Unión Internacional de Telecomunicaciones

UIT-T J.1013

(04/2020)

SECTOR DE NORMALIZACIÓN
DE LAS TELECOMUNICACIONES
DE LA UIT

SERIE J: REDES DE CABLE Y TRANSMISIÓN DE PROGRAMAS RADIOFÓNICOS Y TELEVISIVOS, Y DE OTRAS SEÑALES MULTIMEDIA

Acceso condicional y protección – Soluciones de acceso condicional insertadas e intercambiables y de gestión digital de los derechos

Interfaz común integrada para soluciones CA/DRM intercambiables; la máquina virtual

Recomendación UIT-T J.1013

Recomendación UIT-T J.1013

Interfaz común integrada para soluciones CA/DRM intercambiables; la máquina virtual

Resumen

La Recomendación UIT-T J.1013 forma parte de un conjunto de publicaciones que abarca la máquina virtual de la especificación de la interfaz común integrada (ECI) para soluciones de acceso condicional intercambiable/gestión de derechos digitales (CA/DRM).

Esta Recomendación UIT-T es una transposición de la norma ETSI GS ECI 001-4 y es el resultado de la colaboración entre la CE 9 del UIT-T y el ETSI ISG ECI. Se ha hecho una pequeña modificación en la cláusula 7.3.7.1.

Historia

Edición	Recomendación	Aprobación	Comisión de Estudio	ID único*
1.0	ITU-T J.1013	2020-04-23	9	11.1002/1000/13574

Palabras clave

CA, DRM, intercambio.

* Para acceder a la Recomendación, sírvase digitar el URL http://handle.itu.int/ en el campo de dirección del navegador, seguido por el identificador único de la Recomendación. Por ejemplo, http://handle.itu.int/11.1002/1000/11830-en.

PREFACIO

La Unión Internacional de Telecomunicaciones (UIT) es el organismo especializado de las Naciones Unidas en el campo de las telecomunicaciones y de las tecnologías de la información y la comunicación. El Sector de Normalización de las Telecomunicaciones de la UIT (UIT-T) es un órgano permanente de la UIT. Este órgano estudia los aspectos técnicos, de explotación y tarifarios y publica Recomendaciones sobre los mismos, con miras a la normalización de las telecomunicaciones en el plano mundial.

La Asamblea Mundial de Normalización de las Telecomunicaciones (AMNT), que se celebra cada cuatro años, establece los temas que han de estudiar las Comisiones de Estudio del UIT-T, que a su vez producen Recomendaciones sobre dichos temas.

La aprobación de Recomendaciones por los Miembros del UIT-T es el objeto del procedimiento establecido en la Resolución 1 de la AMNT.

En ciertos sectores de la tecnología de la información que corresponden a la esfera de competencia del UIT-T, se preparan las normas necesarias en colaboración con la ISO y la CEI.

NOTA

En esta Recomendación, la expresión "Administración" se utiliza para designar, en forma abreviada, tanto una administración de telecomunicaciones como una empresa de explotación reconocida de telecomunicaciones.

La observancia de esta Recomendación es voluntaria. Ahora bien, la Recomendación puede contener ciertas disposiciones obligatorias (para asegurar, por ejemplo, la aplicabilidad o la interoperabilidad), por lo que la observancia se consigue con el cumplimiento exacto y puntual de todas las disposiciones obligatorias. La obligatoriedad de un elemento preceptivo o requisito se expresa mediante las frases "tener que, haber de, hay que + infinitivo" o el verbo principal en tiempo futuro simple de mandato, en modo afirmativo o negativo. El hecho de que se utilice esta formulación no entraña que la observancia se imponga a ninguna de las partes.

PROPIEDAD INTELECTUAL

La UIT señala a la atención la posibilidad de que la utilización o aplicación de la presente Recomendación suponga el empleo de un derecho de propiedad intelectual reivindicado. La UIT no adopta ninguna posición en cuanto a la demostración, validez o aplicabilidad de los derechos de propiedad intelectual reivindicados, ya sea por los miembros de la UIT o por terceros ajenos al proceso de elaboración de Recomendaciones.

En la fecha de aprobación de la presente Recomendación, la UIT [ha recibido/no ha recibido] notificación de propiedad intelectual, protegida por patente, que puede ser necesaria para aplicar esta Recomendación. Sin embargo, debe señalarse a los usuarios que puede que esta información no se encuentre totalmente actualizada al respecto, por lo que se les insta encarecidamente a consultar la base de datos sobre patentes de la TSB en la dirección http://www.itu.int/ITU-T/ipr/.

ÍNDICE

Página

Introducción

Esta Recomendación UIT-T[1] es una transposición de la norma ETSI GS ECI 001-4 y es el resultado de la colaboración entre la CE 9 del UIT-T y el ETSI ISG ECI. Se ha hecho una pequeña modificación en la cláusula 7.3.7.1.

El objetivo de esta Recomendación es facilitar la interoperabilidad y la competencia en los servicios de comunicaciones electrónicas y, en particular, en el mercado de los dispositivos de radiodifusión y audiovisuales. Sin embargo, existen otras tecnologías disponibles que también pueden resultar adecuadas y beneficiosas, en función de las circunstancias de los Estados Miembros.

En esta Recomendación se describe el concepto de máquina virtual, que se ejecuta en un entorno aislado y ofrece una serie de instrucciones y funciones de instrucción al sistema. La máquina virtual (VM, *virtual machine*) está diseñada para funcionar en diferentes entornos e interoperar otras aplicaciones presentes en la misma máquina utilizando interfaces bien definidas. Ofrece una combinación de soporte de su propio conjunto de instrucciones y un mecanismo modular para la ejecución de elementos escritos en el **código nativo**[2] de la unidad central de procesamiento (CPU) del **anfitrión ECI**, además de interactuar con el hardware y otros elementos del entorno del **anfitrión ECI**. De este modo la VM dispone de los medios necesarios para ejecutar código fácilmente renovable que puede facilitar una amplia gama de aplicaciones potencialmente seguras, incluida la implementación de clientes CA/DRM.

---

1 En el Apéndice I se han identificado varias áreas para su desarrollo ulterior.

2 El uso de la letra negrita en el texto de esta Recomendación indica términos con definiciones específicas para el contexto de la interfaz común incorporada que pueden diferir del uso común.

Recomendación UIT-T J.1013

Interfaz común integrada para soluciones CA/DRM intercambiables; la máquina virtual

1 Alcance

En esta Recomendación se especifica una máquina virtual destinada a integrarse en la implementación de receptores de televisión digital y descodificadores, y que puede crear un entorno seguro para la ejecución de aplicaciones del núcleo con acceso condicional o aplicaciones cliente de gestión de derechos digitales. El objetivo es crear un entorno de ejecución uniforme en el que tales clientes puedan operar sabiendo que se cumplen los requisitos de calidad de funcionamiento del **anfitrión ECI** mínimos, que se ofrece una API normalizada para la extracción de datos de seguridad fundamentales del contenido (es decir, encapsulados en el contenido) o a través de redes externas (por ejemplo, Internet) y donde es posible acceder a los recursos desde el entorno del **anfitrión ECI** de manera normalizada. Véanse también [b-UIT-T J.1010] y [b-UIT-T J.1011].

La presencia y utilización de la VM permite intercambiar clientes CA/DRM sin límite y soportar simultáneamente múltiples instancias de esos clientes en los **anfitriones ECI**. De esta manera se garantiza que los usuarios y los operadores no están limitados a un proveedor de protección de contenido concreto y se facilita la utilización de distintos tipos de soluciones de seguridad para adaptarse a distintos tipos de contenido. Para los proveedores de sistemas de protección de contenido garantiza la disponibilidad de una plataforma de ejecución conocida que no requiere la integración específica en los dispositivos **anfitriones ECI** de todos y cada uno de los fabricantes.

2 Referencias

Las siguientes Recomendaciones UIT-T y demás referencias contienen disposiciones que, por referencia a las mismas en este texto, constituyen disposiciones de esta Recomendación. En la fecha de publicación, las ediciones citadas estaban en vigor. Todas las Recomendaciones y demás referencias están sujetas a revisión, por lo que se alienta a los usuarios de esta Recomendación a que consideren la posibilidad de aplicar la edición más reciente de las Recomendaciones y demás referencias que se indican a continuación. Se publica periódicamente una lista de las Recomendaciones UIT-T vigentes. En esta Recomendación, la referencia a un documento autónomo no le otorga, como documento independiente, el rango de Recomendación.

[UIT-T J.1012] Recomendación UIT-T J.1012 (2020), *Interfaz común integrada para soluciones CA/DRM intercambiables; Contenedor, cargador, interfaces y revocación CA/DRM*.

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

3 Definiciones

3.1 Términos definidos en otros documentos

Ninguna.

3.2 Términos definidos en esta Recomendación

En esta Recomendación se definen los siguientes términos:

3.2.1 bytecode: código del **cliente ECI** (suele comprender un núcleo de acceso condicional o un cliente de gestión de derechos digitales) que ejecuta la máquina virtual (VM).

3.2.2 equipo en las instalaciones del cliente (CPE): dispositivo del cliente que ofrece las funciones de encriptación y desencriptación especificadas de la interfaz común integrada (**ECI)**.

3.2.3 ECI (interfaz común integrada): arquitectura y sistema especificado en el ISG de ETSI "Embedded CI" (CI integrada), que permite la creación e instalación en equipos en las instalaciones del cliente (**CPE**) de **clientes ECI** intercambiables basados en software y que, por tanto, permiten la interoperabilidad de dispositivos **CPE** en relación con la **ECI**.

3.2.4 cliente ECI (cliente de la interfaz común integrada): realización de un cliente de acceso condicional/gestión de derechos digitales (CA/DRM) conforme con las especificaciones de la interfaz común integrada (**ECI)**.

3.2.5 anfitrión ECI: sistema de hardware y software de un **CPE**, que dispone de funcionalidades relacionadas con la **ECI** y que tiene interfaces con un **cliente ECI**.

3.2.6 código nativo: código programático escrito en el conjunto de instrucciones ejecutables nativas del procesador del **anfitrión ECI**.

3.2.7 instancia VM: instanciación de la VM creada por un **anfitrión ECI** que aparece ante el **cliente ECI** como un entorno de ejecución en el que opera.

4 Abreviaturas y acrónimos

En esta Recomendación se utilizan las siguientes abreviaturas y acrónimos:

API Interfaz de programación de aplicación (*application programming interface*)

CA Acceso condicional (*conditional access*)

CAS Sistema de acceso condicional (*conditional access system*)

CI Interfaz común (*common interface*)

CP Protección de contenido (*content protection*)

CPE Equipo en los locales del cliente (*customer premises equipment*)

CPU Unidad central de procesamiento (*central processing unit*)

DRM Gestión de derechos digitales (*digital rights management*)

ECI Interfaz común integrada (*embedded common interface*)

ECP Protección de contenido (*enhanced content protection*)

ELF Formato ejecutable y enlazable (*executable and linkable format*)

EPG Guía electrónica de programas (*electronic programme guide*)

ID Identificación/identidad/identificador (*identification/identity/identifier*)

OS Sistema operativo (*operating system*)

OTT Superpuesto (*over the top*)

PC Contador de programa (*program counter*)

POSIX Interfaz de sistema operativo portable (*portable operating system interface*)

RISC Ordenador con juego de instrucciones reducido (*reduced instruction set computer*)

VM Máquina virtual (*virtual machine*)

5 Convenios

En la presente Recomendación, cuando un término va en negrita significa que tienen un significado específico en el contexto de la ECI que puede diferir de la utilización corriente de ese término.

6 Principios conceptuales

6.1 La máquina virtual como unidad central de procesamiento

Fundamentalmente una máquina virtual (VM) está formada por una unidad central de procesamiento (CPU) virtual con sus propios códigos y memoria de datos y una serie de interfaces de sistema que facilitan el acceso a las funcionalidades de hardware de la máquina **anfitrión ECI**. La CPU emulada ejecuta código como una CPU de 32 bits virtual y en esta Recomendación el código que ejecuta se denomina **Bytecode** . Dado que la VM es una simulación de un procesador RISC general, puede ejecutar diversas aplicaciones.

6.2 Características de la máquina virtual

La VM creará un entorno monoproceso y monoconexión.

La interfaz con el hardware **anfitrión ECI** y otras funciones adoptan la forma de una biblioteca normalizada de instrucciones, denominadas SYSCALL. La instrucción SYSCALL es una de las instrucciones personalizadas de la VM y suele ejecutarse tras preparar los parámetros que necesita la rutina biblioteca (es decir, transmitidas en "registros" de la VM).

Todas las interacciones entre el **cliente** y el **anfitrión ECI** se realizan mediante esta operación. No se define una arquitectura de **interrupción** y, una vez iniciada, el **cliente ECI** se ejecuta hasta el final. Por consiguiente, no es posible invocar instrucciones en la VM. Aunque de este modo se restringe en cierta medida la flexibilidad, esto se compensa mediante un mejor control de la ejecución de la VM (garantizando la robustez de la operación), la supresión de las condiciones de competencia, de la interferencia con operaciones críticas en el tiempo, etc.

Por consiguiente, el único medio de transmitir datos o mensajes al **cliente ECI** que se ejecuta en la VM es mediante las solicitudes enviadas por el **cliente ECI** al invocar las SYSCALL correspondientes.

6.3 Aislamiento de los clientes ECI individuales

El **cliente ECI** se ejecuta en una máquina virtual presente como aplicación en el firmware del **anfitrión ECI.** Debe ser posible invocar múltiples instancias de la máquina virtual, cada una de ellas posiblemente ejecutando un **cliente ECI** distinto. Esto impone tres requisitos fundamentales al entorno operativo del **anfitrión ECI**:

1) Aislamiento de clientes ECI individuales – El Sistema Operativo debe atribuir suficientes recursos a cada **instancia VM** de manera que todas las instancias simultáneamente activas cumplan los requisitos de calidad de funcionamiento; los valores propuestos se estipulan en [b-UIT-T J-Suppl.7].
2) Las bibliotecas definidas en la cláusula 8 y el Anexo C deben permitir múltiples entradas o implementarse por separado para cada instancia de la VM.

3) El Sistema Operativo y la VM garantizarán que no es posible intercambiar información entre los **clientes ECI** que se ejecutan y el mundo exterior, incluidos otros **clientes ECI**, por medios distintos de los explícitamente especificados para ese fin como parte de la interfaz SYSCALL. Entre otras cosas, esto implica que toda la memoria cartografiada en el espacio de datos de una **interfaz VM** se limpia de antemano de todo contenido anterior y se impedirá todo intento de utilizar condiciones excepcionales en la VM para provocar un comportamiento no especificado. Esto implica también que el **cliente ECI** no podrá modificar su **Bytecode**. Y específicamente implica que el **anfitrión ECI** y la VM deberán proceder a todas las verificaciones necesarias para impedir que un **cliente ECI** provoque un comportamiento imprevisto en el **anfitrión ECI** o las implementaciones de la VM que pueda, por ejemplo, causar directa o indirectamente que el **cliente ECI** pueda manipular (piratear) el **anfitrión ECI**.

6.4 Especificación de la máquina virtual

En las siguientes cláusulas de esta Recomendación se detallan explícitamente los siguientes aspectos de la VM:

1) Arquitectura técnica de la VM.
2) Serie de instrucciones de la VM.
3) Interfaz **anfitrión ECI**.

6.5 Cargador de cliente ECI

Para ejecutar un **cliente ECI**, primero se cargará el **Bytecode** en el espacio de código de la memoria VM y se inicializará el espacio de datos. En la cláusula 9 se abordan algunos aspectos concretos del formato del contenedor del **cliente ECI** y la inicialización de la VM.

7 La máquina virtual

7.1 Entorno de ejecución

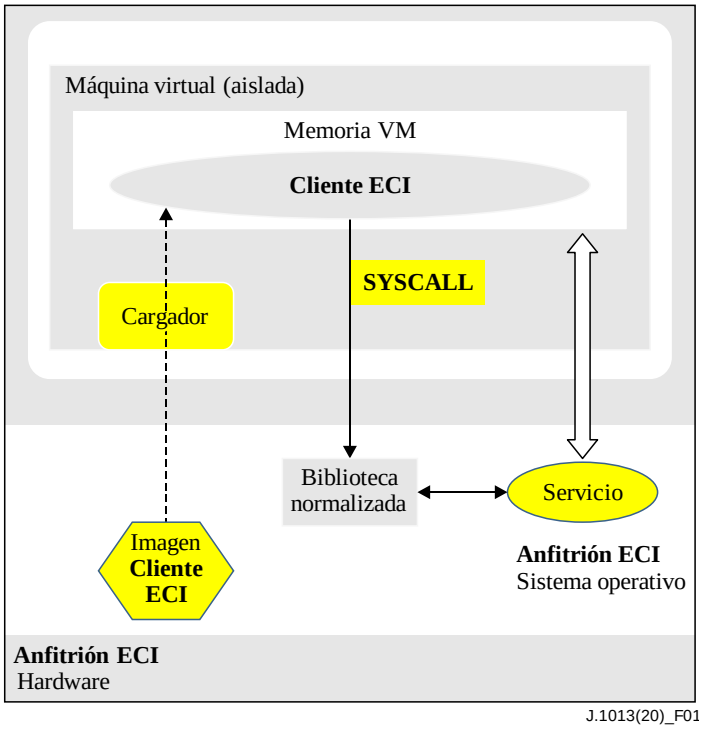

Figura 1 – Entorno del anfitrión VM

Como se muestra en la Figura 1, la VM se ejecutará en un entorno aislado que garantice el aislamiento del sistema operativo del **anfitrión ECI,** de otras instancias de máquina virtual y de cualquier otra aplicación que se ejecute en el **anfitrión ECI**.

La VM comprende una aplicación nativa del **anfitrión ECI**, con su correspondiente memoria, y una biblioteca interfaz y un cargador para instalar el **Bytecode** que forma un **cliente ECI**. La biblioteca interfaz ofrece al **cliente ECI** acceso a las funcionalidades del sistema operativo y el hardware del **anfitrión ECI,** además de a otras aplicaciones que puedan ejecutarse en el **anfitrión ECI** y con las que el **cliente ECI** puede tener que interactuar. Ejemplo típico de ello sería la interacción con la aplicación guía electrónica de programas (EPG), que requiere una autorización para mostrar determinados contenidos al usuario.

7.2 Arquitectura de la máquina virtual

7.2.1 Arquitectura de la CPU

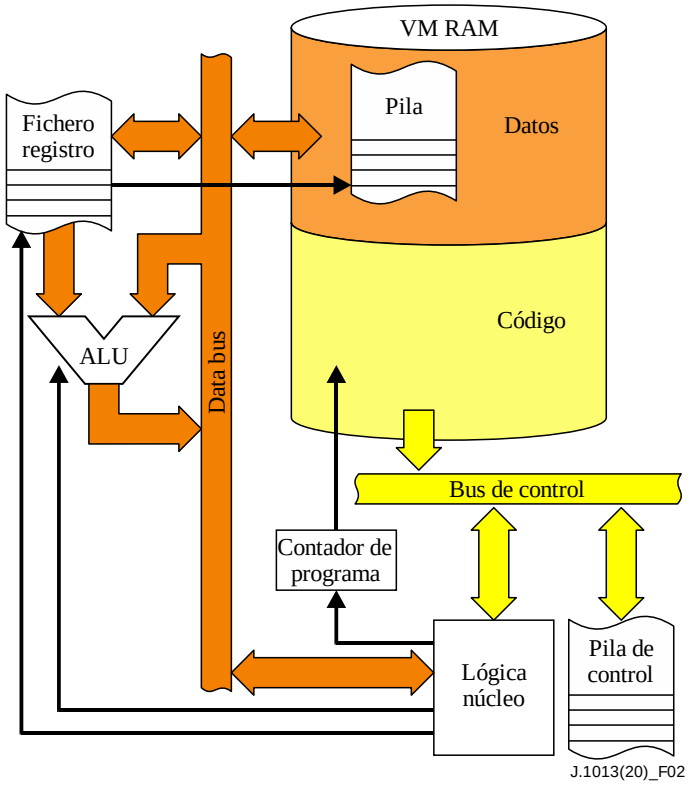

Figura 2 – Arquitectura del procesador virtual

En la Figura 2 se muestra la arquitectura de la CPU de la máquina virtual. La VM es una máquina de registro que tiene las siguientes características:

– Un fichero registro con registros de carácter general de 32 bits. Los registros se organizan en ventanas registro. Cada ventana registro contiene 32 registros. Los últimos 16 registros de cada ventana se solapan con los primeros 16 registros de la ventana siguiente. Dos de estos registros en cada ventana sirven de puntero de pila y puntero de trama. El número total de registros en el fichero registro es REGISTER_FILE_SIZE, como se especifica en el Anexo A.

– Una arquitectura Harvard CPU. Los datos se almacenan en un espacio memoria plano de 32 bits. El código se almacena en un espacio memoria no direccionable de lectura únicamente.

– Una pila de control independiente se ocupa del seguimiento de las direcciones de retorno. El contenido de esta pila es inaccesible para el **Bytecode** o las aplicaciones externas. La pila puede almacenar hasta CONTROL_STACK_SIZE direcciones de retorno. (Véase el Anexo A).
– Instrucciones de carga y almacenamiento para bytes con y sin signo y tipos de datos palabra y media palabra de 8, 16 y 32 bits respectivamente.
– Un conjunto de instrucciones con muchas instrucciones de procesamiento de datos personalizadas para el dominio de aplicación.
– Ordenamiento de bytes nativos para una carga y almacenamiento eficaces, independientemente de su ordenamiento. Se utiliza el alineamiento (alineamiento = tamaño) natural para los tipos básicos a fin de maximizar la portabilidad del **Bytecode.** Dicho de otro modo, la dirección de memoria de una media palabra siempre es par y la dirección de una palabra siempre es múltiplo de cuatro.
– Una instrucción al sistema (SYSCALL) que puede utilizarse para implementar servicios de sistema. Esto permite también ampliar la VM con funciones integradas, por ejemplo, para procesar datos de frecuente recurrencia de manera nativa.
– Memoria paginada que soporta la fragmentación del espacio memoria. Permite establecer una correspondencia entre la memoria nativa y el espacio memoria de la VM.

7.2.2 Registros

En cada ventana de registro pueden verse 32 registros, de R0 a R31. Se reservan dos registros para tratamiento especial. R0 es el puntero de trama y R16 es el puntero de pila. A continuación se detalla la utilización de estos registros.

Al principio de una función, la instrucción ENTER sube 16 registros en la ventana de registro, lo que hace que el antiguo puntero de pila sea ahora el nuevo puntero de trama y crea un nuevo puntero de pila, además de 15 nuevos registros. El nuevo puntero de pila se inicializa restando el tamaño de trama indicado en la instrucción ENTER del puntero de trama.

La instrucción RETURN invierte este proceso. Baja 16 registros en la ventana de registro, restaurando así los antiguos puntero de trama y puntero de pila.

Dado que los R0 a R15 originales no pueden alcanzarse desde la rutina llamada, automáticamente son guardados por el llamado. Dado que la dirección de retorno se guarda en una pila de control distinta, no se utilizan pilas de datos para los registros guardados por el llamado y las direcciones de retorno.

El número verdadero de registros es limitado, por lo que la profundidad de llamada de un **cliente ECI** está limitada (CONTROL_STACK_SIZE). Superar esta profundidad abortaría el programa VM. El número de registros y la correspondiente profundidad de la pila de control pueden especificarse al crear el proceso VM.

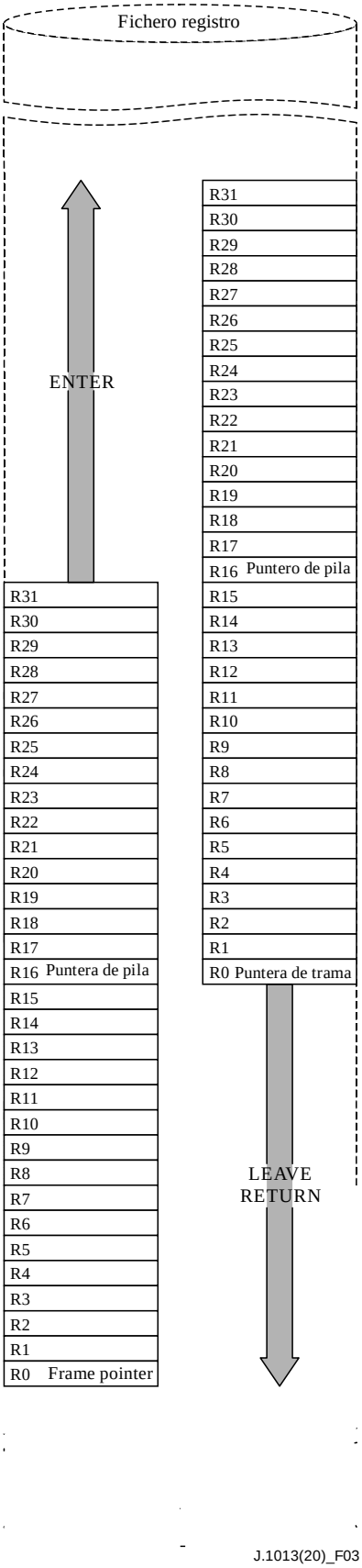

Figura 3 – Arquitectura del fichero registro

7.2.3 Espacio datos

La dirección de datos de base de la VM, definida como DATA_BASE_ADDRESS (véase el Anexo A), será 0x1000000 (16 Mbytes). La dirección más pequeña por encima de la memoria direccionable que no es direccionable es DATA_BASE_ADDRESS + ADDRESSABLE_DATA_SIZE (véase el Anexo A). La dirección de base de la pila estará definida por la implementación de la VM, pero siempre se encontrará hacia el extremo superior del espacio dirección. La VM puede reservar un máximo de VM_RESEVED_SIZE (véase el Anexo A) para fines privados en el espacio dirección del **cliente ECI** "por debajo" del fondo de la pila (en la dirección superior). El **cliente ECI** puede suponer que la parte superior de la heap (vacía) en la inicialización es igual al tamaño de los datos inicializados + el tamaño de los segmentos de datos no inicializados, ambos redondeados a un múltiplo de 4.

En la Figura 4 se ilustra el diseño de la memoria de datos.

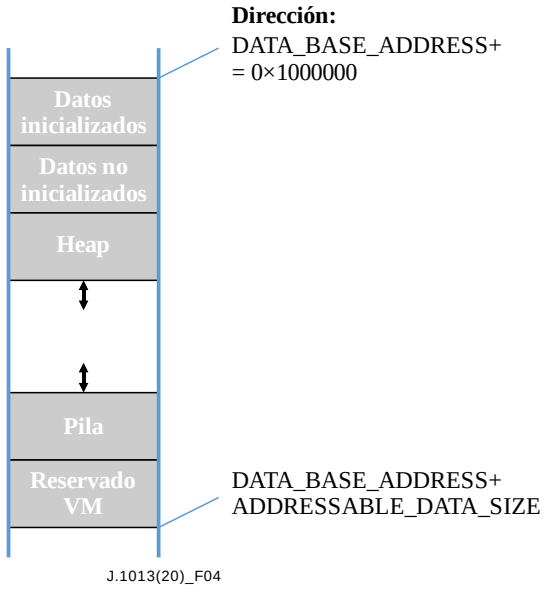

Figura 4 – Diseño de la memoria de datos de la VM

En la inicialización del **cliente ECI,** el cargador del **cliente ECI** cargará los segmentos de datos inicializados y no inicializados empezando por la dirección DATA_BASE_ADDRESS. Todos los bytes del segmento de datos no inicializados se pondrán a cero. El segmento de datos inicializados no está protegido contra la escritura.

NOTA 1 – El tamaño de la pila está limitado en principio. Dado que las estructuras de datos locales definidas en las funciones c suelen estar atribuidas a la pila, el **cliente ECI** pondrá el segmento pila al tamaño correspondiente en caso de que en el código c se utilicen variables locales grandes.

NOTA 2 – El **anfitrión ECI** puede cartografiar las memorias intermedias de mensaje en la memoria reservada VM por debajo de la dirección de la pila de base.

En futuras versiones de la VM se podrá reservar más memoria direccionable para los **clientes ECI**, es decir, que podrán tener una mayor ADDRESSABLE_DATA_SIZE. Por motivos de compatibilidad con versiones anteriores, los **clientes ECI** no dependerán de un valor o gama de valores específicos del puntero de pila que aquí se definen, sino que simplemente utilizarán el puntero de pila transmitido en la inicialización.

El cargador del **cliente ECI** no cargará ficheros de imagen que no se ajustan al convenio de diseño de memoria anterior para los segmentos de datos inicializados y no inicializados.

7.2.4 Espacio código

El programa no puede acceder directamente al código. El programa puede obtener referencias opacas de 32 bits a objetos de código estáticos (por ejemplo, punto de entrada de la rutina, objetivo de salto) denominadas *referencias al código* (véase la instrucción MOVF). Las *referencias al código* sólo podrán utilizarse con instrucciones de flujo de control indirectas (JMPR y CALLR). Las *referencias al código* no son punteros al espacio memoria de código, ni se utiliza con ellas una aritmética de punteros.

La dirección inicial del segmento de código en el espacio dirección de código será 0x00000000. El tamaño máximo del segmento de código se define como CODE_SIZE (véase el Anexo A).

7.2.5 Pila

Por convención se supone que la pila se sitúa en la memoria de datos, contiene sólo palabras, aumenta hacia las direcciones inferiores y su extremo (última palabra introducida) siempre está apuntada por el registro R16 (*puntero de pila*) de la ventana de registro actual.

R0, el puntero de trama, se utiliza como puntero en la pila del llamado, de manera que una rutina llamada puede acceder a los parámetros o datos de otro tipo introducidos en la pila (véase la cláusula 7.2.8).

7.2.6 Ordenamiento

En la memoria del sistema los datos formados por múltiples bytes (palabras y medias palabras) se representan en orden creciente. El software del **cliente ECI** utilizará el orden creciente.

7.2.7 Excepciones

La CPU VM no genera excepciones durante la ejecución. Si una instrucción opera en condiciones distintas de las indicadas en esta Recomendación (por ejemplo, acceso no alineado a una palabra o media palabra en la memoria, acceso a las direcciones de memoria sin memoria correspondiente, derivación a una referencia de código desconocida), el comportamiento no está definido. La VM puede optar por terminar el núcleo. La VM garantizará que bajo ninguna circunstancia un **cliente ECI** que no opere de conformidad con la presente Recomendación pueda acceder a datos no autorizados o influir en cualquier otra aplicación.

7.2.8 Convenio de llamada

Los convenios de llamada transmiten los siete primeros parámetros escalares (punteros y enteros) en R17 a R23. El llamado los verá como R1 a R7.

Los parámetros escalares tras el séptimo son transmitidos a la pila por el llamante de derecha a izquierda. Gracias al mecanismo de ventana de registro, el llamado siempre encontrará el octavo parámetro (de haberlo) apuntado por R0. Así, R0 es el *puntero de trama*. Los parámetros estructurales siempre se pasan a la pila o por referencia. Los punteros siempre se refieren al espacio memoria de la VM.

NOTA – Todas las SYSCALL pasan las estructuras y matrices únicamente por referencia. Este enfoque debe utilizarse también para otras instrucciones.

El llamado deja el valor de retorno en R1, que el llamante verá como R17. Los tipos de tamaño inferior a 32 bits se pasan (y devuelven) como valores de 32 bits.

El retorno estructural se ejecuta pasando un primer parámetro implícito, que es un puntero a la zona memoria donde se prevé almacenar el tipo retorno (pasado por referencia. El llamado escribe su resultado en la ubicación hacia donde apunta este parámetro. Este puntero de retorno se trata como un argumento normal (pasado en R17 → R1), lo que implica que los argumentos regulares de una función, que devuelven una estructura, pasan a otros registros de convenio de llamada (R18..R23 → R2..R7) o por la pila.

7.3 Conjunto de instrucciones de la máquina virtual

7.3.1 Notación

Se utiliza la siguiente notación:

```
rx       Register x.
uimm5    5 bit unsigned immediate.
uimms9   9 bit unsigned immediate. Always a multiple of two.
uimms10  10 bit unsigned immediate. Always a multiple of four.
simm11   11 bit signed immediate.
simm16   16 bit signed immediate.
uimm16   16 bit unsigned immediate.
pcr16    16 bit signed PC-relative
pcr24    24 bit signed PC-relative
imm32    32 bit immediate.
low8(x)  The least significant 8 bits of x.
low16(x) The least significant 16 bits of x.
```

Las descripciones funcionales utilizan la semántica C en tipos enteros de 32 bits. La capacidad de la operación para soportar tipos de datos con o sin signo se indica como observación. El acceso a la memoria viene dado por MEM1(), MEM2() y MEM4(), que dan acceso respectivamente a 1, 2 ó 4 bytes de memoria. El operando es un desplazamiento en el segmento de datos. Cuando proceda, MEM llevará el prefijo U para las operaciones sin signo y S para las operaciones con signo.

7.3.2 Instrucciones aritméticas

7.3.2.1 Operandos de registro

```
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 Registro, inmediato

```
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;
```

Las operaciones dividir y resto sin signo se ajusta a la definición C99: la división trunca el resultado matemático hacia cero; el resto respeta la relación:

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

donde % representa la función resto o el módulo.

El operando derecho de las instrucciones desplazamiento se situará en la horquilla [0, 31]. En caso contrario, el comportamiento no está definido. El desplazamiento a la derecha con signo copia el bit más significativo original en las posiciones vacías. Aritméticamente esto corresponde a la división con una potencia dos, redondeando el resultado matemático a menos infinito (redondeo *al valor inferior*).

7.3.3 Formas breves

Muchas instrucciones de los tres operandos utilizan uno de los operandos también como resultado. Dado que éstos pueden codificarse de manera más compacta, hay disponibles para ellos códigos operativos especiales:

```
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
```

Las operaciones inmediatas para bits prueban o modifican un solo bit. Estas operaciones inmediatas pueden codificarse con 5 bits indicando la posición de bit.

```
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);
```

Muchas comparaciones se efectúan con respecto a cero. Así se ahorra un operando inmediato y cuesta menos de emular.

```
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;
```

Las versiones sin signo de lo anterior no tienen sentido. Son verdaderas o falsas o pueden expresarse mediante `EQZ` o `NEZ`.

7.3.4 Flujo de control

7.3.4.1 Reglas comunes

Las instrucciones de flujo de control con operandos directos codifican sus objetivos en relación con la dirección final de la instrucción. En un flujo de control por registro, el registro ejerce la función de índice de puntero.

7.3.4.2 derivaciones incondicionales e instrucciones de función

```
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 Derivaciones condicionales

```
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 Derivaciones condicionales basadas en comparaciones de memoria con constante

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

Leen una palabra de la memoria y la comparan con una constante.

7.3.4.5 Derivaciones condicionales lejanas

Para cada una de las derivaciones condicionales descritas existe una version *lejana*, que tiene un desplazamiento de 24 bits. El ensamblador debe escoger la version más corta que convenga.

7.3.5 Instrucciones carga y almacenamiento

7.3.5.1 Registro + desplazamiento

```
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 Registro + desplazamiento corto

```
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 Indexado por registro

```
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 Indexado absoluto

```
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;
```

Cabe señalar que no es necesario cargar bytes con indexado absoluto. Por ejemplo, `LDSBAX` es equivalente a `LDSBI`.

7.3.5.5 Acceso a la pila dedicado

A continuación se indican cargas y almacenamientos de palabras que utilizan implícitamente el puntero de trama.

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

7.3.5.6 Transferencia de memoria

Instrucción copia de bloque utilizada para las copias de bloque generadas por compilador.

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

7.3.6 Instrucciones complejas

Las siguientes son instrucciones que realizan una combinación de operaciones, generalmente con operandos inmediatos. En este resumen cada operando designado `i1`, `i2`, etc. es una operación inmediata de 32 bits (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 Varios

7.3.7.1 Instrucciones al sistema

Diversos servicios se implementan mediante instrucciones al sistema.

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

Se implementa un conjunto mínimo de instrucciones al sistema de la interfaz de sistema operativo portable (POSIX) que tiene correspondencia directa en el OS subyacente. Se pueden añadir más servicios específicos de la aplicación.

7.3.7.2 Pseudoinstrucciones

Algunas operaciones pueden expresarse en términos de otras. Están disponibles los siguientes pseudocódigos operativos:

```
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 Interfaz entre el cliente ECI y el anfitrión ECI

8.1 Principios generales

Se dan instrucciones al sistema cuando se ejecuta una instrucción SYSCALL. La instrucción contiene un operando inmediato que identifica la instrucción al sistema. Las instrucciones al sistema son llamadas efectivas a una biblioteca normalizada en las que se pasan los parámetros descritos en la cláusula 7.2.8.

Los primeros siete parámetros (palabras o punteros) se pasan en los registros R1..R8. Si el tipo de parámetro real es un escalar de 8 ó 16 bits, todos se amplían con signo a valores de 32 bits. Los valores de retorno (palabras o punteros) se situarán en R1.

A menos que se indique lo contrario, todas las direcciones de memoria se refieren al espacio memoria VM.

Por motivos de compatibilidad futura, el **cliente ECI** pondrá a cero todos los registros R1..R8 que no se utilicen para pasar parámetros. La función biblioteca podrá suprimir el contenido de todos los registros.

A continuación se indican las instrucciones al sistema de biblioteca obligatorias que todas las implementaciones conformes deben soportar. En el formato utilizado se indica lo siguiente:

– El ID SYSCALL utilizado como operando inmediato (SYSCALL imm32).
– Una descripción de la función biblioteca.
– Una declaración en sintaxis C.
– Una descripción de los parámetros y el valor de retorno.
– Notas adicionales eventuales.

Los parámetros y valores de retorno se expresan de acuerdo con el siguiente convenio:

– **uint*nn*** representa un entero sin signo de *nn* bits (siendo *nn* 8, 16 ó 32). Los valores inferiores a 32 se ampliarán con ceros a 32 bits al situarlos en los registros.
– **int*nn*** representa un entero con signo. Los valores inferiores a 32 bits se ampliarán con signo al situarlos en los registros.
– **void *** representa un puntero genérico.
– **[u]int*nn* *** representa un puntero a un valor o una matriz de valores de tipo [u]int*nn*.
– **struct struct_type *** se refiere a un puntero a una estructura (o matriz de estructuras) en la memoria. Las estructuras siempre se pasan por referencia utilizando este convenio.

8.2 Valor de error

La mayoría de SYSCALL devuelve una palabra negativa para indicar que se ha detectado una condición de error. En el Cuadro 1 se enumeran los valores de error.

Cuadro 1 – Valores de error

Valor	Nombre simbólico	Significado
–49	EPERM	Se ha llamado a una SYSCALL o función CLIB inexistente.
–50	EINVAL	Uno de los parámetros es incorrecto
–51	ERRSYSCALLMSGQUEUE	El número de mensajes enviados al **anfitrión ECI** supera la capacidad de su memoria intermedia.
–52	ERRHEAPSIZE	Se ha solicitado un valor de tamaño de heap inadecuado
–53	ERRSTACKSIZE	Se ha solicitado un valor de tamaño de pila inadecuado

8.3 SYS_EXIT

SYSCALL ID: 0x0001

Descripción: Termina la VM, con un código motivo.

Declaración: void SYS_EXIT(uint32 reason).

Operandos: El **reason** de la terminación.

Devuelve: Nada.

NOTAS – El **reason** toma uno de los valores enumerados en el Cuadro 2.

Cuadro 2 – Valores de SYS_EXIT reason

reason	Significado
0	Terminación normal
0x00000001..0x7FFFFFFF	Condición de error, específica del proveedor del **cliente ECI**
0x80000000..0xFFFFFFFF	Reservado para utilización futura

8.4 SYS_PUTMSG

SYSCALL ID: 0x0003

Descripción: Envía un mensaje asíncrono (solicitud o respuesta).

Declaración: int32 SYS_PUTMSG(MessageBuffer *msg_buffer)

El formato de MessageBuffer se define en [UIT-T J.1012].

Operandos: **msg_buffer** es un puntero a un bloque memoria intermedia de mensajes.

Devuelve: El ID del mensaje asignado por el **anfitrión ECI** (valor de 16 bits no negativo) o cualquiera de los valores de error siguientes (negativo): **ERRSYSCALLMSGQUEUE** (Cuadro 1).

NOTAS – Se considera que la llamada no produce bloqueo en condiciones operativas del **anfitrión ECI** normales. El **anfitrión ECI** copia el contenido de msg_buffer y **el cliente ECI** puede reutilizarlo inmediatamente tras la devolución de la SYSCALL.

8.5 SYS_GETMSG

SYSCALL ID: 0x0004

Descripción: Extrae el siguiente mensaje (sea una solicitud o un resultado) del **anfitrión ECI**. La SYSCALL se bloquea si no hay mensajes disponibles.

Declaración: (MessageBuffer *) SYS_GETMSG()

El formato de MessageBuffer se define en [UIT-T J.1012].

Operandos: Ninguno.

Devuelve: El puntero a la memoria intermedia que contiene el siguiente mensaje del **anfitrión ECI** o cualquiera de los valores de error indicados en el Cuadro 1.

NOTAS – La llamada se bloqueará si el **anfitrión ECI** no tiene mensajes en cola para el **cliente ECI**. El **anfitrión ECI** no cambiará el contenido de la memoria intermedia de mensajes hasta la siguiente SYS_GETMSG SYSCALL. Los **clientes ECI** que quieran acceder a los datos de mensaje tras la siguiente llamada SYS_GETMSG deben copiar estos datos.

8.6 SYS_HEAPSIZE

SYSCALL ID: 0x0100

Descripción: Solicitud para que el **anfitrión ECI** cambie el tamaño de heap al parámetro indicado.

Declaración: int32 SYS_HEAPSIZE(uint32 heapsize)

Operandos: **heapsize**: tamaño al que se ha de poner la heap del **cliente ECI**. Será no negativo, múltiplo de 4 y no causará un desbordamiento de la heap en el segmento pila.

Devuelve: El desplazamiento de la ubicación de la memoria en bytes con respecto a la DATA_BASE_ADDRESS que sea la dirección de memoria no heap más baja en la memoria direccionable o cualquiera de los valores de error (negativo) siguientes: **ERRHEAPSIZE** (Cuadro 1).

NOTAS – La llamada se bloqueará en caso de que el **anfitrión ECI** no tenga mensajes en cola para el **cliente ECI**. Al inicializar el **cliente ECI**, SYS_HEAPSIZE(0) devolverá el desplazamiento del inicio del segmento heap (de tamaño cero en ese momento).

8.7 SYS_STACKSIZE

SYSCALL ID: 0x0200

Descripción: Solicitud para que el **anfitrión ECI** cambie el tamaño de la pila al parámetro indicado.

Declaración: int32 SYS_STACKSIZE(uint32 stacksize)

Operandos: **stacksize**: tamaño al que se ha de poner la heap del **cliente ECI**; será no negativo y múltiplo de 4 y no causará un desbordamiento de la pila en el segmento heap.

Devuelve: El desplazamiento de la ubicación de la memoria en bytes con respecto a DATA_BASE_ADDRESS que sea la dirección de memoria de pila más baja en la memoria direccionable o cualquiera de los valores de error (negativo) siguientes: **ERRSTACKSIZE** (Cuadro 1).

NOTAS – La llamada se bloqueará en caso de que el **anfitrión ECI** no tenga mensajes en cola para el **cliente ECI**.

8.8 SYS_SYNCCALL

SYSCALL ID: 0x1000

Descripción: El **cliente ECI** envía un mensaje síncrono al **anfitrión ECI** y suspende la ejecución hasta la devolución de la instrucción al sistema.

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

Operandos: **tag**: definición idéntica a la del campo MsgTag en la estructura MessageBuffer. El campo MsgFlags se pondrá a cero y el **anfitrión ECI** lo ignorará. **p1…pn**: parámetros de la llamada síncrona. Para get-messages con un resultado mayor que una entidad de 32 bits, **p1** es la dirección inicial de la ubicación de memoria a donde se devolverá el resultado, y **p2.. pn** son los parámetros del set-message. Todos los parámetros regulares, incluidas las estructuras y matrices, se pasan por referencia.

Devuelve: Para get-messages que devuelva un resultado que quepa en 32 bits, se devuelve el resultado. En caso contrario no hay resultado de retorno. Se ignoran todos los errores. Las configuraciones de parámetro erróneas simplemente no producen resultados y/o no tienen efecto. Los mensajes llamada devuelven un código estado definido por su semántica específica. Los resultados pueden devolverse a la ubicación de los parámetros puntero a la SYSCALL, como define la semántica de mensaje específica.

NOTAS – Esta SYSCALL no se bloqueará.

8.9 SYS_CLIB

SYSCALL ID: 0x0300

Descripción: Esta SYSCALL actúa como una interfaz de programación de aplicaciones (API) para que el **cliente ECI** pueda utilizar las funciones de biblioteca C normalizadas. En el Anexo C se detalla el conjunto de funciones soportadas.

Declaración: SYS_CLIB(uint32 clibfunc, etc.)

Operandos: **clibfunc** identifica la función de biblioteca C que se ha de llamar, como se indica en el Anexo C. Todos los demás operandos se definen en la lista de llamadas de función C.

Devuelve: Un valor de retorno, como se indica en la biblioteca del Anexo C, o cualquiera de los valores de error indicados en el Cuadro 1.

NOTAS – Como las distintas funciones de biblioteca C toman números y tipos de parámetros diferentes, éstos no se describen explícitamente en esta Recomendación. En el anexo se detalla el formato de todos los operandos. Todos los operandos de SYS_CLIB son valores escalares o punteros a valores no escalares en la memoria VM. Dado que algunas funciones de biblioteca C pueden tomar parámetros no escalares, la VM convertirá los parámetros pasados por valor antes de pasar la ejecución a la biblioteca.

9 Ciclo de Bytecode

9.1 Introducción

La VM se implementa como parte del firmware del **anfitrión ECI**. El sistema operativo del **anfitrión ECI** la carga/ejecuta dinámicamente cuando se ha de ejecutar un **cliente ECI**. Puede haber múltiples instancias para distintos **clientes ECI**, si tienen que estar simultáneamente disponibles.

El **cliente ECI** está escrito en el conjunto de instrucciones de la VM como se describe anteriormente. El vendedor del sistema CP (proveedor CA u operador DRM) lo prepara y pone a disposición del **anfitrión ECI** como imagen de código lógico. A nivel local, se transforma para adaptarse al diseño específico del **anfitrión ECI** y su entorno operativo y se carga en la VM cuando es necesario. Se ejecuta en la VM hasta que se termina deliberadamente (o se da una condición de error). Entonces se detiene la ejecución del **cliente ECI** y se termina la VM.

9.2 Carga de un nuevo cliente ECI en la VM

La VM actúa como un anfitrión intermediario para los **clientes ECI** externos exactamente como si el **cliente ECI** fuese una aplicación nativa ejecutándose en el dispositivo **anfitrión ECI**. La única diferencia es que el **cliente ECI** está instalado por el Sistema Operativo del dispositivo **anfitrión ECI** en la VM, en lugar de como una aplicación nativa.

Para cargar el **cliente ECI**, el subsistema VM crea en primer lugar un contexto procesador virtual. Para la carga, esto supone atribuir la memoria VM e instalar el código y los contenidos del segmento datos en ella como si fuesen aplicaciones nativas, pero donde el código y los datos inicializados facilitados en los ficheros con formato ejecutable y vinculable (ELF) [ETSI GS ECI 001-4] (véanse los detalles en el Anexo D) se transfieren a la memoria atribuida para la VM.

Dado que el segmento código no es accesible desde el programa, la implementación puede optar por llevar a cabo cualquier forma de preprocesamiento (por ejemplo, optimización) en el momento de la carga. De hecho, en esta Recomendación sólo se describe el formato del programa en la imagen. La representación interna depende por entero de la implementación. La única condición es que todas las referencias al código puedan ser utilizadas por el programa con la misma semántica.

También es posible preprocesar la imagen del **cliente ECI** cuando se extrae por primera vez para el dispositivo **anfitrión ECI** y se almacena de forma que pueda cargarse directamente cuando sea necesario. Se trata de una manera más eficaz de retener y lanzar **clientes ECI** cuando se descargan y recargan regularmente.

9.3 Inicialización de la VM

Se ha de crear el contexto CPU general de la VM, a saber, el fichero registro, la pila de control, las zonas de datos y pila y el Contador de Programas, más cualquier bandera y lógica de control/estado. Estos elementos no se detallan en esta Recomendación, pues dependen de la implementación.

El fichero registro se pone de manera que R0 se sitúe al principio del espacio fichero registro. Todos los registros se ponen a 0. Así, los registros Puntero de Trama y Puntero de Pila (R0 y R16, respectivamente) de la primera ventana se ponen de manera que, cuando se introduce la primera palabra en la pila, el Puntero de Pila se reduce previamente a –4 (0xFFFFFFFC) y la palabra se almacena ahí.

El Contador de Programas se inicializa al principio del segmento código, a menos que el miembro *e_entry* del encabezamiento ELF [ETSI GS ECI 001-4] en el fichero de imagen del **cliente ECI** tenga un valor distinto de cero, en cuyo caso el Contador de Programas se pone al valor (dirección virtual) especificado por el miembro *e_entry*. Se entrega entonces el control al componente ejecución de la VM, denominado "Bucle de Ejecución Central".

9.4 Bucle de Ejecución Central

La esencia de la VM se encuentra en el bucle de ejecución central, que lee y traduce cada instrucción secuencial al conjunto de acciones correspondientes en los registros, la memoria VM y/o las llamadas a la biblioteca. El bucle ejecuta instrucciones hasta que se da una excepción. La terminación del programa puede formar parte de la práctica de ejecución normal, por ejemplo si el programa ejecuta la instrucción al sistema `SYS_EXIT`, o puede ser resultado de un error, por ejemplo, si se desborda la pila de control.

Si se termina el "Bucle de Ejecución Central", la VM se cierra y deberá reinstanciarse cuando se vuelva a necesitar el **cliente ECI** en el futuro.

Anexo A

Recursos del sistema VM

(Este Anexo forma parte integrante de la presente Recomendación.)

En esta Recomendación se utilizan los siguientes parámetros para definir la calidad de funcionamiento de la VM. Los valores propuestos para los parámetros pueden encontrarse en [b-UIT-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

Anexo B

Códigos operativos para la VM

(Este Anexo forma parte integrante de la presente Recomendación.)

En la siguiente codificación se especifica cómo se codifican las instrucciones en la imagen binaria. A continuación se muestran distintos formatos. La línea resumen presenta una lista de campos separados por comas que forman la instrucción. Se trata de bits explícitos o de un nombre de campo seguido por un indicador de ancho de campo. Los distintos códigos operativos con el mismo formato se enumeran siguiendo el patrón correspondiente que ocupa el campo 'op'. Los bits se enumeran en orden decreciente.

Por ejemplo, `SUB R3,R5,R17` se codifica como:

```
1011 00001 00011 00101 10001
```

o, en cuartetos:

```
1011 0000 1000 1100 1011 0001
```

o, en bytes:

```
0xB0, 0x8C, 0xB1
```

Los nombres de instrucción van seguidos de un número, que es el número de código operativo definido de esa instrucción. Este número será el mismo en todas las futuras versiones del conjunto de instrucciones de la VM.

Los campos de los códigos operativos no tendrán más de cuatro bytes. Por consiguiente, los campos de 32 bits empezarán en el extremo del byte. Ningún campo tendrá más de 32 bits.

```
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

11001100000000010001, 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
```

Anexo C

Rutinas de biblioteca C normalizadas

(Este Anexo forma parte integrante de la presente Recomendación.)

C.1 Introducción

En este anexo se detalla un conjunto de rutinas de biblioteca C99 normalizadas [b-ISO/CEI 9899], que podrá utilizar el **cliente ECI**. Para cada función se definen los detalles de los operandos pasados por el **cliente ECI** y el valor de retorno.

Téngase en cuenta que por "cadena" se entiende una secuencia de bytes distintos de cero terminada por un byte cero.

Las funciones detalladas a continuación se muestran como llamadas a la biblioteca C normalizadas. En todos los casos, el primer parámetro irá en R2 (pues R1 contendrá el ID de la función, clibfunc). La declaración asumirá que todos los valores se pasan como valores escalares o punteros a valores no escalares. Si una función biblioteca llama para que un parámetro no escalar se pase por un valor, a continuación la SYSCALL los pasará por referencia y la VM deberá convertir el parámetro como exija la biblioteca.

Los valores de retorno siempre son valores escalares o punteros devueltos en R1.

NOTA – El valor seleccionado para clibfunc se forma de la siguiente manera:

((clibfunc >> 8) & 0x000000FF) = El número de subcapítulo del capítulo normalizado C que trata de las funciones de biblioteca, codificado como un decimal en código binario. (Para C99, el capítulo es 7 y la biblioteca <string.h> está en el subcapítulo 21.)

((clibfunc >> 4) & 0x0000000F) = El tipo de función en la biblioteca – el número siguiente al número de subcapítulo.

(clibfunc & 0x0000000F) = El número de función de un tipo concreto en la biblioteca – el número siguiente al número de tipo de función.

Por ejemplo, memmove() se describe en el encabezamiento 7.21.2.2 en [b-ISO/CEI 9899]. Por consiguiente, clibfunc se codifica como 0x00002122.

C.2 memmove

clibfunc: 0x00002122

Descripción: Copia n bytes de la memoria a que apunta s2 en la memoria a que apunta s1. Las memorias pueden solaparse.

Declaración uint8 * memmove(uint8 * s1, uint8 * s2, uint32 n)

Devuelve: s1

C.3 strcpy

clibfunc: 0x00002123

Descripción Copia la cadena (incluido el carácter de terminación) a que apunta s2 en la memoria a que apunta s1. No se definen los resultados si las zonas memoria se solapan.

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

Devuelve: s1

C.4 strncpy

clibfunc: 0x00002124

Descripción: Igual que strcpy(), pero, como máximo, se copian n bytes. Si la longitud de s2 es superior a n, se añadirá un byte nulo (a s1[n]).

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

Devuelve: s1

C.5 strcat

clibfunc: 0x00002131

Descripción: Añade una copia de la cadena a que apunta s2 (incluido el carácter de terminación) al final de la cadena a que apunta s1. No se definen los resultados si las zonas memoria se solapan.

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

Devuelve: s1

C.6 strncat

clibfunc: 0x00002132

Descripción: Añade una copia de la cadena a que apunta s2 (incluido el carácter de terminación) al final de la cadena a que apunta s1, pero, como máximo, se copian n bytes. Si la longitud de s2 es superior a n, se añadirá un byte nulo (a n+1 bytes tras el último byte no nulo del s1 original). No se definen los resultados si las zonas memoria se solapan.

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

Devuelve: s1

C.7 memcmp

clibfunc: 0x00002141

Descripción: Compara los primeros n bytes a que apunta s1 con los primeros n bytes a que apunta s2.

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

Devuelve: R1==0 si todos coinciden. En caso contrario, R1 depende de la primera posición por la izquierda en que los valores no coinciden.

R1>0 si el byte de s1 en esa posición es superior al byte de s2.

R1<0 si el byte de s1 en esa posición es superior al byte de s2.

C.8 strcmp

clibfunc: 0x00002142

Descripción: Compara las cadenas a que apuntan s1 y s2.

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

Devuelve: R1==0 si coinciden. En caso contrario, R1 depende de la primera posición por la izquierda en que los valores no coinciden.

R1>0 si el byte de s1 en esa posición es superior al byte de s2.

R1<0 si el byte de s1 en esa posición es superior al byte de s2.

C.9 strncmp

clibfunc: 0x00002144

Descripción: Compara las cadenas a que apuntan s1 y s2, pero sólo hasta n bytes.

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

Devuelve: R1==0 si coinciden. En caso contrario, R1 depende de la primera posición por la izquierda en que los valores no coinciden.

R1>0 si el byte de s1 en esa posición es superior al byte de s2.

R1<0 si el byte de s1 en esa posición es superior al byte de s2.

C.10 memchr

clibfunc: 0x00002151

Descripción: Encuentra la primera aparición del byte en c dentro de los n bytes a que apunta s.

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

Devuelve: Un puntero al byte localizado o 0, si no se encuentra ningún byte.

C.11 strchr

clibfunc: 0x00002152

Descripción: Encuentra la primera aparición del byte en c dentro de la cadena a que apunta s, hasta el byte (nulo) de terminación inclusive.

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

Devuelve: Un puntero al byte localizado o 0, si no se encuentra ningún byte.

C.12 strcspn

clibfunc: 0x00002153

Descripción: Calcula la longitud del segmento inicial máximo de la cadena a que apunta s1, formada enteramente por bytes que no pertenecen a la cadena a que apunta s2.

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

Devuelve: La longitud calculada.

C.13 strpbrk

clibfunc: 0x00002154

Descripción: Encuentra la primera aparición en la cadena a que apunta s1 de cualquier byte de la cadena a que apunta s2.

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

Devuelve: La localización del primer byte que cumple la condición o 0, si no se encuentran bytes de ese tipo.

C.14 strrchr

clibfunc: 0x00002155

Descripción: Encuentra la última aparición del byte en c dentro de la cadena a que apunta s, hasta el byte (nulo) de terminación inclusive.

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

Devuelve: Un puntero al byte localizado o 0, sin no se encuentra ningún byte.

C.15 strspn

clibfunc: 0x00002156

Descripción: Calcula la longitud del segmento inicial máximo de la cadena a que apunta s1, compuesta enteramente por bytes que pertenecen a la cadena a que apunta s2.

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

Devuelve: El valor calculado.

C.16 strstr

clibfunc: 0x00002157

Descripción: Encuentra la primera aparición de la cadena a que apunta s1 (byte de terminación exclusiva) en la cadena a que apunta s2.

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

Devuelve: Un puntero a la localización o 0, si no se encuentra.

C.17 memset

clibfunc: 0x00002161

Descripción: Copia n veces el byte menos significativo de c en la memoria a que apunta s.

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

Devuelve: s

Anexo D

Formato del fichero cliente ECI

(Este Anexo forma parte integrante de la presente Recomendación.)

El fichero de imagen **cliente ECI** se ajustará a la especificación del formato de fichero de objeto ELF [ETSI GS ECI 001-4]. En este Anexo se describe la información específica necesaria para ajustarse a la especificación de la VM. Dado que la VM soporta una arquitectura de 32 bits y un ordenamiento creciente, la identificación del fichero ELF en *e_ident* [ETSI GS ECI 001-4] utilizará los valores del Cuadro D.1.

Cuadro D.1 – Configuración de *e_ident* conforme con ECI

Nombre	Valor
e_ident[EI_CLASS]	ELFCLASS32
e_ident[EI_DATA]	ELFDATA2LSB

En el Cuadro D.2 se enumeran los valores que se utilizarán para algunos encabezamientos ELF.

Cuadro D.2 – Configuración ECI para los encabezamientos ELF

Nombre	Valor
e_type	ET_EXEC
e_machine	ET_NONE
e_version	EV_CURRENT

El cargador rechazará todo fichero de imagen **cliente ECI** cuyos valores difieran de los presentados en este Anexo.

Apéndice I

Aspectos que se han de mejorar

(Este Apéndice no forma parte integrante de la presente Recomendación.)

Se ha llegado a la conclusión de que es necesario mejorar y validar la presente Recomendación para que cumpla los requisitos estipulados en [UIT-T J.1010] y que [UIT-T J.1010] debe actualizarse para que incorporar los requisitos de la especificación de protección mejorada del contenido (ECP) de MovieLabs [b-ECP]. Las Recomendaciones [b-UIT-T J.1011], [UIT-T J.1012], UIT-T J.1013, [b-UIT-T J.1014], [b-UIT-T J.1015] y [b-UIT-T J.1015.1] deben actualizarse en el futuro para incorporar esas modificaciones a [UIT-T J.1010].

Varios Estados Miembros de la UIT y otras partes interesadas de diversas industrias – incluidos fabricantes de dispositivos y componentes electrónicos, los propietarios y titulares de contenido protegido por derecho de autor, proveedores de servicios de televisión por satélite (OTT) y de televisión por cable, y los proveedores de soluciones de sistemas de acceso condicional (CAS) y de gestión de derechos digitales (DRM) – de todo el mundo han expresado su preocupación por el hecho de que la interfaz común integrada (ECI) no satisface plenamente los requisitos de la ECP ni los requisitos más amplios de protección del contenido de la industria.

Más concretamente, sus preocupaciones se plantearon en las contribuciones a la reunión de la Comisión de Estudio 9 (CE 9) del UIT-T (16 a 23 de abril de 2020). En las contribuciones de Israel, Australia, el Miembro de Sector del UIT-T, Samsung, y los Asociados de la CE 9, Sky Group y MovieLabs, se propuso la introducción de diversas modificaciones en las Recomendaciones sobre ECI, pero no se llegó a un acuerdo al respecto. Estas modificaciones se consignaron en [b-SG9 Report 17 Ann.1].

Las propuestas tienen por objeto:

1) Simplificar el sistema ECI reduciendo su alcance.
2) Eliminar el DRM.
3) Eliminar la recodificación del contenido.
4) Eliminar la gestión de software.
5) Añadir API para el almacenamiento seguro y las operaciones criptográficas.
6) Permitir escalas de claves específicas para cada proveedor.
7) Utilizar los requisitos de la norma UIT-T J.1207 TEE.
8) Incluir la implementación de la TEE para la VM.
9) Mejorar la robustez de los algoritmos criptográficos, por ejemplo, utilizando SHA-384.
10) Utilizar certificados normalizados, como los de la Recomendación UIT-T X.509.
11) Reconsiderar las comunicaciones entre clientes.
12) Realizar declaraciones de coordinación adicionales con el ETSI.
13) Realizar revisiones adicionales por pares.
14) Analizar alternativas al modelo de autoridad fiduciaria.
15) Definir más detalladamente los aspectos técnicos de las normas de cumplimiento y robustez de ECI.
16) Añadir requisitos de diversidad, por ejemplo, la aleatorización del espacio de direcciones.
17) Añadir requisitos para verificar la integridad en tiempo de ejecución.

Estas propuestas responden a que la protección de contenidos y las amenazas de su compromiso evolucionan continuamente. La ECI fue concebida originalmente casi una década antes de que se aprobara de esta Recomendación UIT-T. Los sistemas como la ECI han de evaluarse regularmente respecto del estado actual de la técnica tanto en lo relativo a las técnicas analíticas como a los requisitos de protección de la industria.

Existen otros mecanismos que permiten la interoperabilidad. En particular, en el caso de la utilización de DRM, la mayoría de los servicios de vídeo por Internet han desplegado otras soluciones que ofrecen interoperabilidad y satisfacen sus necesidades.

Es importante que haya una mayor claridad, por cuanto muchos Estados Miembros consideran que las normas de la UIT tienen gran influencia en el desarrollo de sus mercados e industrias. La lista de preocupaciones garantiza la implementación de ECI en sus mercados nacionales, lo que puede requerir comprender plenamente las repercusiones de la presente Recomendación UIT-T y velar por que se tomen en consideración dichas cuestiones cuando se examine la legislación o la reglamentación o cuando las necesidades del mercado exijan que los aparatos de televisión digital de consumo sean interoperables. También garantiza que los fabricantes de equipo tecnológico, que pueden preferir utilizar un conjunto único de requisitos u otras normas a la hora de diseñar los productos, puedan tener en cuenta estas cuestiones al desarrollar productos para diferentes mercados.

Bibliografía

[b-UIT-T J.1010] Recomendación UIT-T J.1010 (2016), *Interfaz común insertada para soluciones CA/DRM intercambiables: Casos y requisitos de utilización*.

[b-UIT-T J.1011] Recomendación UIT-T J.1011 (2016), *Interfaz común insertada para soluciones CA/DR intercambiables: Arquitectura, definiciones y visión general*.

[b-UIT-T J.1014] Recomendación UIT-T J.1014 (2020), *Embedded common interface* for *exchangeable CA/DRM solutions; Advanced security – ECI-specific functionalities*.

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

[b-UIT-T J.1015.1] Recomendación UIT-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-UIT-T J-Suppl.7] Recomendaciones UIT-T de la Serie J – Suplemento 7 (2020), *Embedded common interface (ECI) for exchangeable CA/DRM solutions; Guidelines for the implementation of ECI (EG)*.

[b-SG9 Report 17 Ann.1] Informe de la reunión de la CE 9 del UIT-T, SG9-R17-Annex 1 (2020), Anexo 1 al Informe 17 de la reunión virtual de la CE 9 celebrada del 16 al 23 de abril de 2020.
https://www.itu.int/md/T17-SG09-R-0017/es

[b-ECP] MovieLabs Specification for Enhanced Content Protection – Version 1.2
Disponible en: <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/CEI 9899] ISO/CEI 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*. Disponible en: <https://refspecs.linuxfoundation.org/elf/elf.pdf>.

SERIES DE RECOMENDACIONES DEL UIT-T

Serie A	Organización del trabajo del UIT-T
Serie D	Principios de tarificación y contabilidad y cuestiones económicas y políticas de las telecomunicaciones/TIC internacionales
Serie E	Explotación general de la red, servicio telefónico, explotación del servicio y factores humanos
Serie F	Servicios de telecomunicación no telefónicos
Serie G	Sistemas y medios de transmisión, sistemas y redes digitales
Serie H	Sistemas audiovisuales y multimedia
Serie I	Red digital de servicios integrados
Serie J	**Redes de cable y transmisión de programas radiofónicos y televisivos, y de otras señales multimedia**
Serie K	Protección contra las interferencias
Serie L	Medio ambiente y TIC, cambio climático, ciberdesechos, eficiencia energética, construcción, instalación y protección de los cables y demás elementos de planta exterior
Serie M	Gestión de las telecomunicaciones, incluida la RGT y el mantenimiento de redes
Serie N	Mantenimiento: circuitos internacionales para transmisiones radiofónicas y de televisión
Serie O	Especificaciones de los aparatos de medida
Serie P	Calidad de la transmisión telefónica, instalaciones telefónicas y redes de líneas locales
Serie Q	Conmutación y señalización, y mediciones y pruebas asociadas
Serie R	Transmisión telegráfica
Serie S	Equipos terminales para servicios de telegrafía
Serie T	Terminales para servicios de telemática
Serie U	Conmutación telegráfica
Serie V	Comunicación de datos por la red telefónica
Serie X	Redes de datos, comunicaciones de sistemas abiertos y seguridad
Serie Y	Infraestructura mundial de la información, aspectos del protocolo Internet, redes de próxima generación, Internet de las cosas y ciudades inteligentes
Serie Z	Lenguajes y aspectos generales de soporte lógico para sistemas de telecomunicación

Impreso en Suiza
Ginebra, 2020