметров
–51	ERRSYSCALLMSGQUEUE	Количество сообщений, переданных **на хост ECI**, превышает объем его буфера
–52	ERRHEAPSIZE	Запрошено ненадлежащее значение размера динамической памяти
–53	ERRSTACKSIZE	Запрошено ненадлежащее значение размера стека

8.3 SYS_EXIT

Идентификатор системного вызова: 0x0001

Описание: завершает работу ВМ с возвращением кода причины.

Объявление: void SYS_EXIT(uint32 reason)

Операнды: **reason** – код причины завершения работы.

Возвращаемое значение: отсутствует.

ПРИМЕЧАНИЯ. – Аргумент **reason** принимает одно из значений, перечисленных в таблице 2.

Таблица 2 – Коды причин завершения работы, возвращаемые SYS_EXIT

reason	Смысл
0	Нормальное завершение работы
0x00000001..0x7FFFFFFF	Состояние ошибки, специфичное для поставщика клиента ECI
0x80000000..0xFFFFFFFF	Зарезервировано для будущего использования

8.4 SYS_PUTMSG

Идентификатор системного вызова: 0x0003

Описание: передает асинхронное сообщение (запрос или ответ).

Объявление: int32 SYS_PUTMSG(MessageBuffer *msg_buffer)

Формат структуры буфера сообщений MessageBuffer определен в [ITU-T J.1012].

Операнды: **msg_buffer** – указатель на блок буфера сообщений.

Возвращаемое значение: идентификатор сообщения, присвоенный **хостом ECI** (16-битовое неотрицательное значение), или один из приведенных ниже кодов ошибки (отрицательных): **ERRSYSCALLMSGQUEUE** (таблица 1).

ПРИМЕЧАНИЯ. – Предполагается, что вызов не будет блокироваться в нормальных условиях работы **хоста ECI**. Содержимое msg_buffer копируется **хостом ECI** и может далее использоваться **клиентом ECI** сразу после возврата из системного вызова.

8.5 SYS_GETMSG

Идентификатор системного вызова: 0x0004

Описание: получает следующее сообщение (запрос или результат) от **хоста ECI**. При отсутствии сообщения системный вызов блокируется.

Объявление: (MessageBuffer *) SYS_GETMSG()

Формат структуры буфера сообщений MessageBuffer определен в [ITU-T J.1012].

Операнды: отсутствуют.

Возвращаемое значение: указатель на буфер, содержащий текстовое сообщение от **хоста ECI** или один из кодов ошибки, перечисленных в таблице 1.

ПРИМЕЧАНИЯ. – Вызов блокируется, если у **хоста ECI** нет сообщений в очереди для **клиента ECI**. **Хост ECI** не изменяет содержимое буфера сообщений до следующего вызова SYS_GETMSG. **Клиенты ECI**, которым требуется доступ к данным сообщений после следующего вызова SYS_GETMSG, должны скопировать эти данные.

8.6 SYS_HEAPSIZE

Идентификатор системного вызова: 0x0100

Описание: запрос **хосту ECI** на установку нового размера динамической памяти, равного значению переданного аргумента.

Объявление: int32 SYS_HEAPSIZE(uint32 heapsize)

Операнды: **heapsize** – новый размер динамической памяти **клиента ECI**. Это значение должно быть неотрицательным и кратным четырем, а также не должно вызывать переполнения сегмента динамической памяти с выходом в сегмент стека.

Возвращаемое значение: смещение ячейки памяти относительно DATA_BASE_ADDRESS в байтах, представляющее собой наименьший не относящийся к динамической памяти адрес в адресуемой памяти, или один из перечисленных ниже кодов ошибки **ERRHEAPSIZE** (таблица 1).

ПРИМЕЧАНИЯ. – Вызов блокируется, если у **хоста ECI** нет сообщений в очереди для **клиента ECI**. При инициализации **клиента ECI** запрос SYS_HEAPSIZE(0) возвращает смещение начала сегмента динамической памяти (которая на этот момент имеет нулевой размер).

8.7 SYS_STACKSIZE

Идентификатор системного вызова: 0x0200

Описание: запрос **хосту ECI** на установку нового размера стека, равного значению переданного аргумента.

Объявление: int32 SYS_STACKSIZE(uint32 stacksize)

Операнды: **stacksize** – новый размер стека **клиента ECI**. Это значение должно быть неотрицательным и кратным четырем, а также не должно вызывать переполнения сегмента стека с выходом в сегмент динамической памяти.

Возвращаемое значение: смещение ячейки памяти относительно DATA_BASE_ADDRESS в байтах, представляющее собой наименьший не относящийся к стеку адрес в адресуемой памяти, или один из перечисленных ниже кодов ошибки **ERRSTACKSIZE** (таблица 1).

ПРИМЕЧАНИЯ. – Вызов блокируется, если у **хоста ECI** нет сообщений в очереди для **клиента ECI**.

8.8 SYS_SYNCCALL

Идентификатор системного вызова: 0x1000

Описание: **клиент ECI** передает **хосту ECI** синхронное сообщение, и его выполнение приостанавливается до возврата из системного вызова.

Объявление: int32 SYS_SYNCCALL(uint32 tag, p1, p2, p3, …, pn)

Операнды: **tag** – то же, что и поле MsgTag структуры MessageBuffer. Поле MsgFlags должно быть установлено на нуль и должно игнорироваться **хостом ECI**. **p1–pn** – параметры синхронного вызова. Для get-сообщений с разрядностью результата более 32 битов **p1** содержит начальный адрес ячейки памяти, где будет храниться возвращаемый результат, а **p2**–**pn** – параметры set-сообщения. Все обычные параметры, в том числе структуры и массивы, передаются по ссылке.

Возвращаемое значение: в случае get-сообщений с разрядностью результата до 32 битов возвращается непосредственно результат. В противном случае возвращаемый результат отсутствует. Все ошибки игнорируются; при ошибочных конфигурациях параметров результат не возвращается и/или не выполняется никаких действий. Сообщения вызова возвращают код состояния в соответствии с их семантикой. Результаты могут возвращаться в ячейках памяти, где хранятся переданные по ссылке параметры системного вызова, в соответствии с семантикой конкретного сообщения.

ПРИМЕЧАНИЯ. – Этот системный вызов не блокируется.

8.9 SYS_CLIB

Идентификатор системного вызова: 0x0300

Описание: Этот системный вызов работает как интерфейс прикладного программирования (API), позволяющий **клиенту ECI** пользоваться функциями стандартной библиотеки языка C. Поддерживаемый набор функций подробно описывается в Приложении C.

Объявление: SYS_CLIB(uint32 clibfunc, ...)

Операнды: **clibfunc** – идентификатор вызываемой библиотечной функции языка C (см. Приложение C). Определения всех остальных операндов даются в списке библиотечных функций языка C.

Возвращаемое значение: возвращаемое значение согласно определению в списке библиотечных функций языка C (см. Приложение C) или один из кодов ошибки, перечисленных в таблице 1.

ПРИМЕЧАНИЯ. – Поскольку число и типы параметров у библиотечных функций языка C различаются, они не описаны в настоящей Рекомендации явным образом. Формат всех операндов подробно описывается в Приложении. Все операнды системного вызова SYS_CLIB представляют собой скалярные значения или указатели на нескалярные значения в памяти ВМ. Ввиду того что некоторые библиотечные функции языка C

могут принимать нескалярные параметры, ВМ должна преобразовывать параметры, передаваемые по ссылке, в параметры, передаваемые по значению, перед входом в библиотечную подпрограмму.

9 Жизненный цикл байт-кода

9.1 Введение

Виртуальная машина реализуется в составе прошивки **хоста ECI**. Она динамически загружается/ выполняется операционной системой **хоста ECI**, когда требуется запустить выполнение **клиента ECI**. Может предоставляться множество экземпляров ВМ для одновременной работы различных **клиентов ECI**.

Клиент ECI написан в кодах ВМ, система команд которой описана выше. Его пишет поставщик системы защиты контента (поставщик услуг условного доступа или оператор системы управления цифровыми правами) и предоставляет **хосту ECI** в виде логического образа кода. Далее он модифицируется локально под конкретный **хост ECI** и его рабочую среду и загружается в ВМ по требованию. **Клиент ECI** выполняется в виртуальной машине, пока его работа не будет завершена (штатно или пока не возникнет состояние ошибки); после этого выполнение **клиента ECI** останавливается, и ВМ завершает работу.

9.2 Загрузка нового клиента ECI в ВМ

ВМ работает в качестве промежуточного хоста для любого внешнего **клиента ECI** точно так, как если бы **клиент ECI** был приложением, написанным в собственных кодах **хоста ECI**. Единственное отличие состоит в том, что **клиент ECI** устанавливается операционной системой **хоста ECI** в ВМ, а не как собственное приложение.

Чтобы загрузить **клиента ECI**, подсистема виртуальной машины сначала создает контекст виртуального процессора. Для загрузки выделяется память в ВМ, и в нее устанавливается содержимое сегментов кода и данных, как если бы это были собственные приложения, но при этом код и инициализированные данные из файлов формата исполняемых и компонуемых файлов (ELF) [ETSI GS ECI 001-4] (более подробную информацию см. в Приложении D) передаются в память, выделенную ВМ.

Поскольку сегмент кода недоступен из программы, в реализации по выбору разработчика на этапе загрузки может осуществляться любая предварительная обработка кода, например оптимизация. В настоящей Рекомендации описывается только формат программы в образе, а внутреннее представление полностью определяется спецификой реализации. Единственное условие заключается в том, чтобы все кодовые ссылки оставались доступными для использования программой с сохранением семантики.

Как вариант, образ **клиента ECI** может быть подвергнут предварительной обработке при первой загрузке на **хост ECI** и затем сохранен в форме, готовой к загрузке по требованию. Это более эффективный способ хранения и запуска **клиентов ECI**, если они регулярно выгружаются и загружаются.

9.3 Инициализация виртуальной машины

При инициализации необходимо создать общий контекст ЦП виртуальной машины, то есть регистровый файл, стек управления, области данных и стека, счетчик команд (PC) и всю необходимую логику управления/отображения состояния и флаги. Они не описываются в настоящей Рекомендации, поскольку зависят от реализации.

Регистровый файл организуется таким образом, что регистр R0 находится в начале файлового пространства. Все регистры устанавливаются на нуль. Указатель кадра и указатель стека (регистры R0 и R16 соответственно) в первом регистровом окне устанавливаются таким образом, что при перемещении в стек первого слова указатель стека предварительно уменьшается до –4 (0xFFFFFFFC) и слово сохраняется там.

Начальное значение счетчика команд соответствует началу сегмента кода, если только записи *e_entry* в заголовке ELF [ETSI GS ECI 001-4] в файле образа **клиента ECI** не присвоено ненулевое значение, в последнем случае счетчик команд устанавливается в значение (виртуальный адрес), содержащееся в

записи *e_entry*. Затем управление передается исполнительному компоненту ВМ, который называется главным циклом исполнения.

9.4 Главный цикл исполнения

Суть ВМ заключается в главном цикле исполнения, который считывает и транслирует каждую из последовательно идущих команд в соответствующий набор операций с регистрами и памятью виртуальной машины и/или вызовов к библиотеке. Команды выполняются в цикле, пока не возникает то или иное исключение. Завершение работы программы может быть частью обычного хода выполнения (например, если программа совершает системный вызов `SYS_EXIT`) или же следствием ошибки (например, при переполнении стека управления).

Если главный цикл исполнения заканчивается, то работа всей виртуальной машины завершается, и если в дальнейшем потребуется обратиться к **клиенту ECI**, необходимо будет создать новый ее экземпляр.

Приложение A

Системные ресурсы виртуальной машины

(Данное Приложение является неотъемлемой частью настоящей Рекомендации)

В настоящей Рекомендации для определения рабочих характеристик ВМ используются перечисленные ниже параметры. Предлагаемые значения этих параметров приведены в [b-ITU-T J Suppl. 7].

– REGISTER_FILE_SIZE
– CONTROL_STACK_SIZE = REGISTER_FILE_SIZE/16
– DATA_BASE_ADDRESS
– ADDRESSABLE_DATA_SIZE
– VM_RESERVED_SIZE
– CODE_SIZE
– DEFAULT_STACK_SIZE
– MIN_RAM

Приложение B

Коды операций виртуальной машины

(Данное Приложение является неотъемлемой частью настоящей Рекомендации)

Ниже описывается кодирование команд в двоичном образе. Приводится обзор различных форматов. В сводной строке через запятую дается список полей, составляющих команду. Это либо явно заданные биты, либо имя поля, за которым следует указатель длины поля. Коды различных операций, представленные в одном формате, перечисляются вместе с соответствующим содержимым поля `op`. Биты следуют в порядке от старшего к младшему.

Например, команда `SUB R3,R5,R17` кодируется как
```
1011 00001 00011 00101 10001
```

или в полубайтах:
```
1011 0000 1000 1100 1011 0001
```

или в байтах:
```
0xB0, 0x8C, 0xB1
```

За именем каждой команды следует число, которое является определенным номером кода операции данной команды. Этот номер должен оставаться неизменным во всех будущих версиях системы команд ВМ.

Длина полей в кодах операций не должна превышать четырех байтов. Соответственно 32-битовые поля должны начинаться на границе байта. Длина поля не превышает 32 бита.

```
0, op:5, r1:5, rd:5
    00000       MOV 16
    00001       ADD2    17
    00010       SUB2    18
    00011       MUL2    19
    00100       AND2    20
    00101       OR2 21
    00110       XOR2    22
    00111       SLL2    23
    01000       SRL2    24
    01001       SRA2    25
    01010       NE2 26
    01011       EQ2 27
    01100       NEZ 28
    01101       EQZ 29
    01110       LTZ 30
    01111       GEZ 31
    10000       GTZ 32
    10001       LEZ 33
    10010       EXTB    34
    10011       EXTH    35
    10100       ZEXTB   36
    10101       ZEXTH   37
    10110       ABS 38
    10111       NEG 39
    11000       NOT 40
    11001       XNOR2   41
    11010       MASKHI  42

100, op:3, r1:5, rd:5, imm:32
    000         ADDI    136
    001         RSUBI   137
    010         ANDI    138
    011         ORI 139
    100         XORI    140
    101         MULI    141
    110         MACI    142
    111         ADDMXI  143

101000, op:2
    00          ENTER0  0
    01          RETURN  1
    10          RETURNL 2
```

```
    11          LEAVE   3

10100100, op:3, rd:5
    000         INC 8
    001         DEC 9
    010         JMPR    10
    011         CALLR   11
    100         CLR 12

1010010100000, op:4, r1:5, r2:5, rd:5
    0000        SDIV    80
    0001        SMOD    81
    0010        UDIV    82
    0011        UMOD    83
    0100        TESTBC  84

101001010001, op:4, r1:5, imm:11, pcr:24
    0000        JFNEC   560
    0001        JFEQC   561
    0010        JFLTC   562
    0011        JFGEC   563
    0100        JFGTC   564
    0101        JFLEC   565
    0110        JFLTUC  566
    0111        JFGEUC  567
    1000        JFLEUC  568
    1001        JFGTUC  569
    1010        JFWNEC  570
    1011        JFWEQC  571

10101000, op:3, r1:5, imm:16
    000         STFP    96
    001         LDFP    97
    010         MOVC    98
    011         SWITCH  99

1011, op:5, r1:5, r2:5, rd:5
    00000       ADD 48
    00001       SUB 49
    00010       MUL 50
    00011       AND 51
    00100       OR  52
    00101       XOR 53
    00110       SLL 54
    00111       SRA 55
    01000       SRL 56
    01001       SLLI    57
    01010       SRAI    58
    01011       SRLI    59
    01100       NE      60
    01101       EQ      61
    01110       LT      62
    01111       GE      63
    10000       LTU 64
    10001       GEU 65
    10010       ANDB    66
    10011       ORB 67
    10100       XORB    68
    10101       LDSB    69
    10110       LDUB    70
    10111       LDSH    71
    11000       LDUH    72
    11001       LDW 73
    11010       LDW1    74
    11011       STB 75
    11100       STH 76
    11101       STW 77
    11110       STW1    78
    11111       TESTB   79

110000, op:2, imm:24
    00          JMP 104
    01          CALL    105
    10          CASE    106

110001000, op:2, rd:5, imm:32
    00          MOVI    108
    01          MOVF    109
```

```
110001001, op:5, r1:5, rd:5, imm:32
    00000       NANDI   144
    00001       NORI    145
    00010       XNORI   146
    00011       NEI 147
    00100       EQI 148
    00101       LTI 149
    00110       GEI 150
    00111       GTI 151
    01000       LEI 152
    01001       LTUI    153
    01010       GEUI    154
    01011       GTUI    155
    01100       LEUI    156
    01101       SMODI   157
    01110       SDIVI   158
    01111       UMODI   159
    10000       UDIVI   160
    10001       STBI    161
    10010       STHI    162
    10011       STWI    163
    10100       LDSBI   164
    10101       LDUBI   165
    10110       LDSHI   166
    10111       LDUHI   167
    11000       LDWI    168
    11001       LDSHAX  169
    11010       LDUHAX  170
    11011       LDWAX   171
    11100       STHAX   172
    11101       STWAX   173

11001000000, op:3, r1:5, rd:5, imm:24
    000         JFNE    576
    001         JFEQ    577
    010         JFLT    578
    011         JFGE    579
    100         JFLTU   580
    101         JFGEU   581

11001000110, op:3, r1:5, r2:5, imm:16
    000         JNE 120
    001         JEQ 121
    010         JLT 122
    011         JGE 123
    100         JLTU    124
    101         JGEU    125

11001000111000, r1:5, r2:5, s:32, o:32
                COPY    112

11001001000, op:3, r1:5, r2:5, imm:8
    000         STBC    128
    001         STHC    129
    010         STWC    130
    011         LDSBC   131
    100         LDUBC   132
    101         LDSHC   133
    110         LDUHC   134
    111         LDWC    135

11001100000000000, op:5, r1:5, rd:5, imm1:32, imm2:32
    00000       ADDANDI2    200
    00001       ADDMULI2    201
    00010       ADDORI2 202
    00011       ADDXORI2    203
    00100       MULADDI2    204
    00101       MULANDI2    205
    00110       MULORI2 206
    00111       MULXORI2    207
    01000       RSUBANDI2   208
    01001       RSUBORI2    209
    01010       RSUBXORI2   210
    01011       ORADDI2 211
    01100       ORMULI2 212

11001100000000010000, op:5, r1:5, imm1:5, rd:5, imm2:32
    00000       SLLADDI2    232
    00001       SLLANDI2    233
    00010       SLLORI2 234
```

```
    00011       SLLRSUBI2   235
    00100       ANDSLLI2    236

1100110000000000010001, op:5, r1:5, imm1:5, rd:5, imm2:32, imm3:32
    00000       LPAI3   392

110011000000001, op:7, r1:5, rd:5, imm1:32, imm2:32, imm3:32
    0000000     MAMI3   264
    0000001     MPMI3   265
    0000010     MOMI3   266
    0000011     MPAI3   267
    0000100     MPOI3   268
    0000101     RORI3   269
    0000110     AMPI3   270

110011000000010, op:7, r1:5, rd:5, imm1:32, imm2:32, imm3:32, imm4:32
    0000000     MPMPI4  424
    0000001     MPOMI4  425

1101, op:4, r1:5, imm:11, imm:16
    0000         JNEC   1
84
    0001         JEQC   185
    0010         JLTC   186
    0011         JGEC   187
    0100         JGTC   188
    0101         JLEC   189
    0110         JLTUC  190
    0111         JGEUC  191
    1000         JLEUC  192
    1001         JGTUC  193
    1010         JWNEC  194
    1011         JWEQC  195

11100000, uimm:8 ENTERC 5

1110001, op:1, uimm:16
    0            ENTER      6
    1            SYSCALL    7
```

Приложение C

Подпрограммы стандартной библиотеки языка C

(Данное Приложение является неотъемлемой частью настоящей Рекомендации)

C.1 Введение

В данном Приложении описывается набор подпрограмм стандартной библиотеки C99 [b-ISO IEC 9899], который должен быть доступен для использования **клиенту ECI**. Для каждой функции определены операнды, передаваемые **клиентом ECI**, и возвращаемое значение.

Следует иметь в виду, что под строкой понимается последовательность ненулевых байтов, завершающаяся нулевым байтом.

Перечисленные ниже функции представляют собой функции стандартной библиотеки языка C. Во всех случаях первый параметр помещается в регистр R2 (поскольку регистр R1 содержит идентификатор функции clibfunc). В объявлении предполагается, что все значения параметров передаются как скалярные значения или указатели на нескалярные значения. Если библиотечная функция требует передачи нескалярного параметра по значению, то команда SYSCALL передает его по ссылке, а от виртуальной машины требуется преобразовать параметр в тот вид, которого требует библиотека.

Возвращаемые значения всегда представляют собой скалярные значения или указатели, передаваемые через регистр R1.

ПРИМЕЧАНИЕ. – Значение clibfunc образуется следующим образом:

((clibfunc >> 8) & 0x000000FF) = представленный в двоично-десятичном коде номер подраздела стандарта языка C, в котором описываются библиотечные функции. (В случае стандарта C99 номер главы – 7, а библиотека <string.h> описывается в подразделе 21.)

((clibfunc >> 4) & 0x0000000F) = тип функции в библиотеке – число, следующее за номером подраздела.

(clibfunc & 0x0000000F) = номер функции конкретного типа в библиотеке – число, следующее за номером типа функции.

Например, функция memmove() описывается в пункте 7.21.2.2 документа [b-ISO IEC 9899]. Поэтому clibfunc для данной функции кодируется как 0x00002122.

C.2 memmove

clibfunc: 0x00002122

Описание: копирование n байтов из области памяти, заданной указателем s2, в область памяти, заданную указателем s1. Области памяти могут пересекаться.

Объявление: uint8 * memmove(uint8 * s1, uint8 * s2, uint32 n)

Возвращаемое значение: s1

C.3 strcpy

clibfunc: 0x00002123

Описание: копирование строки (вместе с оконечным символом) из области памяти, заданной указателем s2, в область памяти, заданную указателем s1. Если области памяти пересекаются, результаты не определены.

Объявление: uint8 * strcpy(uint8 * s1, uint8 * s2)

Возвращаемое значение: s1

C.4 strncpy

clibfunc: 0x00002124

Описание: то же, что и для strcpy(), но копируется не более n байтов. Если длина s2 превышает n, в конец строки присоединяется нулевой байт (по смещению s1[n]).

Объявление: uint8 * strncpy(uint8 * s1, uint8 * s2, uint32 n)

Возвращаемое значение: s1

C.5 strcat

clibfunc: 0x00002131

Описание: присоединение копии строки, заданной указателем s2 (вместе с оконечным символом) в конец строки, заданной указателем s1. Если области памяти пересекаются, результаты не определены.

Объявление: uint8 * strcat(uint8 * s1, uint8 * s2)

Возвращаемое значение: s1

C.6 strncat

clibfunc: 0x00002132

Описание: присоединение копии строки, заданной указателем s2 (вместе с оконечным символом) в конец строки, заданной указателем s1, но с копированием не более n символов. Если длина s2 превышает n, то по смещению n + 1 байтов за последним ненулевым байтом исходной строки s1 присоединяется нулевой байт. Если области памяти пересекаются, результаты не определены.

Объявление: uint8 * strncat(uint8 * s1, uint8 * s2, uint32 n)

Возвращаемое значение: s1

C.7 memcmp

clibfunc: 0x00002141

Описание: сравнение первых n байтов в области памяти, заданной указателем s1, с первыми n байтами в области памяти, заданной указателем s2.

Объявление: uint32 memcmp(uint8 *s1, uint8 *s2, uint32 n)

Возвращаемое значение: R1 = 0 в случае совпадения, иначе R1 зависит от первой позиции, начиная слева, в которой обнаруживается несовпадение значений.

R1 > 0, если значение байта из s1 в этой позиции больше значения соответствующего байта из s2.

R1 < 0, если значение байта из s1 в этой позиции меньше значения соответствующего байта из s2.

C.8 strcmp

clibfunc: 0x00002142

Описание: сравнение строк, заданных указателями s1 и s2.

Объявление: uint32 strcmp(uint8 * s1, uint8 * s2)

Возвращаемое значение: R1 = 0 в случае совпадения, иначе R1 зависит от первой позиции, начиная слева, в которой обнаруживается несовпадение значений.

R1 > 0, если значение байта из s1 в этой позиции больше значения соответствующего байта из s2.

R1 < 0, если значение байта из s1 в этой позиции меньше значения соответствующего байта из s2.

C.9 strncmp

clibfunc: 0x00002144

Описание: сравнение строк, заданных указателями s1 и s2, но только до n байтов.

Объявление: uint32 strncmp(uint8 * s1, uint8 * s2, uint32 n)

Возвращаемое значение: R1 = 0 в случае совпадения, иначе R1 зависит от первой позиции, начиная слева, в которой обнаруживается несовпадение значений.

R1 > 0, если значение байта из s1 в этой позиции больше значения соответствующего байта из s2.

R1 < 0, если значение байта из s1 в этой позиции меньше значения соответствующего байта из s2.

C.10 memchr

clibfunc: 0x00002151

Описание: поиск первого байта со значением c в границах n байтов, заданных указателем s.

Объявление: uint8 * memchr(uint8 *s, uint8 c, uint32 n)

Возвращаемое значение: указатель на найденный байт или 0, если байт с таким значением не найден.

C.11 strchr

clibfunc: 0x00002152

Описание: поиск первого байта со значением c в строке, заданной указателем s, до оконечного (нулевого) байта включительно.

Объявление: uint8 * strchr(uint8 * s, uint8 c)

Возвращаемое значение: указатель на найденный байт или 0, если байт с таким значением не найден.

C.12 strcspn

clibfunc: 0x00002153

Описание: вычисление длины наибольшего начального фрагмента строки, заданной указателем s1, который состоит только из байтов, не входящих в строку, заданную указателем s2.

Объявление: uint32 strcspn(uint8 * s1, uint8 * s2)

Возвращаемое значение: вычисленное значение длины.

C.13 strpbrk

clibfunc: 0x00002154

Описание: поиск первого появления любого байта в строке, заданной указателем s2, в строке, заданной указателем s1.

Объявление: uint32 strpbrk(uint8 * s1, uint8 * s2)

Возвращаемое значение: положение первого байта, отвечающего условию, или 0, если таких байтов не найдено.

C.14 strrchr

clibfunc: 0x00002155

Описание: поиск последнего байта со значением c в строке, заданной указателем s, до оконечного (нулевого) байта включительно.

Объявление: uint8 * strrchr(uint8 * s, uint8 c)

Возвращаемое значение: указатель на найденный байт или 0, если байт с таким значением не найден.

C.15 strspn

clibfunc: 0x00002156

Описание: вычисление длины наибольшего начального фрагмента строки, заданной указателем s1, который состоит только из байтов, входящих в строку, заданную указателем s2.

Объявления: uint32 strspn(uint8 * s1, uint8 * s2)

Возвращаемое значение: вычисленное значение.

C.16 strstr

clibfunc: 0x00002157

Описание: поиск первого появления строки, заданной указателем s1 (за исключением оконечного байта), в строке, заданной указателем s2.

Объявление: uint8 strstr(uint8 * s1, uint8 * s2)

Возвращаемое значение: указатель на найденную позицию или 0, если появление не найдено.

C.17 memset

clibfunc: 0x00002161

Описание: копирование младшего байта c в область памяти, заданную указателем s, в n экземплярах.

Объявление: uint8 * memset(uint8 * s, uint8 c, uint32 n)

Возвращаемое значение: s.

Приложение D

Формат файла клиента ECI

(Данное Приложение является неотъемлемой частью настоящей Рекомендации)

Файл образа **клиента ECI** должен соответствовать спецификации формата объектных файлов ELF [ETSI GS ECI 001-4]. В данном Приложении содержится конкретная информация, необходимая для соблюдения требований спецификации виртуальной машины. Поскольку ВМ поддерживает 32-битовую архитектуру с порядком байтов от младшего к старшему, для идентификации ELF-файла в *e_ident* [ETSI GS ECI 001-4] должны использоваться значения, приведенные в таблице D.1.

Таблица D.1 – Строки *e_ident*, соответствующие стандарту ECI

Имя	Значение
e_ident[EI_CLASS]	ELFCLASS32
e_ident[EI_DATA]	ELFDATA2LSB

В таблице D.2 приведены значения, которые должны использоваться для некоторых полей в заголовке ELF-файла.

Таблица D.2 – Значения полей заголовка ELF-файла, соответствующие стандарту ECI

Имя	Значение
e_type	ET_EXEC
e_machine	ET_NONE
e_version	EV_CURRENT

Загрузчик должен отклонять любой файл образа **клиента ECI** со значениями, отличными от тех, которые указаны в настоящем Приложении.

Дополнение I

Тематические области, требующие доработки

(Данное Дополнение не является неотъемлемой частью настоящей Рекомендации)

Определено, что настоящая Рекомендация нуждается в доработке и валидации, с тем чтобы обеспечить ее соответствие требованиям, установленным в [ITU-T J.1010], и что необходимо обновить Рекомендацию [ITU-T J.1010], отразив в ней требования спецификации системы расширенной защиты контента MovieLabs (ECP) [b-ECP]. Рекомендации [ITU-T J.1011], [ITU-T J.1012], [ITU-T J.1013], МСЭ-Т J.1014, [ITU-T J.1015] и [b-ITU-T J.1015.1] следует в дальнейшем обновить, отразив в них указанные изменения в [ITU-T J.1010].

Ряд Государств – Членов МСЭ наряду с заинтересованными сторонами со всего мира, представляющими самые разные отрасли, включая производителей устройств и электронных компонентов, владельцев и лицензиатов авторских прав на контент, поставщиков услуг на базе технологии over-the-top (OTT) и линейного телевидения, а также поставщиков решений для систем условного доступа (CAS) и управления цифровыми правами (DRM), выразили обеспокоенность тем, что встроенный общий интерфейс (ECI) не в полной мере отвечает требованиям ECP и требованиям к защите контента, предъявляемым в более широком круге отраслей.

Соответствующие вопросы были, в частности, подняты во вкладах к собранию 9-й Исследовательской комиссии МСЭ-Т (ИК9) (16–23 апреля 2020 года). Во вкладах, представленных Израилем, Австралией, Членом Сектора МСЭ-Т компанией Samsung, а также Ассоциированными членами ИК9 компаниями Sky Group и MovieLabs, предлагалось внести ряд изменений в Рекомендации по тематике ECI, однако согласие по ним достигнуто не было. Эти предложения перечислены в [b-SG9 Report 17 Ann.1].

Они состояли, в частности, в следующем:

1) упростить систему ECI, ограничив сферу ее применения;
2) отказаться от DRM;
3) отказаться от повторного шифрования контента;
4) отказаться от управления программным обеспечением;
5) добавить интерфейсы API для защищенного хранения и криптографических операций;
6) предусмотреть возможность использования лестниц ключей, определяемых поставщиком;
7) установить требования к TEE, изложенные в Рекомендации J.1207;
8) включить в Рекомендации реализацию TEE для виртуальной машины;
9) применять более стойкие алгоритмы шифрования, например SHA-384;
10) использовать стандартные сертификаты, подобные приведенным в Рекомендации МСЭ-Т X.509;
11) пересмотреть обмен данными между клиентами;
12) организовать дополнительное взаимодействие с ETCИ;
13) провести дополнительное коллегиальное рассмотрение;
14) рассмотреть возможные альтернативы модели доверительного органа;
15) уточнить технические аспекты правил соответствия и обеспечения устойчивости ECI;
16) добавить требования об обеспечении технического разнообразия, например о рандомизации распределения адресного пространства;
17) добавить требования о проверке целостности данных на этапе выполнения.

Эти предложения отражают постоянную эволюцию защиты контента и способов ее нарушения. Первоначальный замысел ECI возник почти за десять лет до утверждения настоящей Рекомендации

МСЭ-Т. Системы, подобные ECI, необходимо регулярно оценивать на предмет стойкости к современным методам осуществления атак, а также соответствия отраслевым требованиям к защите.

Существуют и другие механизмы обеспечения функциональной совместимости. В частности, что касается применения DRM, большинство интернет-служб доставки видео внедрили другие решения, обеспечивающие функциональную совместимость наряду с решением стоящих перед этими службами задач.

Важной задачей является внесение большей ясности, так как многие Государства-Члены рассматривают стандарты МСЭ как авторитетные источники руководящих указаний по развитию их рынков и отраслей. Упомянутый выше список предложений призван способствовать тому, чтобы можно было в полной мере уяснить все последствия, связанные с настоящей Рекомендацией МСЭ-Т, при внедрении ECI на рынках этих государств и учесть все возможные вопросы при рассмотрении законодательных и нормативных актов и потребностей рынка, требующих обеспечения функциональной совместимости потребительского цифрового телевизионного оборудования. Он также позволяет производителям оборудования, предпочитающим брать за основу при проектировании особые наборы требований или иные стандарты, учитывать эти вопросы в процессе разработки продукции, предназначенной для различных рынков.

Библиография

[b-ITU-T J.1010] Рекомендация МСЭ-Т J.1010 (2016 г.), *Встроенный общий интерфейс для заменяемых решений CA/DRM; сценарии использования и требования*.

[b-ITU-T J.1011] Рекомендация МСЭ-Т J.1011 (2016 г.), *Встроенный общий интерфейс для заменяемых решений CA/DRM; архитектура, определения и обзор*.

[b-ITU-T J.1014] Рекомендация МСЭ-Т J.1014 (2020 г.), *Встроенный общий интерфейс для заменяемых решений CA/DRM; усовершенствованная система безопасности – ориентированные на ECI функциональные возможности*.

[b-ITU-T J.1015] Recommendation ITU-T J.1015 (2020), *Embedded common interface for exchangeable CA/DRM solutions; The advanced security system – Key ladder block.*

[b-ITU-T J.1015.1] Recommendation ITU-T J.1015.1 (2020), *Embedded common interface for exchangeable CA/DRM solutions; Advanced security system – Key ladder block: Authentication of control word-usage rules information and associated data 1*.

[b-ITU-T J-Suppl.7] ITU-T J-series Recommendations – Supplement 7 (2020), *Embedded common interface (ECI) for exchangeable CA/DRM solutions; Guidelines for the implementation of ECI (EG).*

[b-SG9 Report 17 Ann.1] ITU-T SG9 meeting report, SG9-R17-Annex 1 (2020), Annex 1 to Report 17 of the SG9 fully virtual meeting held 16-23 April 2020.
https://www.itu.int/md/T17-SG09-R-0017/en

[b-ECP] MovieLabs Specification for Enhanced Content Protection – Version 1.2. Доступно по адресу: <https://movielabs.com/ngvideo/MovieLabs_ECP_Spec_v1.2.pdf>

[b-ETSI GS ECI 001-3] ETSI GS ECI 001-3 (2017), *Embedded Common Interface (ECI) for exchangeable CA/DRM solutions; Part 3: CA/DRM Container, Loader, Interfaces, Revocation*.

[b-ISO IEC 9899] ISO/IEC 9899:2018 *Information technology – Programming Languages – C.*

[b-TIS-ELF] TIS Committee (1995), *Tool Interface Standard (TIS) Executable and Linking Format (ELF) Specification, version 1.2*.
Доступно по адресу: <https://refspecs.linuxfoundation.org/elf/elf.pdf>

СЕРИИ РЕКОМЕНДАЦИЙ МСЭ-Т

Серия A	Организация работы МСЭ-Т
Серия D	Принципы тарификации и учета и экономические и стратегические вопросы международной электросвязи/ИКТ
Серия E	Общая эксплуатация сети, телефонная служба, функционирование служб и человеческие факторы
Серия F	Нетелефонные службы электросвязи
Серия G	Системы и среда передачи, цифровые системы и сети
Серия H	Аудиовизуальные и мультимедийные системы
Серия I	Цифровая сеть с интеграцией служб
Серия J	**Кабельные сети и передача сигналов телевизионных и звуковых программ и других мультимедийных сигналов**
Серия K	Защита от помех
Серия L	Окружающая среда и ИКТ, изменение климата, электронные отходы, энергоэффективность; конструкция, прокладка и защита кабелей и других элементов линейно-кабельных сооружений
Серия M	Управление электросвязью, включая СУЭ и техническое обслуживание сетей
Серия N	Техническое обслуживание: международные каналы передачи звуковых и телевизионных программ
Серия O	Требования к измерительной аппаратуре
Серия P	Качество телефонной передачи, телефонные установки, сети местных линий
Серия Q	Коммутация и сигнализация, а также соответствующие измерения и испытания
Серия R	Телеграфная передача
Серия S	Оконечное оборудование для телеграфных служб
Серия T	Оконечное оборудование для телематических служб
Серия U	Телеграфная коммутация
Серия V	Передача данных по телефонной сети
Серия X	Сети передачи данных, взаимосвязь открытых систем и безопасность
Серия Y	Глобальная информационная инфраструктура, аспекты межсетевого протокола, сети последующих поколений, интернет вещей и "умные" города
Серия Z	Языки и общие аспекты программного обеспечения для систем электросвязи

Отпечатано в Швейцарии
Женева, 2020 г.