(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 442 842**

(51) Int. CI.:

G06F 9/45 (2006.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(96) Fecha de presentación y número de la solicitud europea: **11.04.2003 E 03717297 (0)**

(97) Fecha y número de publicación de la concesión europea: **13.11.2013 EP 1497722**

(54) Título: **Optimización de código de programa generado por compilador**

(30) Prioridad:

15.04.2002 DE 10216602

(45) Fecha de publicación y mención en BOPI de la traducción de la patente:
13.02.2014

(73) Titular/es:

GIESECKE & DEVRIENT GMBH (100.0%)
PRINZREGENTENSTRASSE 159
81677 MÜNCHEN, DE

(72) Inventor/es:

BALDISCHWEILER, MICHAEL y
NESS, WERNER

(74) Agente/Representante:

TORNER LASALLE, Elisabet

DESCRIPCIÓN

Optimización de código de programa generado por compilador.

La invención se refiere a la programación de soportes de datos portátiles así como a la ejecución de programa por soportes de datos portátiles. Un soporte de datos portátil en el sentido del presente documento puede ser en particular una tarjeta chip (*smart card*, tarjeta inteligente) en diferentes formas de construcción o un módulo chip.

Los soportes de datos portátiles, tal como son habituales en la actualidad, presentan un núcleo de procesador y varias memorias fabricadas en diferentes tecnologías. En una configuración típica están previstas, por ejemplo, una ROM programada por máscara, una EEPROM programable y borrable eléctricamente y una RAM escribible. La RAM sirve como memoria de trabajo durante la ejecución del programa, mientras que el código de programa que va a ejecutarse por el núcleo de procesador puede estar almacenado en la ROM y/o en la EEPROM. En el apartado 3.4 del libro “Handbuch der Chipkarten” de W. Rankl y W. Effing, Hanser Verlag, 3ª edición 1999, se describen éstas configuraciones y configuraciones similares de soportes de datos portátiles.

Normalmente una celda de memoria en la EEPROM ocupa aproximadamente cuatro veces la superficie de chip de una celda de memoria de ROM. Por tanto, por motivos de ahorro de superficie o de mayor capacidad de memoria disponible con la misma superficie es deseable disponer el código de programa que va a ejecutarse en su mayor parte posible en la ROM. No obstante, el contenido de la ROM programada por máscara debe fijarse ya de manera invariable en la fabricación para grandes números de unidades del soporte de datos. Por el contrario, la EEPROM sólo se escribe al completar e inicializar una serie de soportes de datos o al personalizar los soportes de datos individuales. Por ello, debido a la mayor flexibilidad es ventajoso un almacenamiento del código de programa ejecutable en su mayor parte posible en la EEPROM. Esto se refiere tanto a la programación de menores números de unidades de soportes de datos como a la corrección de errores y la introducción de funciones adicionales en el caso de grandes series.

Por tanto, en la programación de soportes de datos portátiles, por un lado existe el problema de aprovechar la ROM programada por máscara o una memoria comparable en su mayor parte posible y, por otro lado, conseguir una flexibilidad lo más elevada posible para modificaciones de programa y/o para la fabricación de soportes de datos en menores números de unidades.

Según la invención, este problema se soluciona en su totalidad o en parte mediante un procedimiento con las características de la reivindicación 1, un producto de programa informático según la reivindicación 8 y un soporte de datos portátil según la reivindicación 10. Las reivindicaciones dependientes definen configuraciones preferidas de la invención. El orden de enumeración de las etapas en las reivindicaciones de procedimiento no debe entenderse como limitación del alcance de protección. Más bien están previstas configuraciones de la invención en las que estas etapas se realizan en otro orden o total o parcialmente en paralelo o total o parcialmente combinadas.

La invención parte de la idea básica de utilizar, para la optimización del código de programa, una biblioteca predefinida que contiene una pluralidad de fragmentos de código de biblioteca. En la operación de optimización según la invención, en el código de programa que va a optimizarse se buscan a su vez fragmentos de código de programa que, con respecto a su acción o función, corresponden en cada caso a un fragmento de código de biblioteca. Tales fragmentos de código de programa se sustituyen en cada caso por una llamada del fragmento de código de biblioteca correspondiente. El código de programa optimizado se almacena en una primera zona de almacenamiento del soporte de datos (por ejemplo, en la EEPROM), mientras que la biblioteca está prevista para su almacenamiento en una segunda zona de almacenamiento (por ejemplo, en la ROM).

La operación de optimización según la invención llevó, en las pruebas llevadas a cabo por los inventores, a una reducción considerable del tamaño del código de programa previsto para la primera zona de almacenamiento. Este resultado es sorprendente porque de manera intuitiva se supondría que con un volumen realista de la biblioteca sólo se obtendrían unas pocas coincidencias de partes del código de programa con los fragmentos de código de biblioteca.

La reducción del tamaño de código provocada por la invención tiene como consecuencia que con un soporte de datos con un equipamiento de memoria predeterminado pueden recibirse códigos de programa para funciones adicionales en la primera zona de almacenamiento. En caso de que la primera zona de almacenamiento esté configurada como EEPROM o en una tecnología comparable, entonces este código de programa sólo tiene que cargarse al completar o inicializar o personalizar el soporte de datos. Por tanto, una modificación o nueva generación del código de programa, que por su compacidad implementa un gran número de funciones, en primer lugar es rápida y en segundo lugar ya es posible para pequeños números de unidades de soportes de datos o incluso para soportes de datos individuales.

La biblioteca predefinida se encuentra según la invención en la segunda zona de almacenamiento, esto es, por ejemplo, en la ROM programada por máscara. Por regla general, el ahorro de código de programa conseguido mediante la optimización según la invención es menor que el tamaño de la biblioteca. Sin embargo, también en este

caso es ventajosa la utilización de la invención, porque la primera zona de almacenamiento valiosa se aprovecha mejor. En caso de que en el código de programa generado por compilador se encuentren muchos fragmentos de código que en cada caso pueden sustituirse por grupos por en cada caso un único fragmento de código de la biblioteca, y en caso de que la biblioteca sólo contenga unos pocos fragmentos de código no necesarios, el código de programa, mediante la optimización, puede disminuir incluso en más de la longitud de biblioteca. En este caso, el uso de la invención es ventajoso incluso cuando la primera y la segunda zona de almacenamiento sólo son segmentos conceptuales del mismo campo de memoria físico.

Según la invención, para la optimización se buscan fragmentos de código de programa, es decir, segmentos en el código de programa generado por compilador que pueden sustituirse por fragmentos de código de biblioteca correspondientes. En la creación del programa, el programador no tiene que tener en cuenta esta operación de optimización posterior; en particular, no es necesario que prevea en el programa llamadas de rutinas de biblioteca. Por tanto, el desarrollo del programa no se ve dificultado de ningún modo por la invención.

En la terminología utilizada en el presente documento, los términos “código de programa” o “fragmento de código” se referirán tanto a código de máquina ejecutable antes o después del enlace como al código fuente de ensamblador correspondiente. Dicho de otro modo, en diferentes configuraciones de la invención, la operación de optimización según la invención puede llevarse a cabo tanto basándose en el código fuente de ensamblador generado por compilador como basándose en el código de máquina ya ensamblado. En el caso mencionado en primer lugar, el ensamblaje y dado el caso el enlace sólo se producen tras la optimización. La biblioteca, durante la optimización, también puede estar presente como código fuente de ensamblador y/o como código de máquina ya ensamblado.

En general, una sustitución de un fragmento de código de programa por un fragmento de código de biblioteca siempre es posible cuando ambos fragmentos de código ejecutan funciones mutuamente correspondientes. En este caso pueden llevarse a cabo cálculos complejos con respecto a las acciones precisas de los fragmentos de código para, por ejemplo, activar una operación de sustitución también cuando instrucciones individuales en los fragmentos de código están cambiadas de manera no perjudicial. Por el contrario, en ejemplos de realización especialmente sencillos, sólo se lleva a cabo una sustitución cuando los fragmentos de código son idénticos con respecto al código de máquina definido por los mismos. Sin embargo, también en esta configuración sencilla es necesario un cierto análisis de los fragmentos de código, porque por regla general no debe sustituirse por ejemplo un fragmento de código que presenta un salto con un destino de salto que no está situado en el fragmento de código.

Se obtienen posibilidades de sustitución adicionales cuando se utilizan fragmentos de código parametrizados que, de manera similar a una llamada de procedimiento, contienen uno o varios parámetros (por ejemplo direcciones de memoria o valores numéricos).

Preferiblemente, por regla general, la llamada de un fragmento de código de biblioteca se produce por una instrucción de llamada de subrutina introducida en el código de programa. Entonces en la biblioteca está prevista una instrucción de retorno que sigue directamente al fragmento de código de biblioteca. En algunas formas de realización pueden ser válidas excepciones a esta regla, cuando el fragmento de código que va a sustituirse interviene en el flujo de programa. En caso de que, por ejemplo, el fragmento de código termine con una instrucción de retorno de subrutina, entonces por regla general la llamada puede producirse por medio de una instrucción de salto.

Según la invención, la biblioteca utilizada está predefinida, es decir, no depende del código de programa procesado en la ejecución de optimización actual. Sin embargo, para conseguir los mejores resultados de optimización posibles, la biblioteca está configurada preferiblemente de modo que contiene entradas adecuadas para estructuras del código de programa que aparecen con frecuencia. Tales segmentos de código que aparecen con frecuencia pueden depender en particular del hardware y/o un sistema operativo del soporte de datos y/o de un compilador utilizado en la generación del código de programa generado por compilador.

El producto de programa informático previsto según la invención puede ser en particular un soporte de datos legible por ordenador tal como, por ejemplo, un medio de almacenamiento electrónico o magnético u óptico, aunque no se limita a soportes de datos físicos. En el sentido utilizado en el presente documento, como producto de programa informático también se entenderán señales eléctricas u ópticas (por ejemplo nivel de tensión de una conexión de comunicación). El producto de programa informático contiene código de programa que ejecuta las etapas de optimización según la invención. Preferiblemente el producto de programa informático contiene además un compilador y/o un ensamblador y/o un programa de enlace y/o un programa de carga.

El producto de programa informático según la invención y el soporte de datos portátil según la invención están perfeccionados preferiblemente con características que corresponden a las características descritas anteriormente y/o mencionadas en las reivindicaciones de procedimiento.

Características, objetivos y ventajas adicionales de la invención se deducen a partir de la siguiente descripción de un ejemplo de realización y varias alternativas de realización. Se remite al dibujo esquemático, en el que la única figura

(figura 1) muestra una representación de un soporte de datos portátil así como diferentes versiones del código de programa en un ejemplo de realización de la invención.

La invención se utiliza en la programación de un soporte 10 de datos portátil, que en el ejemplo de realización descrito en este caso está configurado como tarjeta chip. De una manera en sí conocida, el soporte 10 de datos contiene un chip de semiconductor con un núcleo 12 de procesador, una ROM 14 programada por máscara, una EEPROM 16, una RAM 18 y una interfaz 20 para la comunicación sin contacto o con contacto. Los componentes mencionados están conectados entre sí a través de un bus 22. En alternativas de realización, los tres campos 14, 16, 18 de memoria pueden estar configurados en otras tecnologías; en particular puede utilizarse la tecnología *flash* para la ROM 14 y/o la EEPROM 16.

En los campos 14, 16, 18 de memoria están previstas de manera conceptual una primera y una segunda zona 24, 26 de almacenamiento. La primera zona 24 de almacenamiento sirve para recibir el código de programa optimizado en forma de código de máquina ejecutable. En la segunda zona 26 de almacenamiento se almacena una biblioteca 28 predefinida igualmente en forma de código de máquina ejecutable. La primera zona 24 de almacenamiento se encuentra en el ejemplo de realización descrito en este caso en la EEPROM 16, y la segunda zona 26 de almacenamiento se encuentra en la ROM 14. De una manera en sí conocida, la ROM 14 contiene además de la segunda zona 26 de almacenamiento otras rutinas predeterminadas de manera fija, que por ejemplo forman un sistema operativo del soporte 10 de datos. La EEPROM 16 contiene además un sistema de ficheros para datos, que se almacenarán de manera no volátil en el soporte 10 de datos.

La biblioteca 28 presenta un gran número de fragmentos 30A, 30B, 30C, ... de código de biblioteca predefinidos, que a continuación en general se designan con 30x. En la figura 1, los fragmentos 30x de código de biblioteca, por motivos de una representación más clara, se muestran como código fuente de ensamblador. Por regla general, a cada fragmento 30x de código de biblioteca le sigue directamente una instrucción 32A, 32B, ... de retorno de subrutina (a continuación en general denominada 32x). Sin embargo, puede omitirse la instrucción 32x de retorno de subrutina, cuando no puede conseguirse en la ejecución del fragmento 30x de código de biblioteca, porque por ejemplo cada ejecución del programa del fragmento 30x de código de biblioteca termina en un punto de salida o en un retorno de subrutina contenido en el fragmento 30x de código de biblioteca.

El desarrollo del programa para el soporte 10 de datos portátil parte de un código 34 fuente de alto nivel, que en la figura 1 se representa a modo de ejemplo en el lenguaje de programación C. El fragmento mostrado en la figura 1 espera a que el tercer bit desde la posición de las unidades del registro de entrada INPORT adopte el valor “0”, y a continuación pone el registro de salida OUTPORT al valor hexadecimal “FF”. Un compilador 36 en sí conocido convierte el código 34 fuente de alto nivel en código 38 de programa generado por compilador, que en la figura 1 se representa en forma de código fuente de ensamblador para el conjunto de instrucciones 6805. En alternativas de realización se prevén otros conjuntos de instrucciones, en cada caso conforme al núcleo 12 de procesador.

Un programa 40 de optimización ejecuta las etapas de optimización esenciales para el presente ejemplo de realización. El programa 40 de optimización procesa el código 38 de programa generado por compilador y además accede a información sobre los fragmentos 30x de código de biblioteca contenidos en la biblioteca 28. En diferentes variantes de realización, en esta información puede estar contenida por ejemplo una copia de la biblioteca 28 en el código fuente de ensamblador y/o una copia de la biblioteca 28 en el código de máquina ejecutable y/o una especificación de la acción de los fragmentos 30x de código de biblioteca individuales en un lenguaje de descripción adecuado. Además puede estar prevista información adicional como por ejemplo índices o tablas *hash*, para acelerar las operaciones de búsqueda llevadas a cabo por el programa 40 de optimización.

El programa 40 de optimización identifica los fragmentos 42 de código de programa contenidos en el código 38 de programa generado por compilador, que en la ejecución mediante el núcleo 12 de procesador presentan una función idéntica a los fragmentos 30x de código de biblioteca contenidos en la biblioteca 28. Para ello, en el presente ejemplo de realización se utiliza un procedimiento relativamente sencillo, en el que el código 38 de programa generado por compilador se compara a nivel de texto fuente de ensamblador con las entradas individuales en la biblioteca 28. Con respecto al comando abreviado y las indicaciones de dirección y de valor a este respecto puede tener lugar una comparación textual. Por el contrario, los destinos de salto simbólicos deben transformarse antes de la comparación en una forma estandarizada o en un valor relativo numérico. Por el contrario, en alternativas de realización la optimización puede producirse basándose en un código 38 de programa generado por compilador, que ya está presente en forma de código de máquina ensamblado.

Un fragmento 42 de código de programa, para el que en la operación de comparación se encontró un fragmento 30x de código de biblioteca correspondiente, se sustituye en la operación de optimización por una llamada de este fragmento 30x de código de biblioteca. En la figura 1 por ejemplo el fragmento 42 de código de programa y el fragmento 30B de código de biblioteca son idénticos a excepción de la denominación simbólica del destino de salto. Por tanto, el programa 40 de optimización sustituye este fragmento 42 de código de programa en el código 44 de programa optimizado por una llamada del fragmento 30B de código de biblioteca. En el presente ejemplo, esta llamada está configurada como instrucción 46 de llamada de subrutina. Como el fragmento 42 de código de programa en el presente ejemplo corresponde a un código de máquina de siete bytes de longitud y la instrucción 46

de llamada de subrutina sólo requiere tres bytes, mediante la sustitución se redujo considerablemente el espacio de memoria necesario para el código 44 de programa optimizado.

Tras finalizar la optimización, el código 44 de programa optimizado se convierte mediante un ensamblador 48 en código de máquina ejecutable por el núcleo 12 de procesador. Después de una operación de enlace, dado el caso necesaria, con partes de programa adicionales, al completar o inicializar o personalizar el soporte 10 de datos se carga el código en la primera zona 24 de almacenamiento. La biblioteca 28 ya se encuentra desde la fabricación del chip del soporte 10 de datos en la segunda zona 26 de almacenamiento. Así, el soporte 10 de datos está listo para su uso. Las etapas de conversión, optimización y ensamblaje descritas anteriormente se llevan a cabo por un ordenador de propósito general (no mostrado en la figura 1), que ejecuta el compilador 36, el programa 40 de optimización y el ensamblador 48.

Cuando durante el funcionamiento del soporte 10 de datos, la ejecución de programa por el núcleo 12 de procesador en la primera zona 24 de almacenamiento llega al punto de la instrucción 46 de llamada de subrutina, se ejecuta el fragmento 30B de código de biblioteca en la segunda zona 26 de almacenamiento como subrutina. En su acción, las instrucciones ejecutadas corresponden exactamente al fragmento 42 de código de programa eliminado en la optimización. Tras la ejecución de estas instrucciones, el núcleo 12 de procesador ejecuta un retorno activado por la instrucción 32B de retorno de subrutina a la instrucción en la primera zona 24 de almacenamiento que sigue directamente a la instrucción 46 de llamada de subrutina.

En la optimización ha de prestarse atención de que no se modifiquen las funciones de programa. Así, por ejemplo, los fragmentos 42 de código de programa con instrucciones de salto, que posiblemente presentan un destino de salto situado fuera del fragmento 42 de código de programa, sólo deberían sustituirse tras un análisis preciso. Se permite una sustitución cuando cada ejecución posible del fragmento 42 de código de programa termina con un punto de salida o un retorno de subrutina. Sin embargo, en estos casos la llamada del fragmento 30x de código de biblioteca correspondiente no se produce con una instrucción de llamada de subrutina, sino con una instrucción de salto normal. Estas consideraciones también pueden producirse ya en la creación de la biblioteca 28, de modo que ésta sólo contenga aquellos fragmentos 30x de código de biblioteca, cuyo uso se permite sin condiciones secundarias adicionales.

La biblioteca 28 debería construirse de modo que proporcione fragmentos 30x de código de biblioteca adecuados con la mayor frecuencia posible y así ofrezca el mayor número posible de posibilidades de optimización. Así, el fragmento 30B de código de biblioteca de la figura 1 está adaptado por ejemplo a las propiedades de hardware del soporte 10 de datos. Cuando el bit de entrada consultado en este fragmento 30B de código de biblioteca corresponde a un valor de señal necesario a menudo entonces ha de partirse de la base de que los fragmentos 42 de código de programa correspondientes siempre se encontrarán en el código 42 de programa generado por compilador para las más diferentes aplicaciones del soporte 10 de datos. De manera similar, llamadas de sistema operativo frecuentes pueden cubrirse por fragmentos 30x de código de biblioteca correspondientes. Se obtiene una fuente adicional para fragmentos de código que se repiten en el código 38 de programa generado por compilador por el hecho de que la generación de código en el compilador 36 se produce esquemáticamente y por tanto se generan estructuras de código repetitivas.

Por tanto, en general para la generación de la biblioteca 28 es ventajoso evaluar estadísticamente el código 38 de programa generado por el compilador 36 para una pluralidad de aplicaciones que están previstas para el hardware y el sistema operativo del soporte 10 de datos.

REIVINDICACIONES

1. Procedimiento para la optimización de código (38) de programa generado por compilador, que está previsto para un soporte (10) de datos portátil con un núcleo (12) de procesador así como una primera (24) y una segunda zona (26) de almacenamiento,

- en el que la primera zona (24) de almacenamiento está prevista para recibir el código (44) de programa optimizado,

- en el que la segunda zona (26) de almacenamiento está prevista para recibir una biblioteca (28) con una pluralidad de fragmentos (30x) de código de biblioteca,

- en el que en el código (38) de programa generado por compilador se buscan fragmentos (42) de código de programa que, al menos con respecto a su acción, corresponden en cada caso a un fragmento (30x) de código de biblioteca, en el que los fragmentos (42) de código de programa encontrados de este modo se sustituyen por en cada caso una llamada del fragmento (30x) de código de biblioteca correspondiente,

caracterizado porque

la biblioteca está predefinida, de modo que la biblioteca no depende del código de programa procesado en la ejecución de optimización actual y se fija de manera invariable para la segunda zona (26) de almacenamiento, antes de que se optimice el código de programa generado por compilador.

2. Procedimiento según la reivindicación 1, caracterizado porque se sustituye un fragmento (42) de código de programa por un fragmento (30x) de código de biblioteca sólo cuando ambos fragmentos (42, 30x) de código son idénticos en su forma como código de máquina ejecutable.

3. Procedimiento según la reivindicación 1, caracterizado porque al menos algunos fragmentos (30x) de código de biblioteca están parametrizados.

4. Procedimiento según una de las reivindicaciones 1 a 3, caracterizado porque un fragmento (42) de código de programa que va a sustituirse se sustituye al menos cuando no interviene en el flujo de programa por una instrucción (46) de llamada de subrutina para el fragmento (30x) de código de biblioteca correspondiente.

5. Procedimiento según una de las reivindicaciones 1 a 4, caracterizado porque el código (38) de programa generado por compilador está presente en forma de código fuente de ensamblador, y porque la operación de optimización se realiza a nivel de código fuente.

6. Procedimiento según una de las reivindicaciones 1 a 5, caracterizado porque la biblioteca (28) predefinida está adaptada al hardware del soporte (10) de datos portátil y/o a un sistema operativo del soporte (10) de datos portátil y/o a un compilador (36) utilizado en la generación del código (38) de programa generado por compilador.

7. Procedimiento según una de las reivindicaciones 1 a 6, caracterizado porque la primera zona (24) de almacenamiento puede programarse eléctricamente, y/o porque la segunda zona (26) de almacenamiento puede programarse por máscara, y/o porque la primera zona (24) de almacenamiento en el soporte (10) de datos portátil requiere más superficie de chip por celda de memoria que la segunda zona (26) de almacenamiento.

8. Producto de programa informático con instrucciones de programa para un ordenador de propósito general, que hacen que el ordenador de propósito general ejecute un procedimiento según una de las reivindicaciones 1 a 7.

9. Producto de programa informático según la reivindicación 8, caracterizado porque las instrucciones de programa implementan además un compilador (36) para convertir un código (34) fuente de alto nivel en el código (38) de programa generado por compilador.

10. Soporte (10) de datos portátil con un núcleo (12) de procesador, una primera zona (24) de almacenamiento y una segunda zona (26) de almacenamiento, en el que en la primera zona (24) de almacenamiento está contenido un código (44) de programa optimizado, que se generó mediante un procedimiento según una de las reivindicaciones 1 a 7, y en la segunda zona (26) de almacenamiento está contenida una biblioteca (28) predefinida independientemente del código (44) de programa optimizado con una pluralidad de fragmentos (30x) de código de biblioteca.

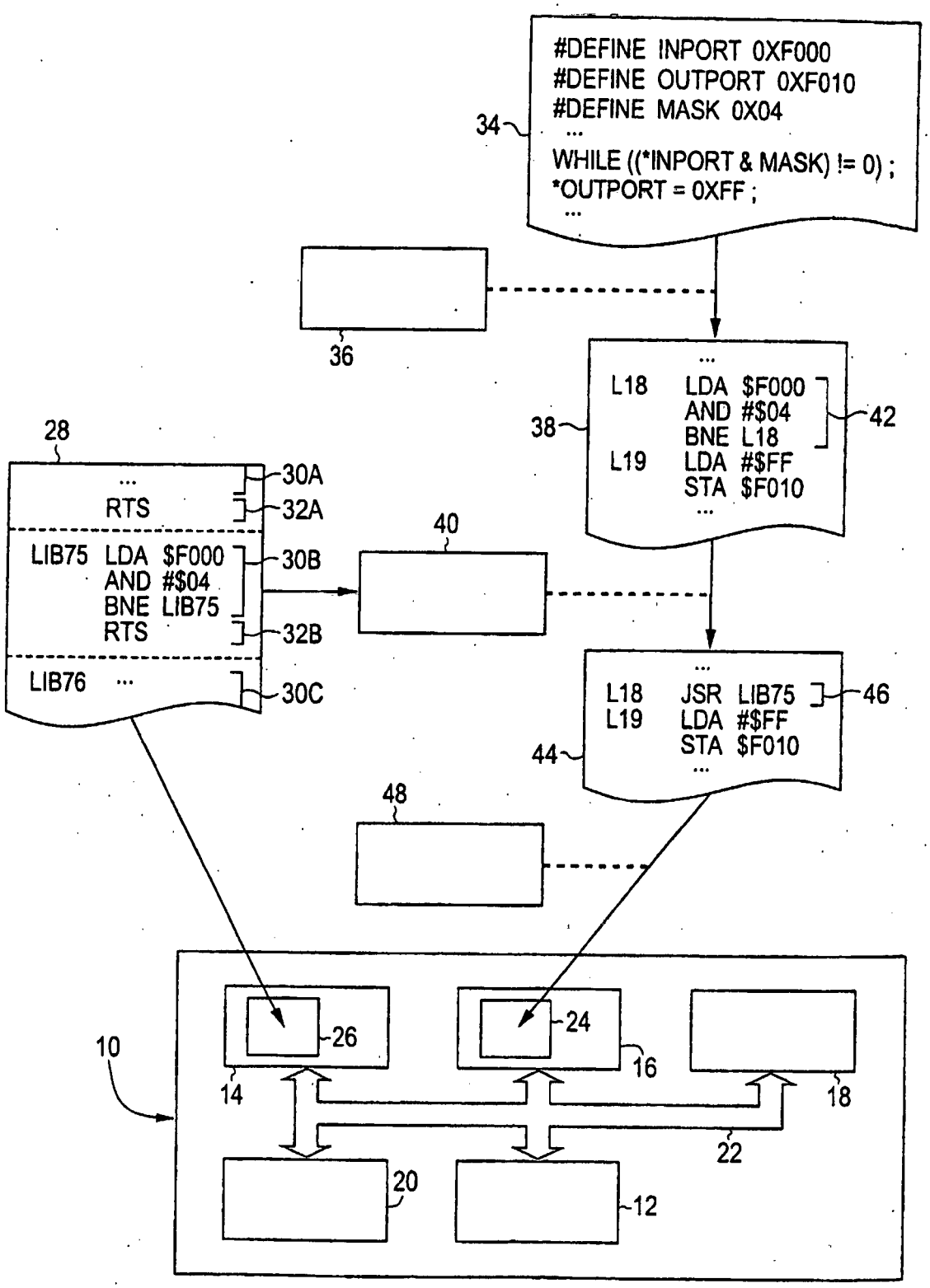
#DEFINE INPORT 0XF000
#DEFINE OUTPORT 0XF010
#DEFINE MASK 0X04
...
WHILE ((*INPORT & MASK) != 0) ;
*OUTPORT = 0XFF ;
...
34
36
...
L18 LDA $F000
AND #$04
BNE L18
L19 LDA #$FF
STA $F010
...
38
42
28
...
RTS
30A
32A
LIB75 LDA $F000
AND #$04
BNE LIB75
RTS
30B
32B
LIB76 ...
30C
40
...
L18 JSR LIB75
L19 LDA #$FF
STA $F010
...
46
44
48
10
26
24
16
18
14
22
20
12

Fig. 1