Union internationale des télécommunications

UIT-T J.1013

(04/2020)

SECTEUR DE LA NORMALISATION
DES TÉLÉCOMMUNICATIONS
DE L'UIT

SÉRIE J: RÉSEAUX CÂBLÉS ET TRANSMISSION DES SIGNAUX RADIOPHONIQUES, TÉLÉVISUELS ET AUTRES SIGNAUX MULTIMÉDIAS

Accès conditionnel et protection – Solutions d'accès conditionnel et de gestion des droits numériques intégrées interchangeables

Interface commune intégrée pour les solutions CA/DRM interchangeables; machine virtuelle

Recommandation UIT-T J.1013

Recommandation UIT-T J.1013

Interface commune intégrée pour les solutions CA/DRM interchangeables; machine virtuelle

Résumé

La Recommandation UIT-T J.1013, qui fait partie d'une publication en plusieurs parties sur l'interface commune intégrée pour les solutions de type accès conditionnel/gestion des droits numériques (CA/DRM) interchangeables, porte sur la machine virtuelle pour les solutions CA/DRM.

Cette Recommandation UIT-T, qui est une transposition de la norme ETSI GS ECI 001-4, est le fruit d'une collaboration entre la CE 9 de l'UIT-T et l'ETSI ISG ECI. Une modification mineure a été apportée au paragraphe 7.3.7.1.

Historique

Edition	Recommandation	Approbation	Commission d'études	ID unique*
1.0	UIT-T J.1013	23-04-2020	9	11.1002/1000/13574

Mots clés

CA, DRM, échange

* Pour accéder à la Recommandation, reporter cet URL http://handle.itu.int/ dans votre navigateur Web, suivi de l'identifiant unique, par exemple http://handle.itu.int/11.1002/1000/11830-en.

AVANT-PROPOS

L'Union internationale des télécommunications (UIT) est une institution spécialisée des Nations Unies dans le domaine des télécommunications et des technologies de l'information et de la communication (ICT). Le Secteur de la normalisation des télécommunications (UIT-T) est un organe permanent de l'UIT. Il est chargé de l'étude des questions techniques, d'exploitation et de tarification, et émet à ce sujet des Recommandations en vue de la normalisation des télécommunications à l'échelle mondiale.

L'Assemblée mondiale de normalisation des télécommunications (AMNT), qui se réunit tous les quatre ans, détermine les thèmes d'étude à traiter par les Commissions d'études de l'UIT-T, lesquelles élaborent en retour des Recommandations sur ces thèmes.

L'approbation des Recommandations par les Membres de l'UIT-T s'effectue selon la procédure définie dans la Résolution 1 de l'AMNT.

Dans certains secteurs des technologies de l'information qui correspondent à la sphère de compétence de l'UIT-T, les normes nécessaires se préparent en collaboration avec l'ISO et la CEI.

NOTE

Dans la présente Recommandation, l'expression "Administration" est utilisée pour désigner de façon abrégée aussi bien une administration de télécommunications qu'une exploitation reconnue.

Le respect de cette Recommandation se fait à titre volontaire. Cependant, il se peut que la Recommandation contienne certaines dispositions obligatoires (pour assurer, par exemple, l'interopérabilité et l'applicabilité) et on considère que la Recommandation est respectée lorsque toutes ces dispositions sont observées. Le futur d'obligation et les autres moyens d'expression de l'obligation comme le verbe "devoir" ainsi que leurs formes négatives servent à énoncer des prescriptions. L'utilisation de ces formes ne signifie pas qu'il est obligatoire de respecter la Recommandation.

DROITS DE PROPRIÉTÉ INTELLECTUELLE

L'UIT attire l'attention sur la possibilité que l'application ou la mise en œuvre de la présente Recommandation puisse donner lieu à l'utilisation d'un droit de propriété intellectuelle. L'UIT ne prend pas position en ce qui concerne l'existence, la validité ou l'applicabilité des droits de propriété intellectuelle, qu'ils soient revendiqués par un membre de l'UIT ou par une tierce partie étrangère à la procédure d'élaboration des Recommandations.

A la date d'approbation de la présente Recommandation, l'UIT avait été avisée de l'existence d'une propriété intellectuelle protégée par des brevets à acquérir pour mettre en œuvre la présente Recommandation. Toutefois, comme il ne s'agit peut-être pas de renseignements les plus récents, il est vivement recommandé aux développeurs de consulter la base de données des brevets du TSB sous http://www.itu.int/ITU-T/ipr/.

TABLE DES MATIÈRES

Introduction

La présente Recommandation UIT-T[1], qui est une transposition de la norme [b-ETSI GS ECI 001-4] de l'ETSI, est le fruit d'une collaboration entre la CE 9 de l'UIT-T et l'ETSI ISG ECI. Une modification mineure a été apportée au paragraphe 7.3.7.1.

L'objectif de la présente Recommandation est de faciliter l'interopérabilité et la concurrence en matière de services de communications électroniques et, en particulier, sur le marché de la radiodiffusion et des appareils audiovisuels. Toutefois, d'autres technologies sont disponibles et peuvent également être appropriées et efficaces en fonction de la situation dans les États Membres.

La présente Recommandation contient la description d'une machine virtuelle (VM) fonctionnant sur un ordinateur isolé ("bac à sable") et capable d'exécuter un ensemble d'instructions et de fonctions d'appel au système. Cette machine virtuelle est conçue pour fonctionner sous différents environnements et interagit avec d'autres applications se trouvant sur le même ordinateur par le biais d'interfaces bien définies. Elle prend en charge de différentes manières l'exécution de son propre jeu d'instructions, et elle dispose d'un mécanisme modulaire pour exécuter les éléments écrits dans le **code natif**[2] de l'unité centrale de traitement (CPU) de l'**Hôte de l'interface commune intégrée (ECI)**. Enfin, elle interagit avec le matériel et d'autres éléments de l'environnement de l'**Hôte ECI**. La machine virtuelle est ainsi capable d'exécuter rapidement un code renouvelable; elle offre donc un large éventail d'applications potentielles sécurisées, notamment pour créer les clients d'un système d'accès conditionnel et de gestion des droits numériques (CA/DRM).

---

1 Plusieurs domaines nécessitant des développements supplémentaires ont été identifiés dans l'Appendice I.

2 Dans le texte de la présente Recommandation, on utilise des caractères gras pour les termes dont la définition est propre au contexte de l'interface commune intégrée et peut différer de l'usage courant.

Recommandation UIT-T J.1013

Interface commune intégrée pour les solutions CA/DRM interchangeables; machine virtuelle

1 Domaine d'application

La présente Recommandation contient la description d'une machine virtuelle destinée à être intégrée dans des récepteurs de télévision numérique et des décodeurs. Cette machine virtuelle offre un environnement sécurisé pour exécuter un noyau d'accès conditionnel ou les applications client d'un système de gestion des droits numériques. Le but est de proposer à ces clients un environnement d'exécution uniforme, dès lors que les prescriptions minimales de performance de l'**Hôte ECI** sont respectées, qu'une API courante est disponible pour extraire les données de sécurité essentielles à partir des contenus (c'est-à-dire que ces données sont encapsulées dans les contenus) ou de réseaux extérieurs (par exemple l'Internet) et qu'il est possible d'accéder aux ressources de manière normalisée via l'environnement de l'**Hôte ECI**. Voir également les Recommandations [b-UIT-T J.1010] et [b-UIT-T J.1011].

La présence et l'emploi de la machine virtuelle permettent d'échanger des clients CA/DRM à volonté et de prendre en charge plusieurs instances simultanées de ces clients dans des **Hôtes ECI**. Cela permet de garantir que les utilisateurs et les opérateurs ne sont pas liés à un fournisseur particulier de protection des contenus (CP) et de faciliter l'utilisation de différents types de solutions de sécurité pour différents types de contenus. Pour les fournisseurs de systèmes de protection de contenus, cette machine virtuelle constitue un environnement d'exécution bien connu qui ne nécessite pas de travaux d'intégration particuliers avec les différents fabricants et les différents dispositifs d'**Hôtes ECI**.

2 Références

La présente Recommandation se réfère à certaines dispositions de Recommandations UIT-T et autres textes qui, de ce fait, en sont partie intégrante. Les versions indiquées étaient en vigueur au moment de la publication de la présente Recommandation. Toute Recommandation ou tout texte étant sujet à révision, les utilisateurs de la présente Recommandation sont invités à se reporter, si possible, aux versions les plus récentes des références normatives suivantes. La liste des Recommandations de l'UIT-T en vigueur est régulièrement publiée. La référence à un document figurant dans la présente Recommandation ne donne pas à ce document en tant que tel le statut de Recommandation.

[UIT-T J.1012] Recommandation UIT-T J.1012 (2020), *Interface commune intégrée pour les solutions CA/DRM interchangeables; conteneur, chargeur, interfaces et révocation pour solutions CA/DRM*.

[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 Définitions

3.1 Termes définis ailleurs

Aucun.

3.2 Termes définis dans la présente Recommandation

Les termes suivants sont définis dans la présente Recommandation:

3.2.1 bytecode: code du **Client ECI** (comportant généralement un noyau d'accès conditionnel ou l'application client d'un système de gestion des droits numériques) exécuté par la machine virtuelle.

3.2.2 équipement de locaux d'abonné (CPE): dispositif présent chez le client et assurant des fonctions de chiffrement et déchiffrement spécifiées dans l'interface commune intégrée (**ECI**).

3.2.3 interface commune intégrée (ECI): architecture et système définis par le groupe de spécification industrielle "Embedded CI" (interface commune intégrée) de l'ETSI. Ils permettent de développer les programmes de **Clients ECI** interchangeables et de les exécuter dans des équipements de locaux d'abonnés (**CPE**), et assurent ainsi l'interopérabilité des dispositifs **CPE** en ce qui concerne l'**Interface ECI**.

3.2.4 client ECI (client d'une interface commune intégrée): mise en oeuvre d'un client d'accès conditionnel et de gestion des droits numériques (CA/DRM) conforme aux spécifications de l'interface commune intégrée (**ECI**).

3.2.5 hôte ECI: système matériel et logiciel d'un équipement **CPE** qui couvre les fonctionnalités liées à l'**Interface ECI** et comporte des interfaces vers un **Client ECI**.

3.2.6 code natif: code de programmation écrit au moyen du jeu d'instructions exécutables originelles dont dispose le processeur de l'**Hôte ECI**.

3.2.7 instance de machine virtuelle: exemplaire d'une machine virtuelle qui est hébergé par un **Hôte ECI** et qui se présente, pour un **Client ECI**, comme un environnement d'exécution dans lequel le client peut fonctionner.

4 Abréviations et acronymes

La présente Recommandation utilisent les abréviations et acronymes suivants:

API interface de programmation d'application (*application programming interface*)

CA accès conditionnel (*conditional access*)

CAS système d'accès conditionnel (*conditional access system*)

CI interface commune (*common interface*)

CP protection du contenu (*content protection*)

CPE équipement de locaux d'abonné (*customer premises equipment*)

CPU unité centrale de traitement (*central processing unit*)

DRM gestion des droits numériques (*digital rights management*)

ECI interface commune intégrée (*embedded common interface*)

ECP protection de contenu améliorée (*enhanced content protection*)

ELF format exécutable et reliable (*executable and linkable format*)

EPG guide électronique des programmes (*electronic programme guide*)

ID identification/identité/identifiant (*identification/identity/identifier*)

OS système d'exploitation (*operating system*)

OTT Over The Top

PC compteur ordinal (*program counter*)

POSIX interface pour la portabilité des systèmes d'exploitation (*portable operating system interface*)

RISC ordinateur à jeu d'instructions réduit (*reduced instruction set computer*)

VM machine virtuelle (*virtual machine*)

5 Conventions

Dans la présente Recommandation, l'emploi de termes en caractères gras et commençant par une majuscule indique que ces termes s'entendent au sens particulier des interfaces ECI, qui peut être différent de leur emploi courant.

6 Principes conceptuels

6.1 La machine virtuelle en tant qu'unité centrale de traitement

Fondamentalement, une machine virtuelle se compose d'une unité centrale de traitement (CPU) virtuelle dotée de sa propre mémoire pour le code et les données, et d'un jeu d'interfaces système donnant accès à des éléments matériels de l'ordinateur qui héberge l'**Hôte ECI**. L'unité centrale émulée exécute le code à la manière d'une CPU virtuelle de 32 bits; le code qu'elle exécute est appelé **Bytecode** dans la présente Recommandation. Comme la machine virtuelle est une simulation d'un processeur RISC (ordinateur à jeu d'instructions réduit) polyvalent, elle est capable d'exécuter des applications très diverses.

6.2 Caractéristiques de la machine virtuelle

La machine virtuelle offre un environnement d'exécution pour un seul processus et un seul fil d'exécution.

L'interface avec le matériel et d'autres fonctions de l'**Hôte ECI** est une bibliothèque courante d'appels au système (SYSCALL). L'instruction SYSCALL fait partie des instructions propres à la machine virtuelle; on l'exécute généralement après avoir défini les paramètres de la routine de la bibliothèque (c'est-à-dire que l'instruction est passée dans des "registres" de la machine virtuelle).

Toutes les interactions entre le **Client ECI** et l'**Hôte ECI** s'effectuent par le biais de cette interface. Aucune architecture d'interruption n'a été prévue: une fois lancé, le **Client ECI** s'exécute jusqu'à l'achèvement de la tâche. Il est donc impossible de passer directement des appels à la machine virtuelle. Si cette méthode limite quelque peu la souplesse du système, elle offre un meilleur contrôle sur l'exécution de la machine virtuelle (ce qui garantit la robustesse de son fonctionnement) et elle évite ainsi les situations de concurrence, les collisions avec des opérations à durée critique, etc.

Dès lors, le seul moyen de passer des données ou des messages au **Client ECI** qui s'exécute dans une machine virtuelle consiste à faire en sorte que ce soit le **Client ECI** lui-même qui les demande par le biais des appels SYSCALL pertinents.

6.3 Isolement de Clients ECI particuliers

Le **Client ECI** s'exécute dans une machine virtuelle, qui est elle-même une application s'exécutant dans le micrologiciel de l'**Hôte ECI**. Il est possible de créer plusieurs instances de la machine virtuelle, chacune pouvant s'exécuter sur un **Client ECI** différent. Il en découle trois contraintes fondamentales pour l'environnement d'exécution de l'**Hôte ECI**:

1) Isolement de Clients ECI particuliers – Le système d'exploitation (OS) doit attribuer suffisamment de ressources à chaque **Instance de machine virtuelle** pour que toutes les instances s'exécutant simultanément puissent respecter les prescriptions de fonctionnement;

des valeurs pour les prescriptions de fonctionnement sont proposées dans le document [b-UIT-T J Suppl. 7].

2) Les bibliothèques définies au point 8 et dans l'Annexe C doivent être entièrement réentrantes ou être disponibles pour chaque instance de la machine virtuelle de manière indépendante.

3) Le système d'exploitation et la machine virtuelle doivent garantir qu'aucune information ne puisse être échangée entre les **Clients ECI** en cours d'exécution et le monde extérieur, y compris d'autres **Clients ECI**, autrement que par les voies explicitement prévues à cette fin dans l'interface SYSCALL. Cette contrainte signifie notamment que toute la mémoire réservée pour l'espace mémoire des données d'une **Instance de machine virtuelle** doit être vidée de son précédent contenu avant usage, et qu'il faut bloquer toute tentative d'utiliser la machine virtuelle dans des conditions exceptionnelles dans le but de déclencher un comportement non spécifié. Elle signifie aussi qu'il est impossible, pour un **Client ECI**, de modifier son **Bytecode**. Enfin, il découle implicitement de cette contrainte que l'**Hôte ECI** et la machine virtuelle doivent effectuer toutes les vérifications nécessaires pour empêcher un **Client ECI** de déclencher un comportement imprévu de l'**Hôte ECI** ou de l'instance de la machine virtuelle susceptible par exemple de donner volontairement ou non au **Client ECI** la possibilité de manipuler (pirater) l'**Hôte ECI**.

6.4 Spécifications de la machine virtuelle

Nous allons décrire en détail ci-après les éléments suivants de la machine virtuelle:

1) l'architecture technique;

2) le jeu d'instructions; et

3) l'interface de l'**Hôte ECI**.

6.5 Chargeur du Client ECI

Pour pouvoir exécuter le **Client ECI**, le **Bytecode** doit au préalable être chargé dans l'espace de la mémoire de la machine virtuelle attribué au code, tandis que l'espace attribué aux données doit être initialisé. Nous examinerons au point 9 quelques aspects particuliers du format du conteneur du **Client ECI** et de l'initialisation de la machine virtuelle.

7 La machine virtuelle

7.1 Environnement d'exécution

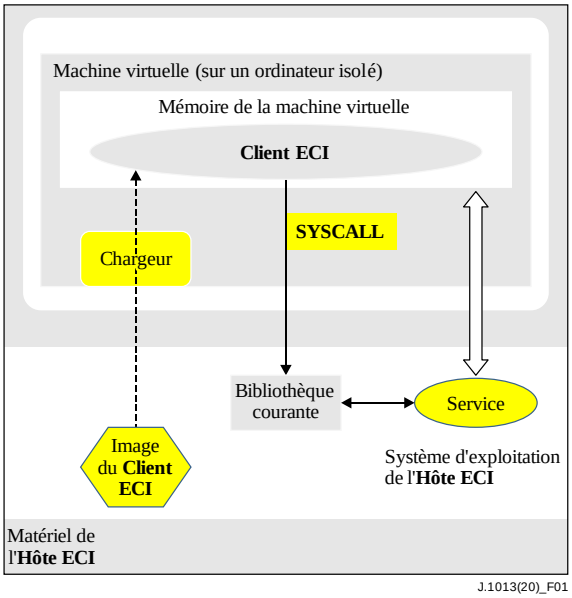

Figure 1 – Environnement de l'hôte de la machine virtuelle

Comme l'illustre la Figure 1, la machine virtuelle est exécutée sur un ordinateur isolé ("bac à sable"), ce qui permet de la séparer du système d'exploitation de l'**Hôte ECI**, d'autres instances de la machine virtuelle et de toute autre application fonctionnant sur l'**Hôte ECI**.

Elle se compose d'une application native de l'**Hôte ECI**, avec la mémoire qui lui est associée, ainsi que d'une bibliothèque d'interfaces et d'un chargeur permettant d'installer le **Bytecode** qui va créer un **Client ECI**. La bibliothèque d'interfaces permet au **Client ECI** d'accéder à des fonctions du système d'exploitation et du matériel de l'**Hôte ECI**, ainsi qu'à d'autres applications susceptibles de s'exécuter en même temps sur l'**Hôte ECI** et avec lesquelles le **Client ECI** pourrait avoir besoin d'interagir. À titre d'exemple classique, on peut citer l'interaction avec un guide électronique des programmes (EPG) ayant besoin d'une autorisation pour pouvoir afficher un contenu particulier à l'intention d'un utilisateur.

7.2 Architecture de la machine virtuelle

7.2.1 Architecture de l'unité centrale de traitement

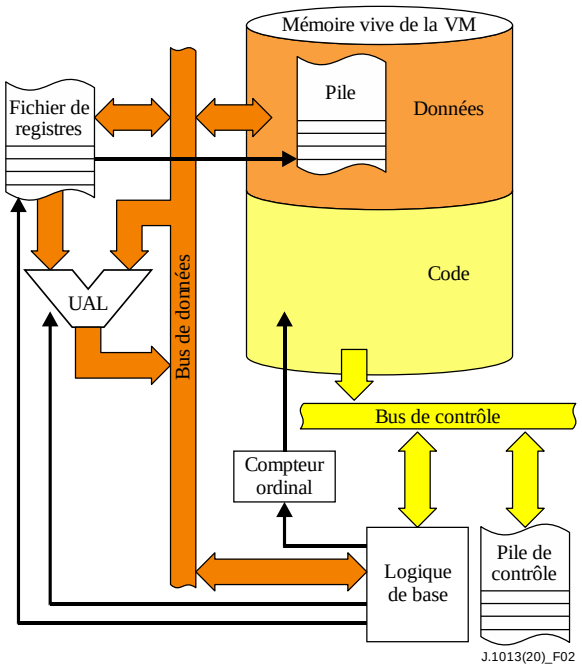

Figure 2 – Architecture du processeur virtuel

La Figure 2 montre l'architecture de l'unité centrale de la machine virtuelle. La machine virtuelle est un système fondé sur des registres qui se compose des éléments suivants:

– Un fichier de registres contenant des registres polyvalents de 32 bits. Ces registres sont organisés en fenêtres de registres. Chaque fenêtre contient 32 registres, dont les 16 derniers chevauchent les 16 premiers de la fenêtre suivante. Dans chaque fenêtre, deux de ces registres servent respectivement de pointeur de pile et de pointeur de cadre. Le nombre total de registres dans le fichier de registres est défini dans le paramètre REGISTER_FILE_SIZE, qui est défini dans l'Annexe A.

– Une architecture d'unité centrale de traitement de type Harvard. Les données sont stockées dans un espace mémoire plat de 32 bits. Le code est stocké dans un espace de mémoire en lecture seule non adressable.

– Une pile de contrôle indépendante conserve la liste des adresses retour. Son contenu est inaccessible au **Bytecode** et aux applications extérieures. Le nombre d'adresses retour que la pile peut stocker est défini dans le paramètre CONTROL_STACK_SIZE (voir l'Annexe A).

– Des instructions de chargement et de stockage pour les octets signés et non signés et les données de type demi-mot et mot, dont la taille est respectivement de 8, 16 et 32 bits.

– Un jeu d'instructions, dont beaucoup d'instructions de traitement de données sont spécialement adaptées au domaine d'application considéré.

– Des octets ordonnés directement dans l'unité centrale pour assurer un chargement et un stockage efficaces, indépendamment du boutisme. On emploie un alignement naturel (par taille) des types de base afin que le **Bytecode** ait une portabilité maximale. Ainsi, l'adresse mémoire d'un demi-mot est toujours paire et l'adresse d'un mot est toujours un multiple de quatre.

– Une instruction d'appel au système (SYSCALL) permettant de créer des services système. Elle permet aussi d'ajouter à la machine virtuelle des fonctions intégrées, par exemple pour effectuer des traitements de données fréquents directement depuis l'unité centrale.
– Une mémoire paginée intervenant lorsque l'espace mémoire est fragmenté. Elle permet d'attribuer une partie de la mémoire de l'unité centrale à l'espace mémoire de la machine virtuelle.

7.2.2 Registres

Chaque fenêtre de registres contient 32 registres numérotés de R0 à R31. Deux d'entre eux sont réservés car ils ont une fonction particulière: le R0 est le pointeur de cadre et le R16 est le pointeur de pile. L'emploi de ces registres est décrit plus en détail ci-dessous.

Lorsqu'une fonction est exécutée, l'instruction ENTER fait monter la fenêtre de registres de seize registres. L'ancien pointeur de pile devient alors le nouveau pointeur de cadre, tandis qu'un nouveau pointeur de pile est créé et que quinze registres supplémentaires deviennent disponibles. Le nouveau pointeur de pile est initialisé en soustrayant la taille du cadre indiquée dans l'instruction ENTER du pointeur de cadre.

L'instruction RETURN inverse ce processus. Elle fait descendre la fenêtre de seize registres, rétablissant ainsi l'ancien pointeur de cadre et l'ancien pointeur de pile.

Comme la routine appelée ne peut atteindre les registres R0 à R15, ceux-ci sont automatiquement sauvegardés par l'appelé. L'adresse de retour étant enregistrée dans une pile de contrôle indépendante, on n'emploie pas de pile de données pour enregistrer les registres sauvegardés par l'appelé et les adresses de retour.

Le nombre réel de registres étant limité, la profondeur des appels d'un **Client ECI** est plafonnée (CONTROL_STACK_SIZE). Tout dépassement de cette profondeur entraîne l'arrêt du programme de la machine virtuelle. Le nombre de registres et la profondeur correspondante de la pile de contrôle peuvent être définis au moment de la création de la machine virtuelle.

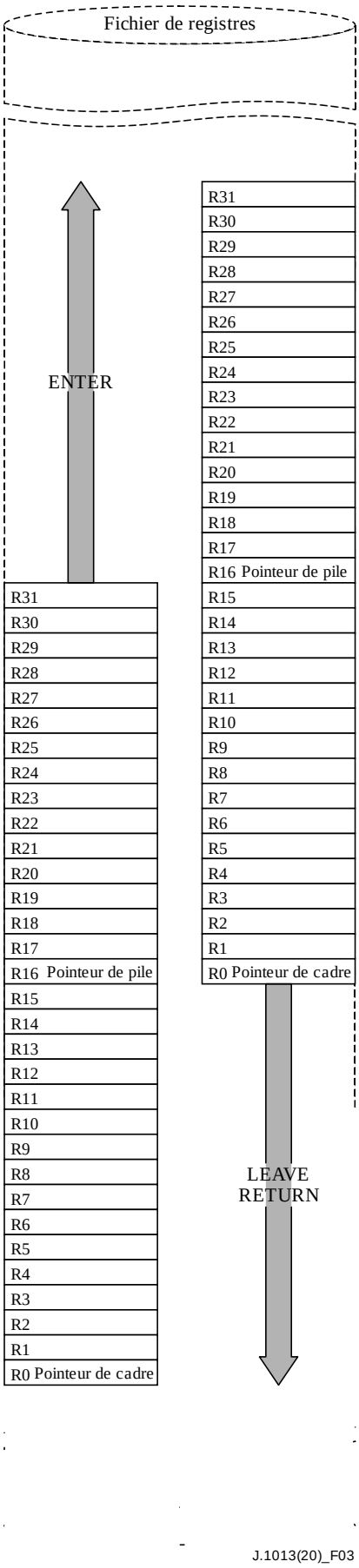

Figure 3 – Architecture du fichier de registres

7.2.3 Espace des données

L'adresse de base des données de la machine virtuelle, qui est définie dans le paramètre DATA_BASE_ADDRESS (voir l'Annexe A), est 0x1000000 (16 Moctets). La plus petite adresse non adressable au-dessus de la mémoire adressable prend la valeur DATA_BASE_ADDRESS + ADDRESSABLE_DATA_SIZE (voir l'Annexe A). L'adresse de base de la pile est définie par la mise en oeuvre de la machine virtuelle, mais elle doit se situer vers le haut de l'espace des adresses. L'espace maximum que la machine virtuelle peut réserver à des fins privées dans l'espace d'adressage du **Client ECI** "en-dessous" de la base de la pile (dont l'adresse est plus élevée) est défini dans le paramètre VM_RESEVED_SIZE (voir l'Annexe A). Au moment de l'initialisation de la machine virtuelle, le pointeur de pile pointe vers le premier emplacement libre dans la pile. Le **Client ECI** peut considérer qu'à ce moment, le sommet du tas (vide) est égal à la taille du segment de données initialisées + la taille du segment de données non initialisées, la somme étant arrondie vers le haut à un multiple de 4.

La structure de la mémoire de données est représentée dans la Figure 4:

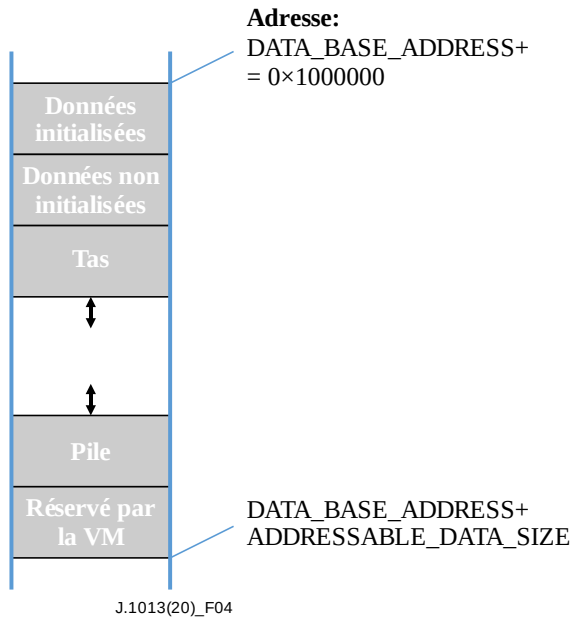

Figure 4 – Structure de la mémoire de données de la machine virtuelle

Au moment de l'initialisation du **Client ECI**, le chargeur de celui-ci envoie les segments de données initialisées et non initialisées à l'adresse DATA_BASE_ADDRESS. Tous les octets du segment de données non initialisées sont mis à zéro. Le segment de données initialisées n'est pas protégé en écriture.

NOTE 1 – Au début, la taille de la pile est limitée. Comme les structures de données locales définies dans les fonctions C sont généralement attribuées à la pile, le **Client ECI** devrait définir une taille du segment de pile suffisante au cas où de grandes variables locales seraient employées dans le code C.

NOTE 2 – L'**Hôte ECI** peut réserver des zones tampons pour les messages dans la mémoire réservée de la machine virtuelle en-dessous de l'adresse de base de la pile.

Les futures versions de la machine virtuelle réserveront peut-être davantage de mémoire adressable pour les **Clients ECI**; en d'autres termes, la valeur du paramètre ADDRESSABLE_DATA_SIZE sera peut-être plus élevée. Dans un souci de rétrocompatibilité, les **Clients ECI** ne doivent pas dépendre de la valeur ou de la gamme de valeurs particulières du pointeur de pile définies actuellement, mais simplement utiliser le pointeur de pile tel qu'il est passé au moment de l'initialisation.

Le chargeur du **Client ECI** ne doit pas charger de fichier image si celui-ci n'est pas conforme à la convention de structure de mémoire mentionnée plus haut à propos des segments de données initialisées et non initialisées.

7.2.4 Espace du code

Le programme ne peut pas accéder directement au code. Il peut obtenir des références opaques en 32 bits à des objets de code statiques (par exemple le point d'entrée d'une routine ou la cible d'un saut), appelées *références de code* (voir l'instruction MOVF). Ces *références de code* ne peuvent être employées que dans les instructions indirectes des flux de commande (JMPR et CALLR). Ce ne sont pas des pointeurs vers l'espace mémoire réservé au code, et elles ne permettent pas d'effectuer des opérations arithmétiques sur les pointeurs.

L'adresse de début du segment de code dans l'espace des adresses réservé au code est 0x00000000. La taille maximale du segment de code est définie dans le paramètre CODE_SIZE (voir l'Annexe A).

7.2.5 Pile

Par convention, la pile est située dans la mémoire des données; elle ne contient que des mots, ne peut croître que vers des adresses plus basses, et le registre R16 (*pointeur de pile*) de la fenêtre de registres en cours pointe toujours vers son sommet (le dernier mot poussé).

Le pointeur de cadre R0 sert à pointer vers la pile de l'appelé afin que les routines appelées puissent accéder aux paramètres ou à d'autres données poussées dans la pile (voir le § 7.2.8).

7.2.6 Boutisme

Les données nécessitant plusieurs octets (demi-mots et mots) sont représentées dans la mémoire du système au format little-endian. Le logiciel du **Client ECI** doit utiliser ce format.

7.2.7 Exceptions

L'unité centrale de traitement de la machine virtuelle ne produit aucune exception pendant l'exécution. Si une instruction s'exécute dans des conditions qui ne sont pas prévues dans la présente Recommandation (par exemple un accès non conforme à un demi-mot ou un mot en mémoire, un accès à une adresse mémoire n'ayant pas de mémoire correspondante, ou un branchement vers une référence de code inconnue), le comportement de la machine virtuelle est indéfini. La machine virtuelle peut choisir d'arrêter le noyau. Elle doit garantir qu'en aucune circonstance, un **Client ECI** fonctionnant en-dehors des conditions définies dans la présente Recommandation ne puisse accéder à des données non autorisées ou avoir un quelconque effet sur d'autres applications.

7.2.8 Convention d'appel

Par convention, les appels passent les sept premiers paramètres scalaires (pointeurs et entiers) aux registres R17 à R23. Pour l'appelé, ces appels apparaissent dans les registres R1 à R7.

Au-delà du septième paramètre scalaire, l'appelant passe les paramètres scalaires à la pile de droite à gauche. Compte tenu du mécanisme de fenêtres de registres, l'appelé verra toujours le registre R0 pointer vers le huitième paramètre (s'il existe). R0 est donc le *pointeur de cadre*. Les paramètres de structure sont toujours passés soit directement à la pile, soit par référence. Les pointeurs font toujours référence à l'espace de mémoire de la machine virtuelle.

NOTE – Les appels SYSCALL ne passent les structures et les tableaux que par référence. Cette méthode devrait également être employée pour d'autres appels.

L'appelé laisse la valeur de retour dans le registre R1, qui pour l'appelant est le registre R17. Les types de taille inférieure à 32 bits sont passés (et renvoyés) comme des valeurs de 32 bits.

La structure envoyée en retour est mise en œuvre en passant un premier paramètre implicite, qui est un pointeur vers la zone de mémoire dans laquelle le type de retour devrait être stocké (passage par

référence). L'appelé écrit sa réponse à l'emplacement indiqué par ce paramètre. Ce pointeur de retour est traité comme un argument normal (donc passé à R17 → R1), ce qui implique que les arguments ordinaires d'une fonction retournant une structure se déplacent vers d'autres registres réservés à la convention d'appel (R18 à R23 → R2 à R7) ou passent par la pile.

7.3 Jeu d'instructions de la machine virtuelle

7.3.1 Notation

On emploie la notation suivante:

```
rx       Registre x.
uimm5    5 bits non signée immédiate.
uimms9   9 bits non signée immédiat. Toujours un multiple de deux.
uimms10  10 bits non signée immédiat. Toujours un multiple de quatre.
simm11   11 bits signée immédiate.
simm16   16 bits signée immédiate.
uimm16   16 bits non signée immédiate.
pcr16    16 bits signée relative au compteur ordinal.
pcr24    24 bits signée relative au compteur ordinal.
imm32    32 bits immédiate.
low8(x)  Les 8 bits de x les moins significatifs.
low16(x) Les 16 bits de x les moins significatifs
```

Les descriptions fonctionnelles emploient la sémantique C sur des types d'entiers à 32 bits. La possibilité de prendre en charge des types de données signées ou non signées est indiquée en commentaire. Pour accéder à la mémoire, on emploie les fonctions MEM1(), MEM2() et MEM4(), en lisant respectivement 1, 2 ou 4 octets de mémoire. L'opérande de ces fonctions est un décalage dans le segment de données. Au besoin, les fonctions MEM s'accompagnent du préfixe U pour indiquer que les opérations sont non signées, ou S pour les opérations comportant une extension de signe.

7.3.2 Instructions arithmétiques

7.3.2.1 Opérandes des registres

```
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 Registre, opération immédiate

```
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;
```

Les opérations signées de division et de reste sont conformes à la définition C99: la division tronque le résultat mathématique vers zéro; le reste doit respecter la relation suivante:

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

où % représente la fonction de reste, ou modulo.

Le bon opérande des instructions de décalage doit se trouver dans l'intervalle [0, 31], faute de quoi le comportement sera indéfini. Le décalage à droite signé copie le bit original le plus élevé vers les positions libres. En termes arithmétiques, cela correspond à une division par une puissance de deux, le résultat mathématique étant arrondi vers moins l'infini (arrondissement *vers le bas*).

7.3.3 Formes courtes

Beaucoup d'occurrences des trois instructions d'opérande répondent aussi par l'un des opérandes. On peut alors coder les opérandes de manière plus compacte en utilisant les opcodes spéciaux suivants:

```
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
```

En termes de bits, les opérations immédiates testent ou modifient un seul bit. Elles peuvent être codées sur 5 bits en donnant la position des bits.

```
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);
```

Beaucoup de comparaisons sont effectuées par rapport à zéro. On économise ainsi l'emploi d'un opérande immédiat et l'émulation est moins coûteuse.

```
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;
```

Les versions non signées de ces comparaisons n'ont pas de sens particulier. Elles sont vraies ou fausses, ou elles peuvent être exprimées par les codes `EQZ` ou `NEZ`.

7.3.4 Flux de commande

7.3.4.1 Règles communes

Les instructions du flux de commande comportant des opérandes directs codent leurs cibles par rapport à l'adresse de fin de l'instruction. Dans les flux de commande utilisant un registre, celui-ci comporte un index des pointeurs de fonction.

7.3.4.2 Branchements inconditionnels et appels de fonction

```
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 Branchements conditionnels

```
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 Branchements conditionnels fondés sur des comparaisons de mémoire avec une constante

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

Ces branchements lisent un mot dans la mémoire et le comparent avec une constante.

7.3.4.5 Branchements conditionnels longs

Pour chacun des branchements conditionnels précités, il existe une version *longue*, dans laquelle le décalage est de 24 bits. L'assembleur devrait choisir la version la plus courte possible.

7.3.5 Instructions de chargement et de stockage

7.3.5.1 Registre + décalage

```
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 Registre + décalage court

```
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 Registre indexé

```
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 Index absolu

```
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;
```

À noter qu'il n'est pas nécessaire de charger des octets dont l'index est absolu. Ainsi, `LDSBAX` est équivalent à `LDSBI`.

7.3.5.5 Accès à une pile dédiée

Ces modes de chargement et de stockage de mots utilisent implicitement le pointeur de cadre.

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

7.3.5.6 Transfert de mémoire

Instruction de copie de bloc permettant de copier des blocs produits par le compilateur.

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

7.3.6 Instructions complexes

Ces instructions effectuent une combinaison d'opérations utilisant généralement des opérandes immédiats. Dans le présent résumé, chaque opérande désigné par les codes `i1`, `i2`, etc. est un opérande immédiat de 32 bit (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 Divers

7.3.7.1 Appels au système

Les appels au système permettent de mettre en oeuvre différents services.

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

Un jeu minimum d'appels au système de type POSIX (interface pour la portabilité des systèmes d'exploitation) est directement réservé au système d'exploitation sous-jacent. On peut ajouter d'autres services plus liés aux applications.

7.3.7.2 Pseudo-instructions

Certaines opérations peuvent être exprimées par le biais d'autres opérations. Les pseudo-opcodes suivants sont disponibles:

```
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 Interface entre le Client ECI et l'Hôte ECI

8.1 Principes généraux

Les appels au système interviennent quand une instruction SYSCALL est exécutée. Cette instruction contient un opérande immédiat qui reconnaît l'appel au système. Celui-ci est en réalité un appel à une bibliothèque courante qui passe les paramètres décrits au § 7.2.8.

Les sept premiers paramètres (mots ou pointeurs) sont passés dans les registres R1 à R8. S'ils sont de type 8 bits ou 16 bits scalaire, il faut procéder à une extension de signe pour atteindre une valeur de 32 bits. Les valeurs de retour (mots ou pointeurs) sont placées dans le registre R1.

Sauf indication contraire, toutes les adresses mémoire font référence à l'espace de mémoire de la machine virtuelle.

Pour des raisons de compatibilité future, le **Client ECI** doit remettre à zéro tous les registres R1 à R8 n'ayant pas servi à passer des paramètres. Le contenu de l'ensemble des registres peut être supprimé au moyen de la fonction correspondante de la bibliothèque.

Les appels au système obligatoires de la bibliothèque, que toute mise en oeuvre conforme doit pouvoir prendre en charge, sont énumérés ci-après. Le format employé contient les éléments suivants:

– L'identifiant du SYSCALL employé comme opérande immédiat (SYSCALL imm32).
– Une description de la fonction de la bibliothèque.
– Une déclaration en syntaxe C.
– Une description des paramètres et de la valeur de retour.
– Toute autre note supplémentaire.

Les paramètres et les valeurs de retour doivent être saisis en respectant la convention suivante:

– **uint*nn*** représente un entier non signé de *nn* bits (*nn* prenant la valeur 8, 16 ou 32). Si les valeurs sont inférieures à 32 bits, il faut procéder à une extension de zéros jusqu'aux 32 bits lors de l'écriture dans les registres;
– **int*nn*** représente un entier signé. Si les valeurs sont inférieures à 32 bits, il faut procéder à une extension de signe jusqu'aux 32 bits lors de l'écriture dans les registres;
– **void *** représente un pointeur générique;
– **[u]int*nn* *** représente un pointeur vers une valeur de type [u]int*nn* ou vers un tableau de ces valeurs;
– **struct struct_type *** est une référence à un pointeur vers une structure (ou un tableau de structures) dans la mémoire. Les structures sont toujours passées par référence selon cette convention.

8.2 Valeur d'erreur

La plupart des appels SYSCALL renvoient un mot négatif pour signaler qu'une situation d'erreur a été détectée. Le Tableau 1 contient la liste de ces valeurs d'erreurs.

Tableau 1 – Valeurs d'erreur

Valeur	Nom symbolique	Signification
–49	EPERM	Un appel a été lancé vers une fonction SYSCALL ou CLIB qui n'existe pas.
–50	EINVAL	L'un des paramètres est faux.
–51	ERRSYSCALLMSGQUEUE	Le nombre de messages envoyés à l'**Hôte ECI** dépasse la capacité de la mémoire tampon de celui-ci.
–52	ERRHEAPSIZE	La valeur de la taille du tas demandée n'est pas conforme.
–53	ERRSTACKSIZE	La valeur de la taille de la pile demandée n'est pas conforme.

8.3 SYS_EXIT

ID SYSCALL: 0x0001

Description: Arrête la machine virtuelle et envoie un code d'explication ("raison").

Déclaration: void SYS_EXIT(uint32 reason)

Opérandes: La **raison** de l'arrêt.

Retour: Aucun.

NOTE – La **raison** prend l'une des valeurs indiquées dans le Tableau 2.

Tableau 2 – Valeurs de la raison passée dans SYS_EXIT

Raison	Signification
0	Arrêt normal
0x00000001..0x7FFFFFFF	Situation d'erreur propre au fournisseur du Client ECI
0x80000000..0xFFFFFFFF	Réservé à un futur emploi

8.4 SYS_PUTMSG

ID SYSCALL: 0x0003

Description: Envoie un message asynchrone (demande ou réponse).

Déclaration: int32 SYS_PUTMSG(MessageBuffer *msg_buffer)

Le format du bloc tampon de message est défini dans la norme [UIT-T J.1012].

Opérandes: **msg_buffer** est un pointeur vers un bloc tampon de message.

Retour: L'identifiant du message attribué par l'**Hôte ECI** (valeur non négative de 16 bits) ou l'une des valeurs (négatives) d'erreur ci-dessous: **ERRSYSCALLMSGQUEUE** (Tableau 1).

NOTE – On considère que l'appel ne se bloque pas dans des conditions d'exploitation normale de l'**Hôte ECI**. L'**Hôte ECI** copie le contenu de l'opérande msg_buffer, qui peut être réutilisé immédiatement par le **Client ECI** après le retour du SYSCALL.

8.5 SYS_GETMSG

ID SYSCALL: 0x0004

Description: Extrait le message suivant (que ce soit une demande ou une réponse) de l'**Hôte ECI**. Le SYSCALL se bloque si aucun message n'est disponible.

Déclaration: (MessageBuffer *) SYS_GETMSG()

Le format du bloc tampon de message (MessageBuffer) est défini dans la norme [UIT-T J.1012].

Opérandes: Aucun.

Retour: Le pointeur vers le tampon contenant le message suivant de l'**Hôte ECI** ou l'une des valeurs d'erreur indiquées dans le Tableau 1.

NOTE – L'appel se bloque si l'**Hôte ECI** n'a plus de message dans la file d'attente pour le **Client ECI**. L'**Hôte ECI** ne modifiera pas le contenu du tampon de message jusqu'à l'appel SYS_GETMSG SYSCALL suivant. Les **Clients ECI** souhaitant accéder aux données du message après l'appel SYS_GETMSG suivant doivent copier ces données.

8.6 SYS_HEAPSIZE

ID SYSCALL: 0x0100

Description: Demande adressée à l'**Hôte ECI** pour modifier la taille du tas selon le paramètre spécifié.

Déclaration: int32 SYS_HEAPSIZE(uint32 heapsize)

Opérandes: **heapsize**: taille du tas du **Client ECI**. Cette valeur ne doit pas être négative, doit être un multiple de 4 et ne doit pas dépasser la taille du tas dans le segment de la pile.

Retour: Le décalage de l'adresse mémoire en octets figurant dans le paramètre DATA_BASE_ADDRESS qui constitue l'adresse mémoire la plus basse en-dehors du tas dans la mémoire adressable, ou toute valeur d'erreur (négative) figurant dans le paramètre ci-dessous: **ERRHEAPSIZE** (Tableau 1).

NOTE – L'appel se bloque si l'**Hôte ECI** n'a pas de message dans la file d'attente pour le **Client ECI**. Lors de l'initialisation du **Client ECI**, SYS_HEAPSIZE(0) renvoie le décalage du début du segment du tas (dont la taille est zéro à ce moment-là).

8.7 SYS_STACKSIZE

ID SYSCALL: 0x0200

Description: Demande adressée à l'**Hôte ECI** pour modifier la taille de la pile selon le paramètre spécifié.

Déclaration: int32 SYS_STACKSIZE(uint32 stacksize)

Opérandes: **stacksize**: taille du tas du **Client ECI**. Cette valeur ne doit pas être négative, doit être un multiple de 4 et ne doit pas dépasser la taille de la pile dans le segment du tas.

Retour: Le décalage de l'adresse mémoire en octets figurant dans le paramètre DATA_BASE_ADDRESS qui constitue l'adresse mémoire la plus basse de la pile dans la mémoire adressable, ou toute valeur d'erreur (négative) figurant dans le paramètre ci-dessous: **ERRSTACKSIZE** (Tableau 1).

NOTE – L'appel se bloque si l'**Hôte ECI** n'a pas de message dans la file d'attente pour le **Client ECI**.

8.8 SYS_SYNCCALL

ID SYSCALL: 0x1000

Description: Le **Client ECI** envoie un message synchrone à l'**Hôte ECI** et suspend l'exécution jusqu'à la réponse de l'appel système.

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

Opérandes: **tag**: même définition que pour le champ MsgTag dans la structure du MessageBuffer. Le champ MsgFlags doit être mis à zéro et l'**Hôte ECI** doit l'ignorer. **p1…pn**: paramètres de l'appel synchrone. Pour les messages *get* dont la réponse est plus longue qu'une entité de 32 bits, **p1** est l'adresse de début de l'emplacement mémoire dans lequel la réponse va être écrite, et **p2.. pn** sont

les paramètres du message *set*. Tous les paramètres habituels, y compris les structures et les tableaux, sont passés par référence.

Retour: Si la réponse des messages *get* tient dans 32 bits, le retour est envoyé. Dans tous les autres cas, aucun retour n'est envoyé. Toutes les erreurs sont ignorées; les configurations contenant des paramètres erronés ne produisent aucune réponse et n'ont aucun effet. Les messages *call* renvoient le code de statut défini dans la sémantique qui leur est propre. Les réponses peuvent être envoyées à l'emplacement spécifié dans les paramètres du pointeur vers le SYSCALL, conformément à la sémantique propre au message.

NOTE – Ce SYSCALL ne se BLOQUE pas.

8.9 SYS_CL IB

ID SYSCALL:0x0300

Description: Ce SYSCALL joue un rôle d'interface de programmation d'application (API) et permet au **Client ECI** d'utiliser les fonctions courantes de la bibliothèque C. L'ensemble des fonctions prises en charge sont indiquées dans la l'Annexe C.

Déclaration: SYS_CLIB(uint32 clibfunc, etc.)

Opérandes: **clibfunc** détermine la fonction de la bibliothèque C à appeler (voir la description dans l'Annexe C). Tous les autres opérandes sont définis dans la liste des appels aux fonctions C.

Retour: L'une des valeurs de retour indiquées dans la bibliothèque décrite à l'Annexe C, ou l'une des valeurs d'erreur indiquées dans le Tableau 1.

NOTE – Étant donné QUE chaque fonction de la bibliothèque C prend en charge un nombre et des types de paramètres différents, ceux-ci ne sont pas décrits de manière explicite dans la présente Recommandation. L'annexe contient le détail du format de tous les opérandes. Tous les opérandes du paramètre SYS_CLIB sont des valeurs scalaires ou des pointeurs vers des valeurs non scalaires dans la mémoire de la machine virtuelle. Certaines fonctions de la bibliothèque C pouvant prendre en charge des paramètres non scalaires, la machine virtuelle doit convertir les paramètres passés par référence en paramètres passés par valeur avant de passer l'exécution à la bibliothèque.

9 Cycle de vie du Bytecode

9.1 Introduction

La machine virtuelle est mise en oeuvre au sein du micrologiciel de l'**Hôte ECI**. Elle est chargée et exécutée de manière dynamique par le système d'exploitation de l'**Hôte ECI** lorsqu'un **Client ECI** doit s'exécuter. Il est possible de créer plusieurs instances de la machine virtuelle pour différents **Clients ECI** s'il est nécessaire d'en disposer simultanément.

Le **Client ECI** est écrit dans le jeu d'instructions de la machine virtuelle comme indiqué plus haut. Il est programmé par un fournisseur de systèmes informatiques (le fournisseur du système d'accès conditionnel ou l'opérateur du système de gestion des droits numériques) et il est mis à la disposition de l'**Hôte ECI** en tant qu'image écrite en code logique. Il est ensuite transformé localement pour être adapté à la structure propre à l'**Hôte ECI** et à son environnement d'exploitation, puis il est chargé dans la machine virtuelle au moment où il est nécessaire. Il s'exécute dans la machine virtuelle jusqu'à ce qu'il soit arrêté de manière délibérée (ou qu'une situation d'erreur se produise). Son exécution prend alors fin et la machine virtuelle s'arrête.

9.2 Chargement d'un nouveau Client ECI dans la machine virtuelle

La machine virtuelle joue le rôle d'hôte intermédiaire pour un **Client ECI** extérieur, exactement comme si ce **Client ECI** était une application native s'exécutant dans l'**Hôte ECI**. La seule différence tient au fait que le **Client ECI** est installé par le système d'exploitation de l'**Hôte ECI** dans la machine virtuelle au lieu d'être une application native.

Pour charger le **Client ECI**, le sous-système de la machine virtuelle crée tout d'abord un contexte de processeur virtuel. Pour que le chargement soit possible, cette opération nécessite notamment d'attribuer de la mémoire à la machine virtuelle et d'écrire le contenu des segments de code et de données dans cette mémoire comme s'il s'agissait d'applications natives, sauf que le code et les données initialisées fournis dans les fichiers au format exécutable et reliable (ELF) [ETSI GS ECI 001-4] (voir les informations détaillées dans l'Annexe D) sont transférés vers la mémoire attribuée à la machine virtuelle.

Comme le programme ne peut pas accéder au segment de code, on peut choisir, pour une mise en oeuvre particulière, d'effectuer une forme quelconque de prétraitement du code (par exemple une optimisation) au moment du chargement. De fait, la présente Recommandation ne fait que décrire le format du programme dans l'image. La représentation interne dépend entièrement de chaque mise en oeuvre, la seule condition étant que toutes les références au code restent utilisables par le programme avec la même sémantique.

Une autre possibilité consiste à prétraiter l'image du **Client ECI** lorsqu'elle est extraite pour la première fois vers l'**Hôte ECI** et à la stocker sous une forme permettant de la charger à la demande. Cette méthode est plus efficace pour conserver et lancer des **Clients ECI** s'ils doivent être déchargés et rechargés régulièrement.

9.3 Initialisation de la machine virtuelle

Il faut tout d'abord créer le contexte général de l'unité centrale de traitement de la machine virtuelle, à savoir le fichier de registres, la pile de contrôle, les zones consacrées aux données et à la pile et le compteur ordinal (PC), et éventuellement tout système logique et tout indicateur de contrôle ou de statut. Ces éléments ne sont pas décrits dans la présente Recommandation car ils dépendent de la mise en oeuvre.

Le fichier de registres est organisé de telle sorte que le registre R0 se trouve au début de l'espace réservé à ce fichier. Tous les registres sont mis à zéro. Le pointeur de cadre et le pointeur de registre (respectivement R0 et R16) de la première fenêtre sont donc disposés de manière à ce qu'au moment où le premier mot est poussé dans la pile, le pointeur de pile soit pré-décrémenté à 4 (0xFFFFFFFC) et que le mot soit stocké à cet emplacement.

Le compteur ordinal est initialisé au début du segment de code, sauf si le membre *e_entry* de l'en-tête du fichier ELF [ETSI GS ECI 001-4] se trouvant dans le fichier image du **Client ECI** a une valeur non nulle; dans ce cas, on attribue au compteur ordinal la valeur (adresse virtuelle) indiquée dans le membre *e_entry*. Le contrôle est ensuite remis à l'élément de la machine virtuelle chargé de l'exécution, qui est appelé "boucle d'exécution centrale".

9.4 Boucle d'exécution centrale

Le coeur de la machine virtuelle se trouve dans la boucle d'exécution centrale, qui lit chaque instruction séquentielle et la traduit en actions correspondantes sur les registres et la mémoire de la machine virtuelle, et/ou en appels à la bibliothèque. La boucle exécute les instructions jusqu'à ce qu'une exception se produise. L'arrêt du programme peut s'inscrire dans l'exécution normale de celui-ci, par exemple s'il exécute l'appel au système `SYS_EXIT`, ou il peut survenir en raison d'une situation d'erreur, par exemple si la pile de contrôle déborde.

Si la "boucle d'exécution centrale" s'arrête, la machine virtuelle s'arrête également. Il faudra la relancer si le **Client ECI** doit à nouveau intervenir par la suite.

Annexe A

Ressources du système de la machine virtuelle

(Cette annexe fait partie intégrante de la présente Recommandation.)

Dans la présente Recommandation, on emploie les paramètres suivants pour définir le fonctionnement de la machine virtuelle. Des valeurs pour ces paramètres sont proposées dans le document [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

Annexe B

Opcodes de la machine virtuelle

(Cette annexe fait partie intégrante de la présente Recommandation.)

Les codes ci-après permettent de coder les instructions dans l'image binaire. Cet aperçu général montre les différents formats possibles. La ligne de résumé contient la liste des champs constituant l'instruction, ces champs étant séparés par une virgule. Ils comportent soit des bits explicites, soit un nom de champ suivi d'un indicateur de la largeur du champ. Différents opcodes au même format sont proposés avec la structure correspondante occupant le champ "op". Les bits sont énumérés dans l'ordre big-endian.

On code par exemple `SUB R3,R5,R17` de la manière suivante:

```
1011 00001 00011 00101 10001
```

ou en quartets:

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

ou en octets:

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

Chaque nom d'instruction est suivi d'un nombre qui est le nombre d'opcodes définis dans cette instruction. Ce nombre doit être le même dans toutes les versions suivantes du jeu d'instructions de la machine virtuelle.

Les champs des opcodes ne doivent pas s'étendre sur plus de quatre octets. Les champs de 32 bits doivent donc débuter à une limite d'octet. Aucun champ ne doit dépasser 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
```

```
1100110000000010000, 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

1100110000000010001, 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
```

Annexe C

Routines courantes de la bibliothèque C

(Cette annexe fait partie intégrante de la présente Recommandation.)

C.1 Introduction

La présente annexe contient une description détaillée des routines courantes de la bibliothèque C99 [b-ISO IEC 9899] que le **Client ECI** peut employer. Pour chaque fonction, les opérandes passés par le **Client ECI** et la valeur de retour sont définis en détail.

À noter que le terme "chaîne" s'entend d'une séquence d'octets non nuls se terminant par un octet nul.

Les fonctions décrites ci-dessous prennent la forme d'appels classiques à la bibliothèque C. Dans tous les cas, le premier paramètre est écrit dans le registre R2 (puisque R1 contient le paramètre clibfunc, qui est l'identifiant de la fonction). La déclaration repose sur l'hypothèse que toutes les valeurs sont passées comme des valeurs scalaires ou des pointeurs vers des valeurs non scalaires. Si une fonction de la bibliothèque appelle un paramètre non scalaire qui doit être passé par une valeur, l'appel au système le passera par référence et la machine virtuelle devra convertir le paramètre conformément aux spécifications de la bibliothèque.

Les valeurs de retour sont toujours des valeurs scalaires ou des pointeurs envoyés dans R1.

NOTE – La valeur attribuée au paramètre clibfunc est définie de la manière suivante:

((clibfunc >> 8) & 0x000000FF) = Le numéro de sous-chapitre du chapitre C courant traitant des fonctions de la bibliothèque, exprimé en décimales en code binaire. (Dans la bibliothèque C99, il s'agit du chapitre numéro 7 et la bibliothèque <string.h> se trouve dans le sous-chapitre 21.)

((clibfunc >> 4) & 0x0000000F) = Type de fonction de la bibliothèque – numéro suivant le numéro du sous-chapitre.

(clibfunc & 0x0000000F) = Numéro de fonction d'un type particulier dans la bibliothèque – numéro suivant le numéro du type de fonction.

Par exemple, la fonction memmove() est décrite dans la section 7.21.2.2 de la norme [b-ISO IEC 9899]. Le paramètre clibfunc est donc codé ainsi: 0x00002122.

C.2 memmove

clibfunc: 0x00002122

Description: Copie n octets depuis la mémoire pointée par s2 vers la mémoire pointée par s1. Les mémoires peuvent se chevaucher.

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

Retour: s1

C.3 strcpy

clibfunc: 0x00002123

Description: Copie la chaîne (y compris le caractère de fin) pointée par s2 dans la mémoire pointée par s1. La réponse est indéfinie si les zones mémoire se chevauchent.

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

Retour: s1

C.4 strncpy

clibfunc: 0x00002124

Description: Comme pour strcpy(), mais n octets au plus sont copiés. Si la longueur de s2 est supérieure à n, un octet nul est ajouté (à s1[n]).

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

Retour: s1

C.5 strcat

clibfunc: 0x00002131

Description: Ajoute la copie de la chaîne pointée par s2 (y compris le caractère de fin) à la fin de la chaîne pointée par s1. La réponse est indéfinie si les zones mémoire se chevauchent.

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

Retour: s1

C.6 strncat

clibfunc: 0x00002132

Description: Ajoute la copie de la chaîne pointée par s2 (y compris le caractère de fin) à la fin de la chaîne pointée par s1, mais n octets au plus sont copiés. . Si la longueur de s2 est supérieure à n, un octet nul est ajouté (à n+1 octets après le dernier octet non nul du s1 original). La réponse est indéfinie si les zones mémoire se chevauchent.

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

Retour: s1

C.7 memcmp

clibfunc: 0x00002141

Description: Compare les n premiers octets pointés par s1 avec les n premiers octets pointés par s2.

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

Retour: R1==0 si tous les octets sont identiques, sinon R1 dépend de la première position en partant de la gauche pour laquelle les valeurs diffèrent.

R1>0 si l'octet de s1 à cette position est supérieur à l'octet de s2.

R1<0 si l'octet de s1 à cette position est inférieur à l'octet de s2.

C.8 strcmp

clibfunc: 0x00002142

Description: Compare les chaînes pointées par s1 et s2.

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

Retour: R1==0 si les chaînes sont identiques, sinon R1 dépend de la première position en partant de la gauche pour laquelle les valeurs diffèrent.

R1>0 si l'octet de s1 à cette position est supérieur à l'octet de s2.

R1<0 si l'octet de s1 à cette position est inférieur à l'octet de s2.

C.9 strncmp

clibfunc: 0x00002144

Description: Compare les chaînes pointées par s1 et s2, mais seulement jusqu'à n octets.

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

Retour: R1==0 si les chaînes sont identiques, sinon R1 dépend de la première position en partant de la gauche pour laquelle les valeurs diffèrent.

R1>0 si l'octet de s1 à cette position est supérieur à l'octet de s2.

R1<0 si l'octet de s1 à cette position est inférieur à l'octet de s2.

C.10 memchr

clibfunc: 0x00002151

Description: Trouve la première occurrence de l'octet dans c à l'intérieur des n octets pointés par s.

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

Retour: Un pointeur vers l'octet trouvé, ou 0 si aucun octet n'a été trouvé.

C.11 strchr

clibfunc: 0x00002152

Description: Trouve la première occurrence de l'octet dans c à l'intérieur de la chaîne pointée par s, la recherche allant jusqu'à l'octet de fin (nul) inclus.

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

Retour: Un pointeur vers l'octet trouvé, ou 0 si aucun octet n'a été trouvé.

C.12 strcspn

clibfunc: 0x00002153

Description: Calcule la longueur du segment initial maximum de la chaîne pointée par s1 qui se compose entièrement d'octets n'appartenant pas à la chaîne pointée par s2.

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

Retour: La longueur calculée.

C.13 strpbrk

clibfunc: 0x00002154

Description: Trouve la première occurrence, dans la chaîne pointée par s1, de tout octet présent dans la chaîne pointée par s2.

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

Retour: L'emplacement du premier octet répondant à ce critère, ou 0 si aucun octet répondant à ce critère n'a été trouvé.

C.14 strrchr

clibfunc: 0x00002155

Description: Trouve la dernière occurrence de l'octet dans c au sein de la chaîne pointée par s, la recherche allant jusqu'à l'octet de fin (nul) inclus.

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

Retour: Un pointeur vers l'octet trouvé, ou 0 si aucun octet n'a été trouvé.

C.15 strspn

clibfunc: 0x00002156

Description: Calcule la longueur du segment initial maximum de la chaîne pointée par s1 qui se compose entièrement d'octets appartenant à la chaîne pointée par s2.

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

Retour: La longueur calculée.

C.16 strstr

clibfunc: 0x00002157

Description: Trouve la première occurrence de la chaîne pointée par s1 (octet de fin exclu) dans la chaîne pointée par s2.

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

Retour: Un pointeur vers la position trouvée, ou 0 si aucune occurrence n'a été trouvée.

C.17 memset

clibfunc: 0x00002161

Description: Copie n fois l'octet de poids faible de c dans la mémoire pointée par s.

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

Retour: s

Annexe D

Format du fichier du Client ECI

(Cette annexe fait partie intégrante de la présente Recommandation.)

Le fichier image du **Client ECI** doit être conforme aux spécifications de format des fichiers objets ELF [ETSI GS ECI 001-4]. La présente annexe contient les informations nécessaires pour se conformer aux spécifications de la machine virtuelle. Étant donné que celle-ci prend en charge l'architecture 32 bits et le format little-endian, il convient d'employer les valeurs du Tableau D.1 pour indiquer l'identifiant du fichier ELF dans le paramètre *e_ident* [ETSI GS ECI 001-4].

Tableau D.1 – Valeurs du paramètre *e_ident* conformes à l'interface ECI

Nom	Valeur
e_ident[EI_CLASS]	ELFCLASS32
e_ident[EI_DATA]	ELFDATA2LSB

Le Tableau D.2 indique les valeurs à employer pour certains éléments de l'en-tête du fichier ELF.

Tableau D.2 – Valeurs conformes à l'interface ECI pour certains éléments de l'en-tête du fichier ELF

Nom	Valeur
e_type	ET_EXEC
e_machine	ET_NONE
e_version	EV_CURRENT

Le chargeur rejettera tout fichier image du **Client ECI** dont les valeurs diffèrent de celles qui sont indiquées dans la présente annexe.

Appendice I

Domaines nécessitant des développements supplémentaires

(Cet appendice ne fait pas partie intégrante de la présente Recommandation.)

Il a été établi que la présente Recommandation nécessite des développements et une validation supplémentaires afin de répondre aux exigences énoncées dans la Recommandation [UIT-T J.1010], et que la Recommandation [UIT-T J.1010] doit être mise à jour pour refléter les exigences de la spécification Enhanced Content Protection (ECP) de MovieLabs [b-ECP]. Les Recommandations [b-UIT-T J.1011], [UIT-T J.1012], UIT-T J.1013, [b-UIT-T J.1014], [b-UIT-T J.1015] et [b-ITU-T J.1015.1] devront à l'avenir être mises à jour pour refléter ces mises à jour de la Recommandation [UIT-T J.1010].

Plusieurs États Membres de l'UIT ainsi que des parties prenantes de divers secteurs – notamment des fabricants d'appareils et de composants électroniques, des propriétaires et des titulaires de licences de contenus protégés par le droit d'auteur, des fournisseurs de services over-the-top (OTT) et de télévision linéaire, et des fournisseurs de solutions de système d'accès conditionnel (CAS) et de gestion des droits numériques (DRM) – basées dans le monde entier se sont déclarés préoccupés par le fait que l'interface commune intégrée (ECI) ne répond pas pleinement aux exigences de la spécification ECP, ni aux exigences plus larges du secteur en matière de protection des contenus.

Plus précisément, leurs préoccupations ont été exprimées dans des contributions soumises à la réunion de la Commission d'étude 9 (CE 9) de l'UIT-T tenue du 16 au 23 avril 2020. Dans leurs contributions, Israël, l'Australie, Samsung (Membre de secteur de l'UIT-T) ainsi que Sky Group et MovieLabs (Associés de la CE 9) ont proposé d'apporter plusieurs modifications aux Recommandations relatives à l'interface ECI, mais aucun accord n'a été trouvé à leur sujet. Ces points sont répertoriés dans le document [b-CE 9 Rapport 17 Ann.1].

Les propositions visent à:

1) simplifier le système ECI en réduisant son champ d'application;
2) supprimer la gestion DRM;
3) supprimer le rechiffrement de contenu;
4) supprimer la gestion des logiciels;
5) ajouter des API pour les opérations de stockage et de chiffrement sécurisées;
6) autoriser des échelles de clés propres aux fournisseurs;
7) utiliser les exigences TEE UIT-T J.1207;
8) inclure l'implémentation TEE pour les machines virtuelles;
9) augmenter la puissance des algorithmes de chiffrement, par exemple en utilisant SHA-384;
10) utiliser des certificats standard, comme UIT-T X.509;
11) revoir les communications entre clients;
12) mener des échanges supplémentaires avec l'ETSI;
13) effectuer une évaluation par les pairs supplémentaires;
14) envisager des alternatives au modèle de l'autorité de confiance;
15) définir plus précisément les aspects techniques des règles de conformité et de robustesse de l'interface ECI;
16) ajouter des exigences en matière de diversité, par exemple la randomisation de l'espace d'adresses;
17) ajouter des exigences relatives à la vérification de l'intégrité de l'exécution.

Ces propositions reflètent le fait que la protection des contenus et les menaces de compromission de contenu sont en constante évolution. L'interface ECI a été conçue initialement près de dix ans avant l'approbation de la présente Recommandation UIT-T. Des systèmes comme l'interface ECI doivent être évalués régulièrement en fonction à la fois des techniques d'attaque et des exigences de protection du secteur les plus récentes.

D'autres mécanismes existent pour assurer l'interopérabilité. En particulier, s'agissant de la gestion DRM, la plupart des services vidéo sur l'Internet ont déployé d'autres solutions pour assurer l'interopérabilité et répondre à leurs besoins.

Il est important d'apporter davantage de clarté, car de nombreux États Membres considèrent les normes de l'UIT comme des guides importants pour le développement de leurs marchés et de leurs secteurs. La liste de préoccupations vise à faire en sorte que la mise en œuvre de l'interface ICE sur les marchés nationaux puisse reposer sur une compréhension parfaite des conséquences de la présente Recommandation de l'UIT-T et que les questions soient prises en compte au moment d'examiner une législation, une réglementation ou des besoins du marché exigeant que les équipements de télévision numérique grand public soient interopérables. Elle vise également à faire en sorte que les fabricants d'équipements techniques, qui peuvent préférer utiliser un ensemble unique d'exigences ou d'autres normes pour concevoir les produits, puissent prendre en compte ces questions lors du développement de produits pour des marchés différents.

Bibliographie

[b-UIT-T J.1010] Recommandation UIT-T J.1010 (2016), *Interface commune intégrée pour les solutions CA/DRM interchangeables; Cas d'utilisation et exigences*.

[b-UIT-T J.1011] Recommandation UIT-T J.1011 (2016), *Interface commune intégrée pour les solutions CA/DRM interchangeables; Architecture, définitions et vue d'ensemble*.

[b-UIT-T J.1014] Recommandation UIT-T J.1014 (2020), *Interface commune intégrée pour les solutions CA/DRM interchangeables; Sécurité évoluée – Fonctionnalités propres aux interfaces ECI*.

[b-UIT-T J.1015] Recommandation UIT-T J.1015 (2020), *Interface commune intégrée pour les solutions CA/DRM interchangeables; Sécurité évoluée – Bloc d'échelle de clés*.

[b-UIT-T J.1015.1] Recommandation UIT-T J.1015.1 (2020), *Interface commune intégrée pour les solutions CA/DRM interchangeables; Système de sécurité évoluée – Bloc d'échelle de clés: authentification des informations sur les règles d'utilisation des mots de contrôle et des données associées 1*.

[b-UIT-T J Suppl. 7] Supplément 7 à la série J de Recommandations UIT-T (2020), *Interface commune intégrée pour les solutions CA/DRM interchangeables; Lignes directrices pour la mise en oeuvre de l'interface commune intégrée*.

[b-CE 9 Rapport 17 Ann.1] Rapport de la réunion de la CE 9 de l'UIT-T, SG9-R17-Annexe 1 (2020), Annexe 1 au Rapport 17 de la réunion de la CE 9 organisée de manière entièrement virtuelle du 16 au 23 avril 2020. https://www.itu.int/md/T17-SG09-R-0017/en

[b-ECP] MovieLabs Specification for Enhanced Content Protection – Version 1.2
Disponible à l'adresse: <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*. Available at <https://refspecs.linuxfoundation.org/elf/elf.pdf>.

SÉRIES DES RECOMMANDATIONS UIT-T

Série A	Organisation du travail de l'UIT-T
Série D	Principes de tarification et de comptabilité et questions de politique générale et d'économie relatives aux télécommunications internationales/TIC
Série E	Exploitation générale du réseau, service téléphonique, exploitation des services et facteurs humains
Série F	Services de télécommunication non téléphoniques
Série G	Systèmes et supports de transmission, systèmes et réseaux numériques
Série H	Systèmes audiovisuels et multimédias
Série I	Réseau numérique à intégration de services
Série J	**Réseaux câblés et transmission des signaux radiophoniques, télévisuels et autres signaux multimédias**
Série K	Protection contre les perturbations
Série L	Environnement et TIC, changement climatique, déchets d'équipements électriques et électroniques, efficacité énergétique; construction, installation et protection des câbles et autres éléments des installations extérieures
Série M	Gestion des télécommunications y compris le RGT et maintenance des réseaux
Série N	Maintenance: circuits internationaux de transmission radiophonique et télévisuelle
Série O	Spécifications des appareils de mesure
Série P	Qualité de transmission téléphonique, installations téléphoniques et réseaux locaux
Série Q	Commutation et signalisation et mesures et tests associés
Série R	Transmission télégraphique
Série S	Equipements terminaux de télégraphie
Série T	Terminaux des services télématiques
Série U	Commutation télégraphique
Série V	Communications de données sur le réseau téléphonique
Série X	Réseaux de données, communication entre systèmes ouverts et sécurité
Série Y	Infrastructure mondiale de l'information, protocole Internet, réseaux de prochaine génération, Internet des objets et villes intelligentes
Série Z	Langages et aspects généraux logiciels des systèmes de télécommunication

Imprimé en Suisse
Genève, 2020