(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 648 121**

(51) Int. CI.:

G06F 21/54 (2013.01)
G06Q 40/00 (2012.01)
G06F 21/75 (2013.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(86) Fecha de presentación y número de la solicitud internacional: **06.12.2004 PCT/IB2004/004030**

(87) Fecha y número de publicación internacional: **16.06.2005 WO05055021**

(96) Fecha de presentación y número de la solicitud europea: **06.12.2004 E 04801337 (9)**

(97) Fecha y número de publicación de la concesión europea: **05.04.2017 EP 1692594**

(54) Título: **Método para asegurar la ejecución de un programa contra ataques por radiación u otros**

(30) Prioridad:

04.12.2003 EP 03293036

(45) Fecha de publicación y mención en BOPI de la traducción de la patente:
28.12.2017

(73) Titular/es:

GEMALTO SA (100.0%)
6, RUE DE LA VERRERIE
92190 MEUDON, FR

(72) Inventor/es:

GIRAUD, NICOLAS y
RAINSARD, STÉPHANE

(74) Agente/Representante:

CASANOVAS CASSA, Buenaventura

DESCRIPCIÓN

Método para asegurar la ejecución de un programa contra ataques por radiación u otros.

Esta invención se refiere a un método ya un dispositivo para asegurar un montaje electrónico que implementa un programa a proteger. Más precisamente, el propósito del método es proponer una defensa contra ataques por radiación, flash, luz, láser, fallo u otro y más generalmente contra cualquier ataque que perturbe la ejecución de las instrucciones del programa. Estos ataques modifican las instrucciones a ejecutar, dando como resultado la no ejecución o la ejecución incorrecta de ciertas partes del programa.

CAMPO TÉCNICO

Cuando se ejecuta un programa, los ataques por ejemplo por láser, fallo o radiación electromagnética modifican los códigos de instrucción ejecutados por el procesador, por ejemplo convirtiendo cualquier instrucción codop en codop 00h (BRSETO en 6805, NOP en 8051 y AVR): las instrucciones del programa se sustituyen por instrucciones inoperativas. En consecuencia, ciertas secciones del código fallan a la hora de ejecutarse o se ejecutan irregularmente, por ejemplo la ejecución de instrucciones inoperantes en lugar de una secuencia de procesamiento de seguridad, por ejemplo en un sistema operativo para tarjeta inteligente. Los ataques pueden perturbar la operación del procesador y provocar saltos inoportunos en la memoria del programa.

La EP-A-13168763 (Hewlett Packard) 4 Junio 2003, y Davida G I et al: "Defendiendo sistemas contra virus mediante autenticación criptográfica", procesos del simposium sobre seguridad y privacidad, Oackland, 1-3 Mayo, 1989, Washington, IEEE Comp. Soc. Press, US, 1 Mayo 1989, páginas 312-318, XP010016032 ISBN: 0-8186-1939-2 describen métodos de protección frente a virus.

Este solicitante presentó una solicitud de patente francesa Nº 0016724 el 21 de diciembre de 2000 relativa a un método para asegurar la ejecución de un programa almacenado en un módulo electrónico controlado por microprocesador, así como el módulo electrónico asociado y la tarjeta de circuito integrado. La solución protegida en dicha aplicación consiste en activar intermitentemente interrupciones y desviar así la ejecución del programa para protegerlo contra posibles ataques. Esta solución ofrece una buena probabilidad de detectar y prevenir los ataques por radiación. Sin embargo, algunos ataques pueden no ser detectados, especialmente si el ataque ocurre brevemente entre dos interrupciones.

Entre las defensas conocidas, otra solución consiste en establecer banderas en un byte de la memoria RAM a intervalos regulares y realizar una comprobación, en un punto particular establecido. Sin embargo, la instalación de este tipo de defensa es tediosa, ya que se deben asignar áreas de memoria volátiles específicas y se debe añadir el procesamiento en el código que se desea proteger, donde sea necesario. Además, como los ataques de este tipo son cada vez más cortos y más precisos, las soluciones conocidas son cada vez menos eficaces. En primer lugar, el ataque puede ser lo suficientemente corto como para no tener ningún efecto en el establecimiento de banderas; la ejecución de una sección del programa puede, por lo tanto, evitarse de una manera totalmente indetectable. En segundo lugar, el mismo software de verificación de la bandera puede ser perturbado.

Un objetivo de esta invención es proponer protección eficaz incluso para ataques muy cortos.

Otro objetivo de esta invención es proponer una solución que pueda ser implementada en los componentes actuales sin adaptación, que consume pocos recursos y que no reduce el rendimiento del conjunto en el que se implementa.

SUMARIO DE LA INVENCIÓN

Esta invención se refiere a un método para asegurar la ejecución de un programa en un conjunto electrónico que incluye medios de procesamiento de datos y medios de almacenamiento de datos, caracterizado porque consiste en comprobar la ejecución de cada instrucción de al menos una porción de dicho programa mediante la realización durante la ejecución de dicha parte de un cálculo que utiliza valores predeterminados, dependiendo de o asociado con cada una de dichas instrucciones y comparando el resultado obtenido con un valor precalculado.

Esta invención también se refiere a un módulo electrónico en el que se implementa dicho método, una tarjeta que comprende dicho módulo y un programa para implementar dicho método.

BREVE DESCRIPCIÓN DE LOS DIBUJOS

Otros objetivos, características y ventajas de la invención aparecerán tras la lectura de la descripción que sigue de la implementación del método de acuerdo con la invención y de un modo de realización de un sistema electrónico diseñado para esta implementación, dado a modo de ejemplo no limitativo, y haciendo referencia a los dibujos adjuntos en los que:

- la figura 1 es una representación esquemática de un ejemplo de un dispositivo en el que se implementa el método según esta invención;
- la figura 2 es una clave para interpretar las figuras 3 a 7 adjuntas: un rectángulo sin sombrear a representa una porción de código ejecutada, un rectángulo sombreado b representa una porción de código que no se ha ejecutado. Las flechas sombreadas c representan un ataque: sus longitudes indican la duración del ataque; sus posiciones indican la parte del código del programa atacado. El rectángulo d con bordes en negrita representa la detección de una anomalía. El rectángulo sombreado de luz e con bordes finos representa datos precalculados. La llave f representa el alcance de un precálculo. El rectángulo g con las esquinas redondeadas representa una dirección de código. Los rectángulos h en perspectiva muestran el estado de la pila. Las amplias flechas sombreadas i cuyos extremos puntiagudos extienden los rectángulos básicos y las elipses sombreadas j representan mecanismos de hardware, las elipses no sombreadas k con doble borde representan mecanismos de software. Los rectángulos l en perspectiva con bordes en negrita representan mecanismos de software en la pila;
- la figura 3 es una representación esquemática de la ejecución de un programa cuando no se enfrenta a un ataque, en el que se han demostrado las etapas del método de seguridad según una forma de realización de esta invención;
- la figura 4 es una representación esquemática de la ejecución de un programa enfrentado a un ataque, en el que se han demostrado las etapas del método de seguridad según la forma de realización representada en la figura 3;
- la figura 5 es una representación esquemática de la ejecución de un programa cuando no se enfrenta a un ataque, en el que se han demostrado las etapas del método de seguridad según otra forma de realización de esta invención;
- la figura 6 es una representación esquemática de la ejecución de un programa enfrentado a un ataque en el que se han demostrado los pasos del método de seguridad según la forma de realización representada en la figura 5;
- la figura 7 es una representación esquemática de la ejecución de un programa cuando no se enfrenta a un ataque, en el que se han demostrado las etapas del método de seguridad según una variante de la forma de realización representada en la figura 5;

FORMA DE REALIZACIÓN DE LA INVENCIÓN

El propósito del método según la invención es asegurar un conjunto electrónico y, por ejemplo, un objeto portátil tal como una tarjeta inteligente que implementa un programa. El conjunto electrónico comprende al menos medios de procesamiento tales como un procesador y medios de almacenamiento tales como una memoria. El programa que se va a asegurar se instala en la memoria, por ejemplo, del tipo ROM (Read Only Memory) de dicho conjunto.

Como ejemplo no limitativo, el conjunto electrónico descrito a continuación corresponde a un sistema de a bordo que comprende un módulo electrónico 1 ilustrado en la figura 1. Este tipo de módulo se realiza generalmente como un microcircuito electrónico monolítico integrado, o chip, el cual una vez físicamente protegido por cualquier medio conocido puede ser montado sobre un objeto portátil como por ejemplo una tarjeta inteligente, un microcircuito o una tarjeta de circuito integrado (tarjeta de microprocesador, etc.) u otra tarjeta que pueda ser usada en varios campos.

El módulo electrónico 1 comprende un microprocesador CPU 3 con una conexión bidireccional a través de un bus interno 5 a una memoria no volátil 7 de tipo ROM, EEPROM, Flash, FeRam u otra que contiene el programa PRO 9 a ejecutar, una memoria volátil 11 de tipo RAM, medios de entrada/salida I/O 13 para comunicar con el exterior, medios de evaluación 15 tales como al menos un contador COUNTER.

El método de acuerdo con la invención consiste en comprobar que el programa 9 se ejecuta completamente tal como está almacenado en la memoria comprobando que cada instrucción en el flujo de ejecución es realmente ejecutada por el microprocesador 3.

El método de acuerdo con la invención consiste en comprobar la ejecución de cada instrucción de al menos una parte de dicho programa realizando, durante la ejecución de dicha parte, un cálculo aritmético usando valores predeterminados, dependiendo de o asociado con cada instrucción y comparando el resultado obtenido con un valor precalculado almacenado en dichos medios de almacenamiento. Un valor predeterminado significa cualquier valor que no dependa de las características físicas del sistema en el que se implemente el método y características físicas de la ejecución de dicha instrucción en este sistema.

El valor predeterminado que depende de una instrucción es, por ejemplo, un valor que depende del contenido, tipo, función, resultado y/o de cualquier otra característica asociada a dicha instrucción como tal. Se entiende por contenido de una instrucción cualquier elemento que forme dicha instrucción, incluyendo el código operativo y los parámetros, comprendiendo dicho contenido uno, varios o todos esos elementos. Por el tipo de una instrucción se entiende una característica de la instrucción que la clasifica en una categoría de instrucción especial. Por función de una instrucción se entiende la función realizada por dicha instrucción en dicha porción del programa en cuestión. Por resultado de una instrucción se entiende cualquier valor obtenido por la ejecución de dicha instrucción, si la hay.

De acuerdo con una forma de realización según esta invención representada en las figuras 3 y 4, el método consiste en calcular durante la operación del microprocesador una suma de comprobación del contenido de las instrucciones ejecutadas durante su ejecución y verificar la suma de comprobación calculada comparándola con un valor precalculado almacenado en la memoria. El valor precalculado puede almacenarse, por ejemplo, en el código protegido durante el desarrollo del programa correspondiente. Una suma de comprobación es una "suma" de un conjunto de elementos de datos, es decir, un valor calculado que depende del contenido de dichos datos, utilizado con fines de verificación.

Un programa comprende numerosas pruebas condicionales y llamadas de rutina traducidas en código de máquina usando ramas, saltos, instrucciones de llamada de rutina o equivalentes, es decir, instrucciones que crean ramas en el flujo de ejecución. Si las pruebas condicionales, las llamadas de rutina o las devoluciones de llamadas se siguen unas a las otras, la sección de código puede consistir en sólo unas pocas instrucciones antes de la siguiente rama en el árbol de posibles rutas de ejecución. Si hay un salto a otra rutina, el punto de entrada de la rutina puede ser alcanzado por varios caminos. En esta invención, cada punto de entrada, instrucción de salto o equivalente en el código de programa forma el inicio de una nueva sección de código para el cálculo de la suma de comprobación.

Para implementar un mecanismo de verificación de la suma de comprobación, el método de acuerdo con esta invención efectúa un precálculo de las sumas de comprobación en cada porción de código a proteger, delimitada por puntos de entrada o salida, direcciones de salto o por ramas, saltos, llamada de rutina o instrucciones de devolución de llamada, entrada en una rutina de manejo de interrupciones o retorno de dicha interrupción o equivalente.

Para aplicar el método de la presente invención a todo el código, como se muestra en las figuras 3 y 4, esta invención utiliza un compilador especialmente adaptado para realizar la tarea de precálculo de suma de comprobación (llave "suma de comprobación precalculada" en las figuras 3 y 4). El paso de las sumas de comprobación precalculadas como parámetros de las instrucciones de la sección de fin de código implica modificaciones significativas en los tipos conocidos de compilador ya que se modificad el conjunto de instrucciones del procesador.

La verificación de la suma de comprobación puede ser realizada por el procesador. El método de acuerdo con esta invención suministra al microprocesador al final de cada porción de código a proteger la suma de comprobación precalculada que se comparará ("verificación CKS" en las figuras 1 y 2) con la suma de control calculada por el procesador durante la ejecución ("cálculo CKS" figs. 1 y 2). Este valor se suministra, por ejemplo, como un parámetro de la instrucción de ramificación, salto, llamada de rutina o devolución de llamada o devolución de una rutina de tratamiento de interrupciones o equivalente y/o marca el final de una parte de código a proteger o de una instrucción que marca el final de una porción de código a proteger. Si una parte de código a proteger termina con una dirección de salto, la verificación de la suma de comprobación puede realizarse con una instrucción añadida especialmente en el conjunto de instrucciones del procesador o simplemente añadiendo una instrucción de salto incondicional a la siguiente instrucción.

La verificación se realiza durante la instrucción final de la porción de código a proteger con el valor precalculado pasado como parámetro. La suma de comprobación calculada por el procesador durante la ejecución es reinicializada para la siguiente sección de código. Si la verificación de la suma de comprobación detecta una diferencia, se activa una acción ("Detección de anomalía" en la figura 4).

La verificación de la suma de comprobación también podría ser realizada por el software sin modificar el conjunto de instrucciones del procesador comparando el valor calculado por el procesador con el valor precalculado. En este caso, no hay necesidad de modificar el conjunto de instrucciones de procesador para pasar la suma de comprobación precalculada como un parámetro de la instrucción final de la sección de código a proteger. La suma de comprobación calculada por el procesador durante la ejecución debe ser simplemente accesible al programa en un registro u otro.

Este método, que requiere sólo recursos limitados, ofrece la ventaja de proporcionar una protección completa del código que se va a ejecutar incluyendo no sólo el codop sino también los parámetros.

De acuerdo con otra forma de realización del método según esta invención representada en las figuras 5 a 7, se asigna un contador 15 a al menos una función, una secuencia o más generalmente a al menos una parte de dicho programa y, de acuerdo con una forma de realización, a cada parte, al inicio de su ejecución. Una porción de programa comprende al menos una instrucción. En el resto de la descripción, como ilustración, la parte considerada es una función. El método consiste en que el procesador incremente dicho contador, al ejecutar cada instrucción de dicha función en curso, por un valor específico a la instrucción ejecutada. Al final de la función, el valor alcanzado, resultado de la secuencia de instrucciones ejecutada, se comprueba comparando con un valor precalculado durante el desarrollo del programa y escrito en el código protegido.

El método de acuerdo con dicha forma de realización puede ser implementado sin añadir o modificar el conjunto de instrucciones del procesador. El método también se puede implementar sin adaptar el compilador.

Al comienzo de cada función, se asigna un contador a la función en una estructura de datos que podría ser la pila del procesador. El contador se inicializa a 0 o un valor determinado para la función ("Inicialización del contador" en las figuras 5 a 7). Para cada instrucción en el cuerpo de la función ejecutada antes de la instrucción de devolución de llamada, el contador de funciones se incrementa por un valor específico a la instrucción ("cálculo CF o CP" en las figuras 5 a 7). Al final de la función, el valor alcanzado por el contador se compara con el valor esperado, precalculado y utilizado como valor de referencia en el software ("comparación CF" en las figuras 5 a 7).

Como se muestra en la figura 6, si se produce un ataque durante la función, las instrucciones no se ejecutan de acuerdo con la secuencia planificada en el programa y el valor del contador al final de la función (CF1=0x25 en la figura 6) es diferente del valor esperado y comprobado (Valor precalculado VP=0x92 en la figura 6); se detecta entonces el ataque ("Detección anomalía" en la figura) y una acción específica es realizada por el programa o el procesador. Si la comprobación no detecta una anomalía, el contador de funciones es desasignado en la estructura de datos (en la figura 5, el contador CF1 se desasigna).

El código de función consiste en numerosas pruebas condicionales y llamadas de rutina o equivalentes, es decir, Instrucciones que crean ramas en el flujo de ejecución. Para obtener el mismo valor para el contador al final de la función, independientemente de la trayectoria de ejecución, las diversas ramas de ejecución posibles deben ser equilibradas.

El valor de incremento del contador para una instrucción puede ser el código máquina de la instrucción (código operativo + valores de parámetro). En este caso, el recuento calculado en las instrucciones ejecutadas corresponde a una suma de comprobación calculada en las instrucciones en el flujo de ejecución perteneciente a la función. Las instrucciones ejecutadas en las funciones llamadas o ejecutadas por interrupción se tienen en cuenta en los contadores de estas funciones y por lo tanto no se tienen en cuenta en el contador de la función de llamada o interrumpida.

El equilibrado de todos los posibles recorridos de ejecución en la función debe realizarse para obtener el mismo valor del contador al final de la función. Este equilibrio es complicado debido al hecho de que las instrucciones no son intercambiables. En consecuencia, a menudo será necesario añadir una o más instrucciones cuyo único propósito sea equilibrar una rama con respecto a otra. Esta solución aumenta el tamaño del código, en mayor o menor medida, debido al reequilibrio de las ramas.

Otra solución consiste en definir clases de instrucción que tengan el mismo valor de incremento. Estas clases con el mismo incremento incluyen instrucciones similares, por ejemplo la clase de ramas condicionales, la clase de operaciones aritméticas y lógicas, la clase de saltos y las llamadas de función. La instrucción NOP tendrá valor de incremento 0; durante un ataque que convertiría las instrucciones del programa en NOP, el contador no se incrementará, lo que significa que el ataque será detectado. Este mecanismo puede utilizarse para verificar que cada instrucción de la trayectoria de ejecución dentro de una función se ejecuta correctamente.

Agrupar instrucciones en clases con el mismo valor de incremento facilita el equilibrado de las ramas: si cada rama realiza operaciones diferentes pero similares, pueden usar instrucciones que tengan valores de incremento iguales. Además, el equilibrado puede llevarse a cabo utilizando instrucciones que son diferentes pero que pertenecen a la misma clase que las utilizadas en otra rama. Por ejemplo, una rama condicional en una rama puede ser equilibrada con una rama incondicional a la siguiente instrucción en otra rama. Los parámetros de instrucción no se tienen en cuenta al incrementar el contador, lo que también facilita el equilibrado de las ramas. Las ramas también pueden ser equilibradas por el acceso directo al contador de funciones si hay una gran diferencia.

La comprobación del valor del contador de funciones, realizada al salir de la función o durante comprobaciones intermedias, puede realizarse por software leyendo el valor del contador de funciones actual, incrementado por el procesador durante la ejecución y comparándolo con el valor precalculado. Esta comprobación también puede ser realizada por el hardware suministrando al procesador el valor de comprobación precalculado. Este valor se puede suministrar como parámetro de una instrucción de retorno de llamada modificada o de una instrucción añadida especialmente al conjunto de instrucciones del procesador. La instrucción realiza la comprobación comparando el valor precalculado suministrado como argumento con el valor del contador de funciones actual.

La asignación del contador de funciones en una estructura de datos y la inicialización de este contador a 0 o a otro valor específico puede llevarse a cabo por software o hardware por el procesador en una instrucción de llamada de función modificada.

Según una variante, el método consiste en inicializar al comienzo de cada rutina un indicador en el contador de rutina después de guardar su valor anterior. Al salir de la rutina, el indicador se reestablece a su valor anterior (la dirección del contador de función de llamada). Este mecanismo puede utilizarse para realizar verificaciones intermedias del contador durante la rutina y especialmente antes de una operación sensible. Si el valor del contador no es el valor esperado en este punto en la ejecución de la rutina, esto significa que la ejecución de la rutina ha sido perturbada.

Asignar el contador de funciones en la pila del procesador es útil si la pila sólo se utiliza para guardar las direcciones de retorno y posiblemente algunos registros durante las llamadas de función. En este caso, el contador permanece en la parte superior de la pila en el cuerpo de la función y, por lo tanto, resulta innecesario mantener un indicador para este contador con el fin de realizar comprobaciones durante la rutina. En otro modo de realización, los contadores de rutina se pueden asignar en una estructura de datos distinta a la pila del procesador (por ejemplo, una pila de compiladores) con un medio para encontrar el contador de la rutina que se está ejecutando (posiblemente un indicador de pila).

Un modo de realización del método ilustrado en la figura 7 consiste en utilizar un contador global (CP en la figura 7) del procesador para realizar el recuento. El incremento cuando se ejecutan las instrucciones es llevado a cabo por el procesador en este contador global (CP). En cada llamada de función o tratamiento de interrupciones, el valor del contador global se añade al contador de funciones de llamada o interrupciones y se restablece a 0 ("CP añadido al valor en la pila CF=CF+CP y CP restablecido" en la figura). Durante un retorno de función o retorno de interrupción, el valor final del contador de funciones se obtiene añadiendo el valor del contador global antes de la comprobación, entonces el contador global se restablece a 0 para la función de llamada o interrupción.

Una ventaja del método de acuerdo con la forma de realización ilustrada en las figuras 5 a 7, denominada método de recuento de comprobación, radica en el hecho de que el recuento de comprobación es específico para cada rutina. Sólo se tienen en cuenta las instrucciones de la rutina, no se tienen en cuenta las rutinas de llamada y en particular las rutinas aleatorias o de desincronización, lo que facilita el equilibrado de ramas, ya que ésto solo tiene que realizarse dentro del cuerpo de cada rutina. Además, el método puede limitarse a aquellas funciones en las que se considera necesario, lo que reduce aún más la escala de la tarea de equilibrado.

Agrupar instrucciones en clases con el mismo valor de incremento facilita el equilibrado de ramas. Los parámetros de instrucción no se tienen en cuenta al incrementar el contador, lo que facilita también el equilibrado de ramas.

El método se puede implementar sin modificar el conjunto de instrucciones del procesador: incluso se puede implementar sin modificar el procesador, por lo tanto en los procesadores actuales.

La verificación del recuento de comprobaciones puede realizarse al final de la rutina solamente, limitando de este modo el incremento en el tamaño del código.

REIVINDICACIONES

1. Método para asegurar la ejecución de un programa en un conjunto electrónico que incluye medios de procesamiento de datos y medios de almacenamiento de datos contra ataques por radiación, flash, luz, láser ó fallo, **caracterizado porque** consiste en comprobar la ejecución de cada instrucción de al menos una parte de dicho programa realizando durante la ejecución de dicha parte un cálculo que utiliza valores predeterminados, dependiendo de o asociado con cada una de dichas instrucciones y comparando el resultado obtenido con un valor precalculado, y activando una acción de detección de anomalía si se encuentra una diferencia.

2. Método según la reivindicación 1, **caracterizado porque** los valores que dependen de cada una de dichas instrucciones son valores que dependen del contenido, tipo, función, resultado y/o de cualquier característica asociada a dichas instrucciones como tales.

3. Método según la reivindicación 1 ó 2, **caracterizado porque** consiste en realizar el cálculo de una suma de comprobación sobre el contenido de dichas instrucciones.

4. Método según la reivindicaciones 1 a 2, **caracterizado porque** consiste en incrementar un contador al ejecutar dicha parte, por un valor dependiente de o asociado a cada una de dichas instrucciones ejecutadas.

5. Método según una de las reivindicaciones 1 a 4, **caracterizado porque** consiste en definir clases de instrucción para las cuales el valor asociado a cada una de dichas instrucciones de dicha clase es idéntico.

6. Tarjeta inteligente que comprende un módulo electrónico que incluye medios de procesamiento de datos y medios de almacenamiento de datos que contienen un programa a ejecutar, **caracterizada porque** los medios de procesamiento incluyen medios para comprobar la ejecución de cada instrucción en al menos una parte de dicho programa realizando durante la ejecución de dicha parte un cálculo usando valores predeterminados, dependiendo de o asociado con cada una de dichas instrucciones, comparando el resultado obtenido con un valor precalculado, y activando una acción de detección de anomalía si se encuentra una diferencia.

7. Tarjeta inteligente según la reivindicación 6, **caracterizada porque** dichos medios comprenden un procesador que tiene instrucciones especiales o instrucciones modificadas de tipo conocido para tener en cuenta el valor precalculado para cada nueva porción de código y realizar la comparación.

8. Tarjeta inteligente según la reivindicación 6, **caracterizada porque** el cálculo realizado se pone a disposición de un módulo de software que realiza la comparación.

9. Tarjeta inteligente según una de las reivindicaciones 6 a 8, **caracterizada porque** dichos medios comprenden un contador asociado a dicha porción.

10. Programa de ordenador que comprende instrucciones de código de programa para ejecutar los pasos del método según una de las reivindicaciones 1 a 5 cuando dicho programa se ejecuta en un conjunto electrónico.

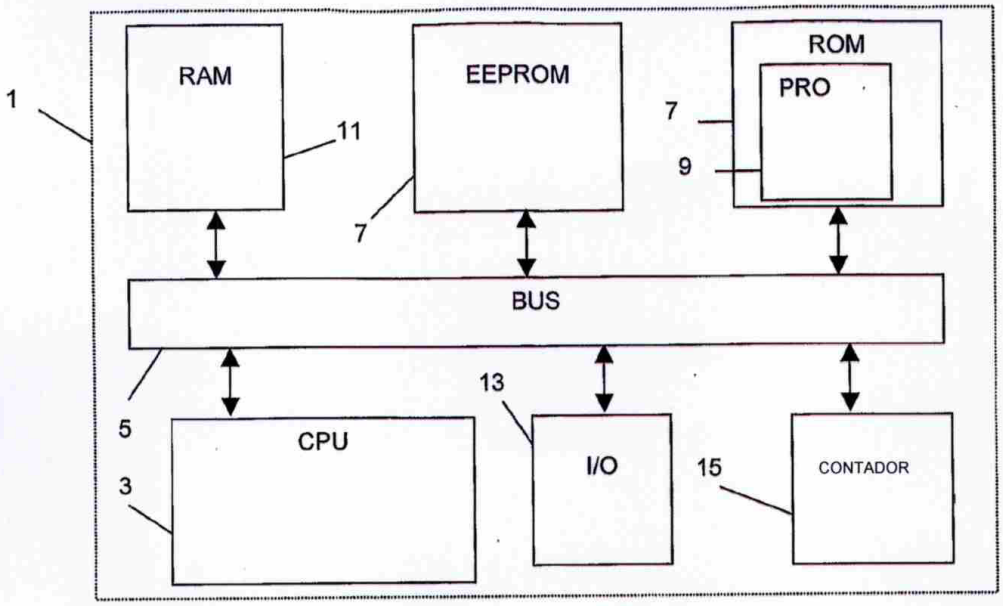
1
RAM
11
EEPROM
7
ROM
PRO
7
9
BUS
5
13
CPU
3
I/O
15
CONTADOR

FIG. 1

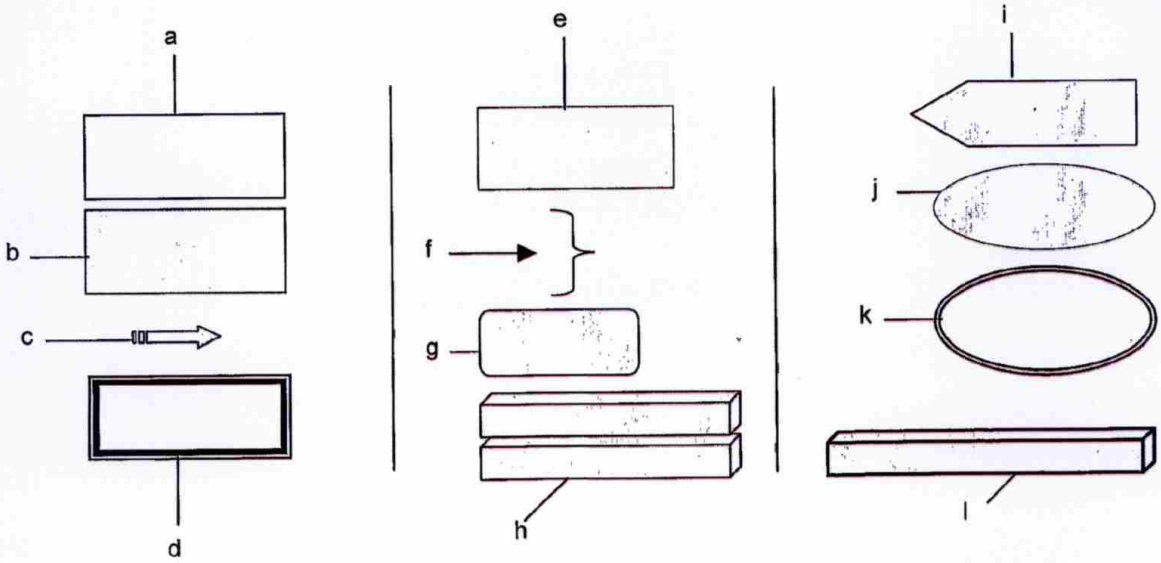
a
b
c
d
e
f
g
h
i
j
k
l

FIG. 2

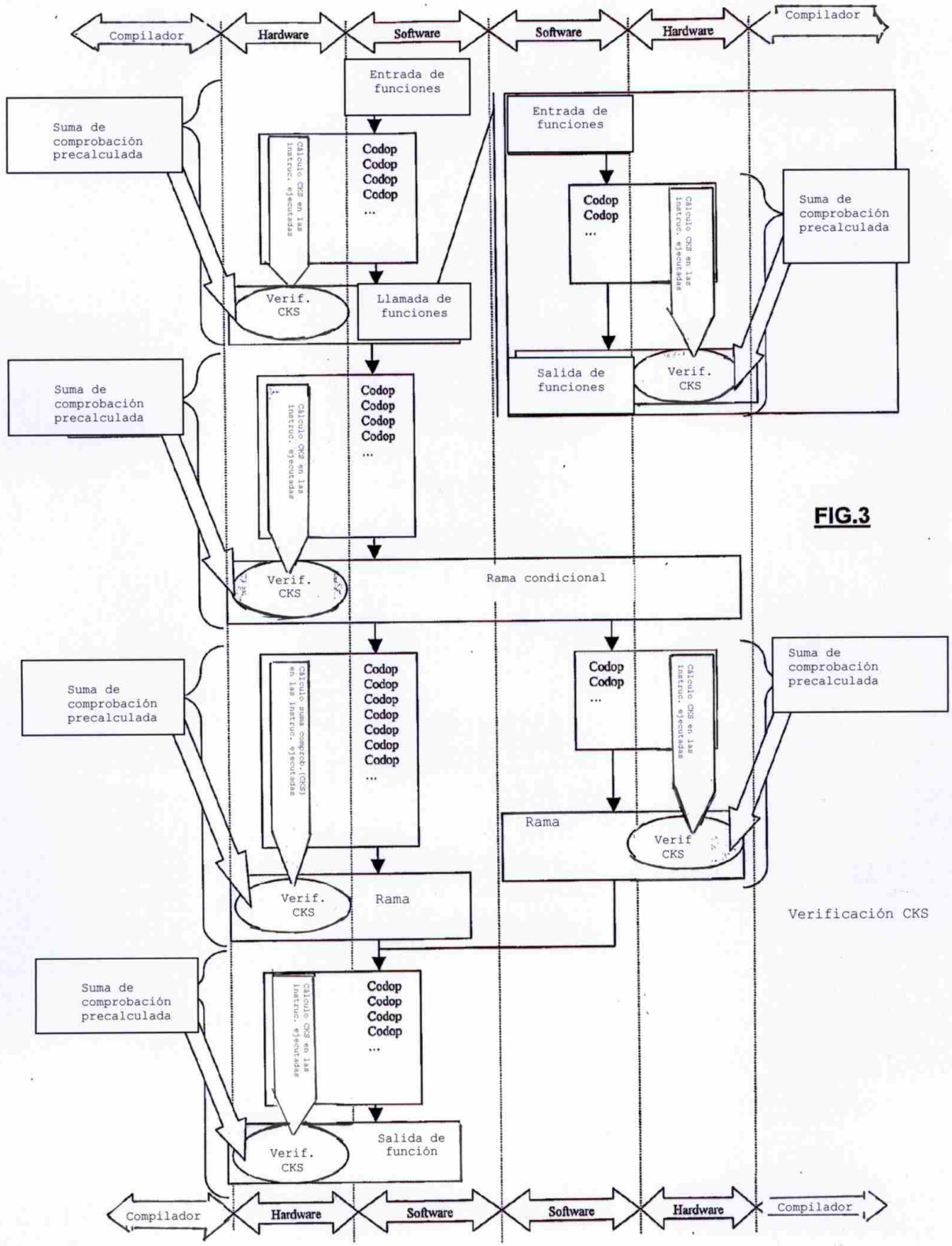

FIG.3

Verificación CKS

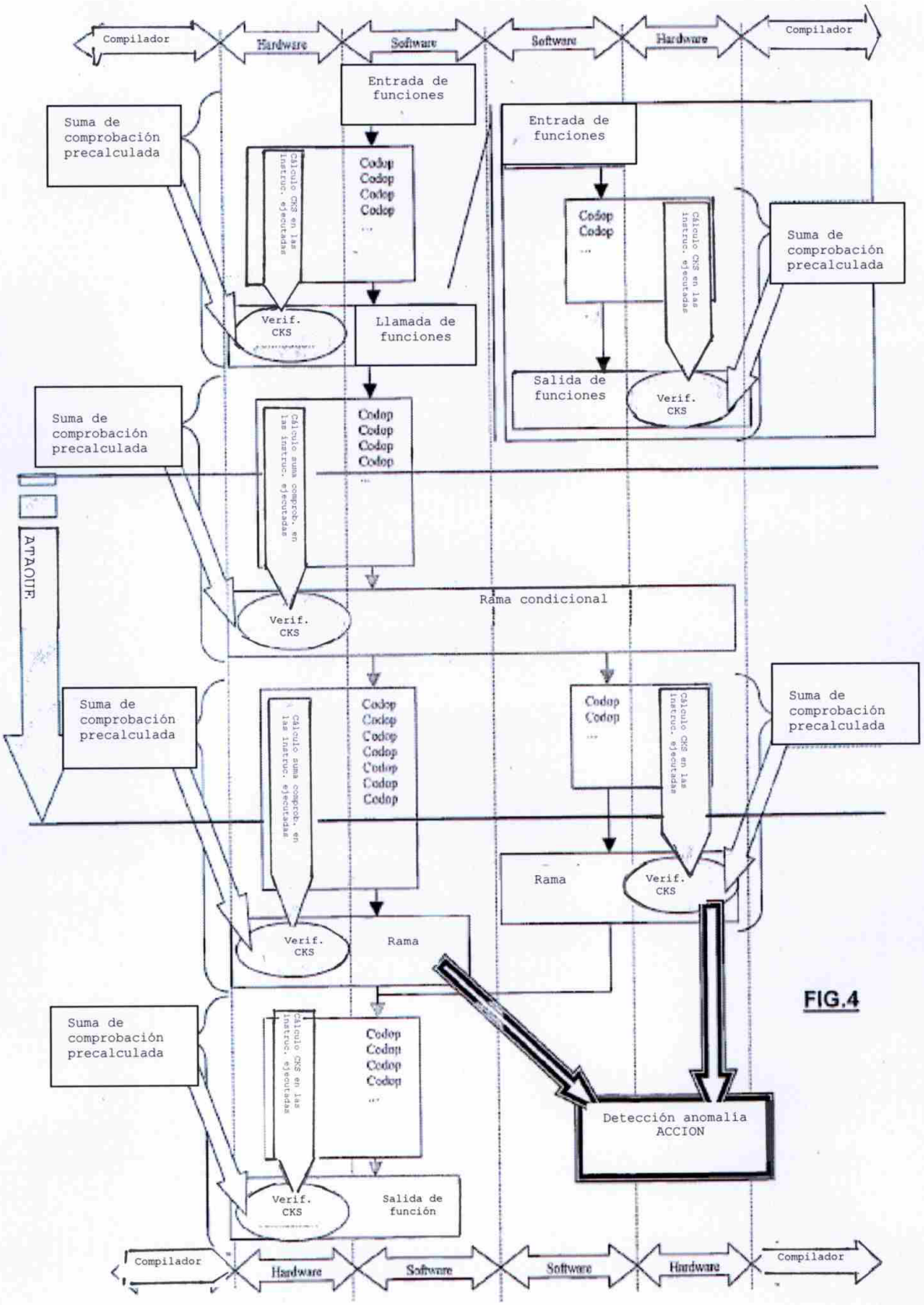
Compilador
Hardware
Software
Software
Hardware
Compilador
Entrada de funciones
Suma de comprobación precalculada
Cálculo CKS en las instruc. ejecutadas
Codop
Verif. CKS
Llamada de funciones
Entrada de funciones
Salida de funciones
Suma de comprobación precalculada
Cálculo suma comprob. en las instruc. ejecutadas
ATAQUE
Rama condicional
Rama
Detección anomalía ACCION
Salida de función

FIG.4

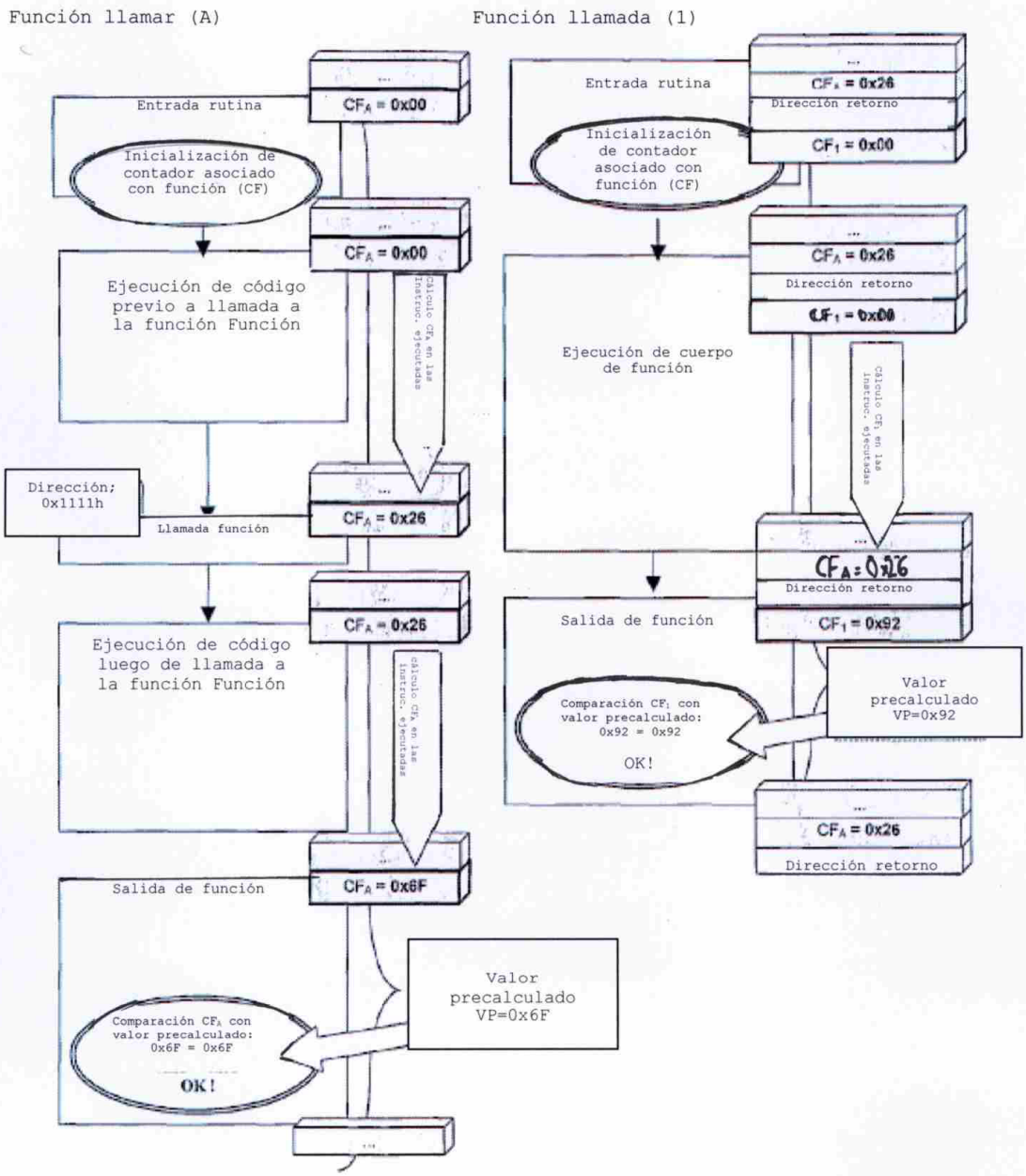
Función llamar (A)
Función llamada (1)
Entrada rutina
Inicialización de contador asociado con función (CF)
Ejecución de código previo a llamada a la función Función
Dirección; 0x1111h
Llamada función
Ejecución de código luego de llamada a la función Función
Salida de función
Comparación CFA con valor precalculado: 0x6F = 0x6F
OK!
Valor precalculado VP=0x6F
Dirección retorno
Ejecución de cuerpo de función
Comparación CF1 con valor precalculado: 0x92 = 0x92
OK!
Valor precalculado VP=0x92

FIG. 5

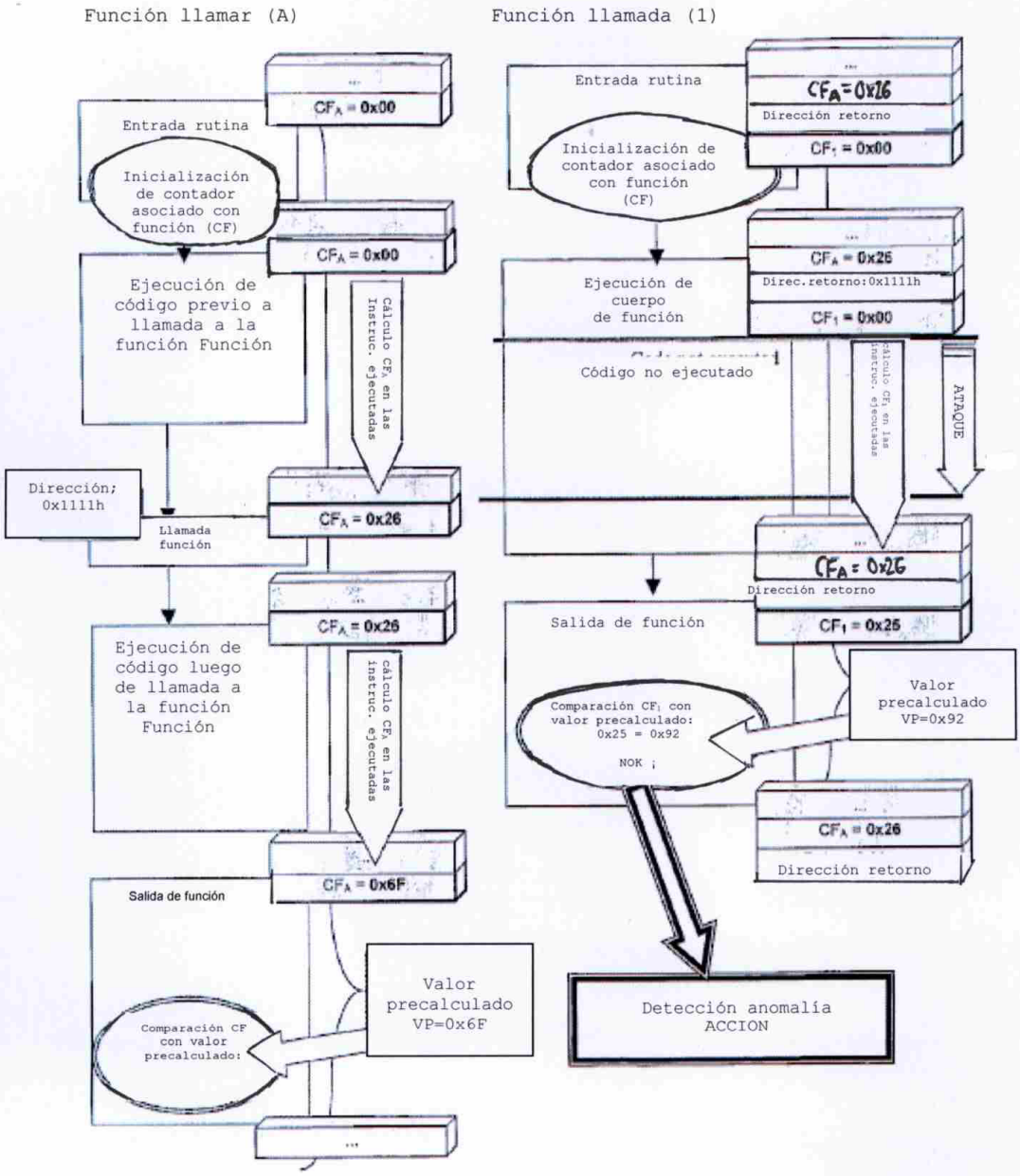
Función llamar (A)
Función llamada (1)
Entrada rutina
Inicialización de contador asociado con función (CF)
Ejecución de código previo a llamada a la función Función
Dirección; 0x1111h
Llamada función
Ejecución de código luego de llamada a la función Función
Salida de función
Cálculo CFA en las Instruc. ejecutadas
Comparación CF con valor precalculado:
Valor precalculado VP=0x6F
Dirección retorno
Ejecución de cuerpo de función
Código no ejecutado
Direc.retorno:0x1111h
Cálculo CF1 en las instruc. ejecutadas
ATAQUE
Salida de función
Comparación CF1 con valor precalculado: 0x25 = 0x92
NOK ¡
Valor precalculado VP=0x92
Detección anomalía ACCION

FIG. 6

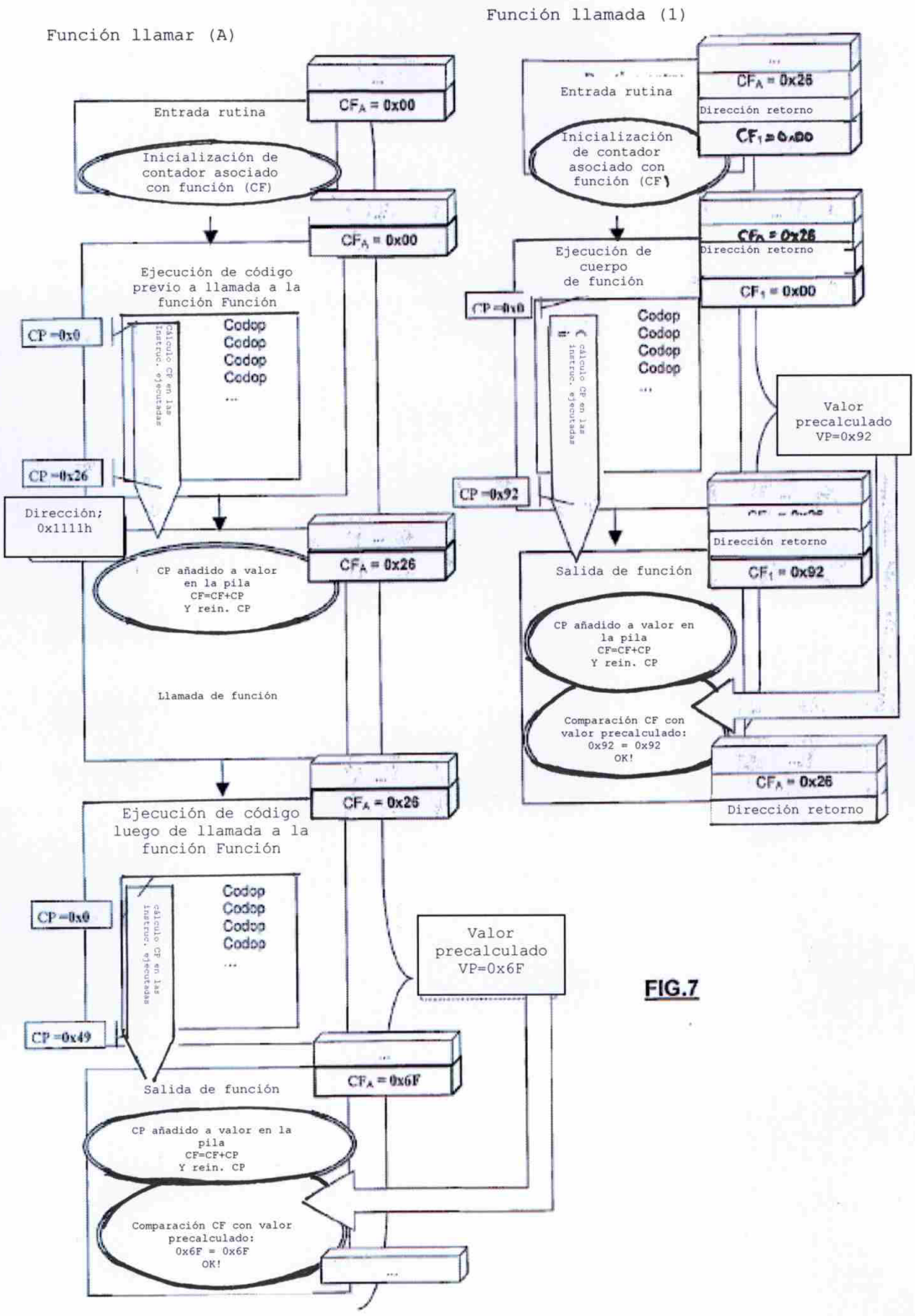
Función llamar (A)
Función llamada (1)
Entrada rutina
Inicialización de contador asociado con función (CF)
Ejecución de código previo a llamada a la función Función
CP =0x0
Codop
Codop
Codop
Codop
CP =0x26
Dirección; 0x1111h
CP añadido a valor en la pila CF=CF+CP Y rein. CP
Llamada de función
Ejecución de código luego de llamada a la función Función
CP =0x0
Codop
Codop
Codop
Codop
CP =0x49
Salida de función
CP añadido a valor en la pila CF=CF+CP Y rein. CP
Comparación CF con valor precalculado: 0x6F = 0x6F OK!
Valor precalculado VP=0x6F
Entrada rutina
Inicialización de contador asociado con función (CF)
Ejecución de cuerpo de función
CP =0x0
Codop
Codop
Codop
Codop
CP =0x92
Salida de función
CP añadido a valor en la pila CF=CF+CP Y rein. CP
Comparación CF con valor precalculado: 0x92 = 0x92 OK!
Valor precalculado VP=0x92
Dirección retorno
CF1 = 0x92
CFA = 0x26
Dirección retorno
CF1 = 0x00
CFA = 0x00
CFA = 0x6F

FIG.7