

(19) 日本国特許庁(JP)

(12) 特 許 公 報(B2)

(11) 特許番号

特許第3861925号

(P3861925)

(45) 発行日 平成18年12月27日(2006.12.27)

(24) 登録日 平成18年10月6日(2006.10.6)

(51) Int. Cl.

G06F 9/455 (2006.01)

F I

G06F 9/44 310A

請求項の数 9 (全 54 頁)

(21) 出願番号	特願2006-71248 (P2006-71248)	(73) 特許権者	000002369
(22) 出願日	平成18年3月15日(2006.3.15)		セイコーエプソン株式会社
(62) 分割の表示	特願2006-35051 (P2006-35051)		東京都新宿区西新宿2丁目4番1号
	の分割	(74) 代理人	110000028
原出願日	平成7年12月11日(1995.12.11)		特許業務法人明成国際特許事務所
(65) 公開番号	特開2006-155673 (P2006-155673A)	(72) 発明者	緒方 英昭
(43) 公開日	平成18年6月15日(2006.6.15)		長野県諏訪市大和三丁目3番5号 セイコ
審査請求日	平成18年4月14日(2006.4.14)		ーエプソン株式会社内
(31) 優先権主張番号	特願平6-305957	(72) 発明者	谷本 晃仁
(32) 優先日	平成6年12月9日(1994.12.9)		長野県諏訪市大和三丁目3番5号 セイコ
(33) 優先権主張国	日本国(JP)		ーエプソン株式会社内
(31) 優先権主張番号	特願平7-260724	(72) 発明者	中岡 康
(32) 優先日	平成7年9月12日(1995.9.12)		長野県諏訪市大和三丁目3番5号 セイコ
(33) 優先権主張国	日本国(JP)		ーエプソン株式会社内

最終頁に続く

(54) 【発明の名称】 エミュレーション装置およびその方法

(57) 【特許請求の範囲】

【請求項1】

少なくとも演算処理部、記憶部、入出力部およびハードディスクを備えたコンピュータである実行マシン上で、所定バイト数のデータを単位とし、該単位のデータを前記ハードディスクにおいて、トラック、セクタを用いて管理するディスクシステムを用い、該ハードディスクから、前記実行マシンとは、データの入出力用の手続を定義したBIOSおよびディスクシステムを用いてデータの処理を行なう点で共通するオペレーティングシステムをロードして、前記実行マシンのアーキテクチャとは異なるアーキテクチャを備えたコンピュータであるターゲットマシン用のプログラムを実行させるエミュレーション装置であって、

前記実行マシンの起動時に、少なくとも二つのタスクが実行できる論理空間を前記実行マシンに確保するタスク管理手段と、

該確保された論理空間に、前記実行マシン用に用意されたBIOSをロードし、実行可能な状態に展開する実行マシンBIOS展開手段と、

前記実行マシン用のBIOSが実行可能に展開された前記一つの論理空間に、該展開された実行マシン用のBIOSを呼び出す所定のファンクションコールを実現するプログラムを実行可能に読み込む実行マシン処理プログラム読込手段と、

前記確保された他の論理空間に、前記ターゲットマシン用に用意されたBIOSをロードし、所定のファンクションコールを利用して実行可能な状態に展開するターゲットマシンBIOS展開手段と、

前記ターゲットマシン用のBIOSが実行可能に展開された前記他の論理空間に、該展開されたターゲットマシン用のBIOSを呼び出す該ターゲットマシンの前記ファンクションコールに対応したアプリケーションプログラムを読み込むターゲットマシンプログラム読込手段と

該展開されたターゲットマシン用のBIOSが前記アプリケーションプログラムから前記システムコールを用いて呼び出されたとき、この呼出を取得して、該呼び出されたBIOSと等価な機能を有する前記実行マシン用のBIOSを呼び出す呼出手段と

を備えたエミュレーション装置。

【請求項2】

請求項1記載のエミュレーション装置であって、

10

前記実行マシンに接続されたハードディスクの記憶領域を検索し、該ハードディスク上に連続して存在する所定の大きさの領域を確保し、該領域を該ターゲットマシンの仮想的なハードディスクとして扱う仮想ディスク確保手段を備えたエミュレーション装置。

【請求項3】

請求項1記載のエミュレーション装置であって、

前記ターゲットマシンプログラム読込手段によりファンクションコールに対応したプログラムの読み込みの後に、該ターゲットマシン用のオペレーティングシステム用に用意されたファイルを読み出すオペレーティングシステム読出手段と、

該読み出したファイルに含まれるオペレーティングシステム用のプログラムを、前記他の論理空間にロードし、該オペレーティングシステム用のプログラムから前記ターゲットマシン用のBIOSおよびディスクシステムをファンクションコールを用いて呼び出し可能に関連づけることにより、前記実行マシンに、前記ターゲットマシン用のオペレーティングシステムを構築するシステム構築手段と

20

を備えたエミュレーション装置。

【請求項4】

少なくとも演算処理部、記憶部、入出力部を備えたコンピュータであり、該コンピュータに用意されたオペレーティングシステムのメモリ空間に周辺装置を利用するためのプログラムであるデバイスドライバを実行可能にロードすることで前記周辺装置を利用可能とする実行マシン上で、前記周辺装置を利用するためにオペレーティングシステムから呼び出されるデバイスドライバを利用する点では共通するオペレーティングシステムをロードして、実行マシンのアーキテクチャとは異なるアーキテクチャを備えたコンピュータであるターゲットマシン用のプログラムを実行させるエミュレーション装置であって、

30

前記実行マシンに接続された前記周辺機器用のデバイスドライバを、メモリ空間にロードし、前記実行マシンの前記オペレーティングシステムから呼び出せるよう関連付けることで、該デバイスドライバを実行可能な状態に組み込む第1の組込手段と、

該第1の組込手段による前記デバイスドライバの組込の後で、前記ターゲットマシンに用意されたオペレーティングシステムを、前記メモリ空間にロードして、実行可能な状態に組み込むオペレーティングシステム組込手段と、

該ターゲットマシン用のオペレーティングシステムに、前記周辺機器用の該ターゲットマシン上で動作可能に作られたデバイスドライバを少なくとも前記オペレーティングシステムから呼び出せるように関連付けることで、該ターゲットマシン用のデバイスドライバを呼び出す処理を実行可能に組み込む第2の組込手段と、

40

前記ターゲットマシンのデバイスドライバを呼び出す前記処理の実行を検出し、前記実行マシンのデバイスドライバを起動して、前記周辺装置へのアクセスを実行させる起動手段と

を備えたエミュレーション装置。

【請求項5】

少なくとも演算処理部、記憶部、入出力部を備えたコンピュータであり、該コンピュータに用意されたオペレーティングシステムのメモリ空間に周辺装置を利用するためのプログラムであるデバイスドライバを実行可能にロードすることで前記周辺装置を利用可能と

50

する実行マシン上で、前記周辺装置を利用するためにオペレーティングシステムから呼び出されるデバイスドライバを利用する点では共通するオペレーティングシステムをロードして、実行マシンのアーキテクチャとは異なるアーキテクチャを備えたコンピュータであるターゲットマシン用のプログラムを実行させるエミュレーション装置であって、

前記実行マシンに接続された前記周辺機器用のデバイスドライバを、メモリ空間にロードし、前記実行マシンの前記オペレーティングシステムから呼び出せるよう関連付けることで、該デバイスドライバを実行可能な状態に組み込む第1の組込手段と、

該実行マシン用のデバイスドライバが実行される際に、該デバイスドライバが前記周辺機器との間でデータをやり取りする際に用いる第1のバッファと、

該第1の組込手段による前記デバイスドライバの組込の後で、前記ターゲットマシンに用意されたオペレーティングシステムを、前記メモリ空間にロードして、実行可能な状態に組み込むオペレーティングシステム組込手段と、

該ターゲットマシン用のオペレーティングシステムに、前記周辺機器用の該ターゲットマシン上で動作可能に作られたデバイスドライバを少なくとも前記オペレーティングシステムから呼び出せるように関連付けることで、該ターゲットマシン用のデバイスドライバを呼び出す処理を実行可能に組み込む第2の組込手段と、

該ターゲットマシンのデバイスドライバが実行される際に、該デバイスドライバが前記周辺機器との間でデータをやり取りするのに用いる第2のバッファと、

前記ターゲットマシンのデバイスドライバを呼び出す前記処理の実行を検出し、前記実行マシンのデバイスドライバを起動して、前記周辺装置へのアクセスを実行させると共に、前記第1のバッファと前記第2のバッファとの間で、前記周辺装置との間でやり取りするデータの転送を行なわせる転送手段と

を備えたエミュレーション装置。

【請求項6】

請求項5記載のエミュレーション装置であって、

前記転送手段は、前記実行マシン用の前記デバイスドライバが実行される際に一度に扱う前記第1のバッファのデータ量と前記ターゲットマシン用の前記デバイスドライバが実行される際に一度に扱う前記第2のバッファのデータ量との差異に基づいて、両バッファ間で前記差異を解消する回数のデータ転送を行なう手段であるエミュレーション装置。

【請求項7】

少なくとも演算処理部、記憶部、入出力部およびハードディスクを備えたコンピュータである実行マシン上で、所定バイト数のデータを単位とし、該単位のデータを前記ハードディスクにおいて、トラック、セクタを用いて管理するディスクシステムを用い、該ハードディスクから、前記実行マシンとは、データの入出力用の手続を定義したBIOSおよびディスクシステムを用いてデータの処理を行なう点で共通するオペレーティングシステムをロードして、前記実行マシンのアーキテクチャとは異なるアーキテクチャを備えたコンピュータであるターゲットマシン用のプログラムを実行させるエミュレーション方法であって、

前記実行マシンの起動時に、少なくとも二つのタスクが実行できる論理空間を前記実行マシンに確保し、

該確保された一つの論理空間に、前記実行マシン用に用意されたBIOSをロードし、実行可能な状態に展開し、

前記実行マシン用のBIOSが実行可能に展開された前記一つの論理空間に、該展開された実行マシン用のBIOSを呼び出す所定のファンクションコールを実現するプログラムを実行可能に読み込み、

前記確保された他の論理空間に、前記ターゲットマシン用に用意されたBIOSをロードし、所定のファンクションコールを利用して実行可能な状態に展開し、

前記ターゲットマシン用のBIOSが実行可能に展開された前記他の論理空間に、該展開されたターゲットマシン用のBIOSを呼び出す該ターゲットマシンの前記ファンクションコールに対応したアプリケーションプログラムを読み込み、

10

20

30

40

50

該展開されたターゲットマシン用のBIOSが前記アプリケーションプログラムから前記システムコールを用いて呼び出されたとき、この呼出を取得して、該呼び出されたBIOSと等価な機能を有する前記実行マシン用のBIOSを呼び出す

エミュレーション方法。

【請求項8】

少なくとも演算処理部、記憶部、入出力部を備えたコンピュータであり、該コンピュータに用意されたオペレーティングシステムのメモリ空間に周辺装置を利用するためのプログラムであるデバイスドライバを実行可能にロードすることで前記周辺装置を利用可能とする実行マシン上で、前記周辺装置を利用するためにオペレーティングシステムから呼び出されるデバイスドライバを利用する点では共通するにデバイスドライバを組み込んで動作する周辺装置を備えた実行マシン上で、実行マシンのアーキテクチャとは異なるアーキテクチャを備えたコンピュータであるターゲットマシン用のプログラムを実行させるエミュレーション方法であって、

10

前記実行マシンに接続された前記周辺機器用のデバイスドライバを、メモリ空間にロードし、前記実行マシンの前記オペレーティングシステムから呼び出せるよう関連付けることで、該デバイスドライバを実行可能な状態に組み込み、

該デバイスドライバの組込の後で、前記ターゲットマシンに用意されたオペレーティングシステムを、前記メモリ空間にロードして、実行可能な状態に組み込み、

該ターゲットマシン用のオペレーティングシステムに、前記周辺機器用の該ターゲットマシン上で動作可能に作られたデバイスドライバを少なくとも前記オペレーティングシステムから呼び出せるように関連付けることで、該ターゲットマシン用のデバイスドライバを呼び出す処理を実行可能に組み込み、

20

前記ターゲットマシンのデバイスドライバを呼び出す前記処理の実行を検出し、前記実行マシンのデバイスドライバを起動して、前記周辺装置へのアクセスを実行させる

エミュレーション方法。

【請求項9】

少なくとも演算処理部、記憶部、入出力部を備えたコンピュータであり、該コンピュータに用意されたオペレーティングシステムのメモリ空間に周辺装置を利用するためのプログラムであるデバイスドライバを実行可能にロードすることで前記周辺装置を利用可能とする実行マシン上で、前記周辺装置を利用するためにオペレーティングシステムから呼び出されるデバイスドライバを利用する点では共通するオペレーティングシステムをロードして、実行マシンのアーキテクチャとは異なるアーキテクチャを備えたコンピュータであるターゲットマシン用のプログラムを実行させるエミュレーション方法であって、

30

前記実行マシンに接続された前記周辺機器用のデバイスドライバを、メモリ空間にロードし、前記実行マシンの前記オペレーティングシステムから呼び出せるよう関連付けることで、該デバイスドライバを実行可能な状態に組み込み、

該実行マシン用のデバイスドライバが実行される際に、該デバイスドライバが前記周辺機器との間でデータをやり取りする際に用いる第1のバッファを用意し、

前記デバイスドライバの組込の後で、前記ターゲットマシンに用意されたオペレーティングシステムを、前記メモリ空間にロードして、実行可能な状態に組み込み、

40

該ターゲットマシン用のオペレーティングシステムに、前記周辺機器用の該ターゲットマシン上で動作可能に作られたデバイスドライバを少なくとも前記オペレーティングシステムから呼び出せるように関連付けることで、該ターゲットマシン用のデバイスドライバを呼び出す処理を実行可能に組み込み、

該ターゲットマシンのデバイスドライバが実行される際に、該デバイスドライバが前記周辺機器との間でデータをやり取りするのに用いる第2のバッファを用意し、

前記ターゲットマシンのデバイスドライバを呼び出す前記処理の実行を検出し、前記実行マシンのデバイスドライバを起動して、前記周辺装置へのアクセスを実行させると共に、第1のバッファと前記第2のバッファとの間で、前記周辺装置との間でやり取りするデータの転送を行なわせる

50

エミュレーション方法。

【発明の詳細な説明】

【技術分野】

【0001】

本発明は、エミュレーション装置およびその方法に関し、詳しくは少なくとも演算処理部、記憶部、入出力部を備えたコンピュータである実行マシン上で、実行マシンのアーキテクチャとは異なるアーキテクチャを備えたコンピュータであるターゲットマシン用のプログラムを実行させるエミュレーション装置及びその方法に関する。

【背景技術】

【0002】

従来、この種のエミュレーション装置としては、特開平5-46406号に開示されているように、ターゲットマシンのOSファンクションコール、BIOSファンクションコールやI/O命令、さらには割り込みテーブルなどを、個々にエミュレーションすることで、ターゲットマシン用に作られたアプリケーションプログラムを実行マシン上で実行可能とする仮想計算機エミュレーション装置が知られている。

【0003】

こうしたエミュレーション装置では、特定の中央演算処理部（以下、CPUという）に備えられた機能を用い、エミュレーションを行なうべき命令が実行されたり、エミュレーションを行なうべきアドレスやI/Oがアクセスされた時、これをトラップして、ターゲットマシン上でいかなる処理を実行しようとしているかを解析し、同様の結果が実行マシン上で得られるよう、予め用意した代替処理を実行するのである。例えば、ターゲットマシンにおいて所定の機能が割り当てられたI/Oアドレスをアクセスする命令が実行されようとした場合には、これを特権命令の処理としてトラップし、いかなる機能を実現しようとしているかを解析し、同様の処理がなされるよう実行マシンのI/Oアドレスに対する書込処理を実行するのである。

【発明の開示】

【発明が解決しようとする課題】

【0004】

しかしながら、従来のエミュレーション装置では、あるターゲットマシンを実行マシン上で完全にエミュレートすることは、エミュレーションプログラムが巨大なものになってしまうことから、実際には実現困難であるという問題があった。したがって、通常のエミュレーション装置では、OSが定義しているファンクションコールと、多用されるI/Oアクセス等を中心に、ある程度の範囲をエミュレートしているに過ぎない。

【0005】

また、エミュレーションを容易にするため、ディスクのアクセスや描画命令などの一連の処理からなる機能を単位としてエミュレートするのが一般的であり、この場合には、アプリケーションソフトウェアが、各機能を構成する個々の処理を直接実行しようとする、正常にエミュレートすることができないという問題があった。

【0006】

ターゲットマシンにせよ実行マシンにせよ、それらのアーキテクチャは、それらのマシンの開発の歴史や開発時の制約などを背負っており、単純明快な構成に終始することは期待し難い。細部に立ち入ると、様々な例外や付加機能等を有しており、単純なエミュレーションでは、到底対応できないのである。このため、マシンを構成するあらゆる処理、機能について完全なエミュレーションを実現しようすると、エミュレーション装置の構成も複雑なものとなり、エミュレーション用のプログラムも複雑化し巨大化するという問題があった。特にこうした問題は、ハードディスクなどの外部記憶装置をアクセスする場合に、既存のファイルシステムとの関係、あるいは周辺機器制御するデバイスドライバの実現方法を巡って顕在化する。

【0007】

本発明のエミュレーション装置およびその方法は、こうした問題を解決し、あるターゲ

10

20

30

40

50

ットマシンのアーキテクチャの大部分を、他の実行マシン上でエミュレートすることを目的としてなされ、次の構成を採った。

【課題を解決するための手段】

【0035】

本発明の第1のエミュレーション装置は、

少なくとも演算処理部、記憶部、入出力部およびハードディスクを備えたコンピュータである実行マシン上で、所定バイト数のデータを単位とし、該単位のデータを前記ハードディスクにおいて、トラック、セクタを用いて管理するディスクシステムを用い、該ハードディスクから、前記実行マシンとは、データの入出力用の手続を定義したBIOSおよびディスクシステムを用いてデータの処理を行なう点で共通するオペレーティングシステムをロードして、前記実行マシンのアーキテクチャとは異なるアーキテクチャを備えたコンピュータであるターゲットマシン用のプログラムを実行させるエミュレーション装置であって、

10

前記実行マシンの起動時に、少なくとも二つのタスクが実行できる論理空間を前記実行マシンに確保するタスク管理手段と、

該確保された論理空間に、前記実行マシン用に用意されたBIOSをロードし、実行可能な状態に展開する実行マシンBIOS展開手段と、

前記実行マシン用のBIOSが実行可能に展開された前記一つの論理空間に、該展開された実行マシン用のBIOSを呼び出す所定のファンクションコールを実現するプログラムを実行可能に読み込む実行マシン処理プログラム読込手段と、

20

前記確保された他の論理空間に、前記ターゲットマシン用に用意されたBIOSをロードし、所定のファンクションコールを利用して実行可能な状態に展開するターゲットマシンBIOS展開手段と、

前記ターゲットマシン用のBIOSが実行可能に展開された前記他の論理空間に、該展開されたターゲットマシン用のBIOSを呼び出す該ターゲットマシンの前記ファンクションコールに対応したアプリケーションプログラムを読み込むターゲットマシンプログラム読込手段と

該展開されたターゲットマシン用のBIOSが前記アプリケーションプログラムから前記システムコールを用いて呼び出されたとき、この呼出を取得して、該呼び出されたBIOSと等価な機能を有する前記実行マシン用のBIOSを呼び出す呼出手段と

30

を備えたことを要旨としている。

【0036】

このエミュレーション装置に対応した第2のエミュレーション方法の発明は、

少なくとも演算処理部、記憶部、入出力部およびハードディスクを備えたコンピュータである実行マシン上で、所定バイト数のデータを単位とし、該単位のデータを前記ハードディスクにおいて、トラック、セクタを用いて管理するディスクシステムを用い、該ハードディスクから、前記実行マシンとは、データの入出力用の手続を定義したBIOSおよびディスクシステムを用いてデータの処理を行なう点で共通するオペレーティングシステムをロードして、前記実行マシンのアーキテクチャとは異なるアーキテクチャを備えたコンピュータであるターゲットマシン用のプログラムを実行させるエミュレーション方法であって、

40

前記実行マシンの起動時に、少なくとも二つのタスクが実行できる論理空間を前記実行マシンに確保し、

該確保された一つの論理空間に、前記実行マシン用に用意されたBIOSをロードし、実行可能な状態に展開し、

前記実行マシン用のBIOSが実行可能に展開された前記一つの論理空間に、該展開された実行マシン用のBIOSを呼び出す所定のファンクションコールを実現するプログラムを実行可能に読み込み、

前記確保された他の論理空間に、前記ターゲットマシン用に用意されたBIOSをロードし、所定のファンクションコールを利用して実行可能な状態に展開し、

50

前記ターゲットマシン用のBIOSが実行可能に展開された前記他の論理空間に、該展開されたターゲットマシン用のBIOSを呼び出す該ターゲットマシンの前記ファンクションコールに対応したアプリケーションプログラムを読み込み、

該展開されたターゲットマシン用のBIOSが前記アプリケーションプログラムから前記システムコールを用いて呼び出されたとき、この呼出を取得して、該呼び出されたBIOSと等価な機能を有する前記実行マシン用のBIOSを呼び出すこと

を要旨としている。

【0037】

このエミュレーション装置またはその方法によれば、実行マシンの起動時に、実行マシンには少なくとも二つのタスクが実行できる論理空間が確保され、このタスクの一つに実行マシン用に用意されたBIOSが実行可能に展開され、他のタスクにターゲットマシン用に用意されたBIOSが実行可能に展開される。ターゲットマシン用のBIOSがアプリケーションプログラム（オペレーティングシステムも含む）から呼ばれたとき、この呼出を取得して、呼び出されたBIOSと等価な機能を有する実行マシン用のBIOSが呼び出される。したがって、ターゲットマシン用のBIOSの呼出を等価な機能を有する実行マシン用のBIOSで代替でき、エミュレーションを簡略に構成できるという利点がある。

10

【0038】

ここで、実行マシンに接続されたハードディスクの記憶領域に、所定の大きさの領域を確保し、該領域を該ターゲットマシンの仮想的なハードディスクとして扱う仮想ディスク確保手段と、

20

前記実行マシン用のBIOSが実行可能に展開された前記一つのタスクに、該展開された実行マシン用のBIOSを利用して該実行マシンに所定のファンクションコールを実現するプログラムを実行可能に読み込む実行マシン処理プログラム読込手段と、

前記ターゲットマシン用のBIOSが実行可能に展開された前記他のタスクに、該展開されたターゲットマシン用のBIOSを利用する該ターゲットマシンのファンクションコールに対応したプログラムを読み込むターゲットマシンプログラム読込手段と

を設けることも可能である。この場合には、第5発明において、BIOSを展開したのに対して、BIOSを利用して実行マシンに所定の所定のファンクションコールを実現することができる。したがって、両マシンに対応するファンクションコールが存在する場合には、これを呼び出す構成とすれば良く、エミュレーションの精度を向上し、かつエミュレーション装置を容易に構成することができる。

30

【0039】

ここで、ターゲットマシンプログラム読込手段によりファンクションコールに対応したプログラムの読み込みの後に、該ターゲットマシン用のオペレーティングシステムを構築するためのファイルを読み出すオペレーティングシステム読出手段と、該読み出したファイルを用いて、前記実行マシンに、前記ターゲットマシン用のオペレーティングシステムを構築するシステム構築手段とを備えた構成を採ることも可能である。この場合には、ターゲットマシンのオペレーティングシステムを実行マシンに容易に構築することができる。

40

【0040】

次に、本発明の第2のエミュレーション装置について説明する。第2のエミュレーション装置は、

少なくとも演算処理部、記憶部、入出力部を備えたコンピュータであり、該コンピュータに用意されたオペレーティングシステムのメモリ空間に周辺装置を利用するためのプログラムであるデバイスドライバを実行可能にロードすることで前記周辺装置を利用可能とする実行マシン上で、前記周辺装置を利用するためにオペレーティングシステムから呼び出されるデバイスドライバを利用する点では共通するオペレーティングシステムをロードして、実行マシンのアーキテクチャとは異なるアーキテクチャを備えたコンピュータであるターゲットマシン用のプログラムを実行させるエミュレーション装置であって、

50

前記実行マシンに接続された前記周辺機器用のデバイスドライバを、メモリ空間にロードし、前記実行マシンの前記オペレーティングシステムから呼び出せるよう関連付けることで、該デバイスドライバを実行可能な状態に組み込む第1の組込手段と、

該第1の組込手段による前記デバイスドライバの組込の後で、前記ターゲットマシンに用意されたオペレーティングシステムを、前記メモリ空間にロードして、実行可能な状態に組み込むオペレーティングシステム組込手段と、

該ターゲットマシン用のオペレーティングシステムに、前記周辺機器用の該ターゲットマシン上で動作可能に作られたデバイスドライバを少なくとも前記オペレーティングシステムから呼び出せるように関連付けることで、該ターゲットマシン用のデバイスドライバを呼び出す処理を実行可能な組み込む第2の組込手段と、

前記ターゲットマシンのデバイスドライバを呼び出す前記処理の実行を検出し、前記実行マシンのデバイスドライバを起動して、前記周辺装置へのアクセスを実行させる起動手段と

を備えたことを要旨とする。

【0041】

また、このエミュレーション装置に対応した発明である第2のエミュレーション方法は、

少なくとも演算処理部、記憶部、入出力部を備えたコンピュータであり、該コンピュータに用意されたオペレーティングシステムのメモリ空間に周辺装置を利用するためのプログラムであるデバイスドライバを実行可能にロードすることで前記周辺装置を利用可能とする実行マシン上で、前記周辺装置を利用するためにオペレーティングシステムから呼び出されるデバイスドライバを利用する点では共通するにデバイスドライバを組み込んで動作する周辺装置を備えた実行マシン上で、実行マシンのアーキテクチャとは異なるアーキテクチャを備えたコンピュータであるターゲットマシン用のプログラムを実行させるエミュレーション方法であって、

前記実行マシンに接続された前記周辺機器用のデバイスドライバを、メモリ空間にロードし、前記実行マシンの前記オペレーティングシステムから呼び出せるよう関連付けることで、該デバイスドライバを実行可能な状態に組み込み、

該デバイスドライバの組込の後で、前記ターゲットマシンに用意されたオペレーティングシステムを、前記メモリ空間にロードして、実行可能な状態に組み込み、

該ターゲットマシン用のオペレーティングシステムに、前記周辺機器用の該ターゲットマシン上で動作可能に作られたデバイスドライバを少なくとも前記オペレーティングシステムから呼び出せるように関連付けることで、該ターゲットマシン用のデバイスドライバを呼び出す処理を実行可能な組み込み、

前記ターゲットマシンのデバイスドライバを呼び出す前記処理の実行を検出し、前記実行マシンのデバイスドライバを起動して、前記周辺装置へのアクセスを実行させることを要旨としている。

【0042】

このエミュレーション装置またはその方法は、実行マシンのオペレーティングシステムにデバイスを組み込んで使用される周辺装置のエミュレーションに関するものであり、実行マシン用のデバイスドライバとターゲットマシン用のデバイスドライバが存在する場合には、ターゲットマシンに用意されたオペレーティングシステムに組み込まれるデバイスドライバの少なくとも呼出側を組み込み、ターゲットマシン上でこの周辺装置が呼び出されると、この呼出を取得して、実行マシンに組み込まれたデバイスドライバを起動する。この構成によれば、両マシンのデバイスドライバの少なくとも一部をそのまま利用して、周辺装置のエミュレーションを容易に行なうことができる。なお、周辺装置としては、光磁気ディスク、フレキシブルディスク、ハードディスク、CD-ROM等の記憶装置のみならず、モータやソレノイドなどのアクチュエータを駆動する装置、MIDI装置、スキャナ、プリンタなど、デバイスドライバを用いる総ての周辺装置に適用可能である。

【0043】

次に、本発明の第3のエミュレーション装置について説明する。本発明の第3のエミュレーション装置は、

少なくとも演算処理部、記憶部、入出力部を備えたコンピュータであり、該コンピュータに用意されたオペレーティングシステムのメモリ空間に周辺装置を利用するためのプログラムであるデバイスドライバを実行可能にロードすることで前記周辺装置を利用可能とする実行マシン上で、前記周辺装置を利用するためにオペレーティングシステムから呼び出されるデバイスドライバを利用する点では共通するオペレーティングシステムをロードして、実行マシンのアーキテクチャとは異なるアーキテクチャを備えたコンピュータであるターゲットマシン用のプログラムを実行させるエミュレーション装置であって、

前記実行マシンに接続された前記周辺機器用のデバイスドライバを、メモリ空間にロードし、前記実行マシンの前記オペレーティングシステムから呼び出せるよう関連付けることで、該デバイスドライバを実行可能な状態に組み込む第1の組込手段と、

該実行マシン用のデバイスドライバが実行される際に、該デバイスドライバが前記周辺機器との間でデータをやり取りする際に用いる第1のバッファと、

該第1の組込手段による前記デバイスドライバの組込の後で、前記ターゲットマシンに用意されたオペレーティングシステムを、前記メモリ空間にロードして、実行可能な状態に組み込むオペレーティングシステム組込手段と、

該ターゲットマシン用のオペレーティングシステムに、前記周辺機器用の該ターゲットマシン上で動作可能に作られたデバイスドライバを少なくとも前記オペレーティングシステムから呼び出せるように関連付けることで、該ターゲットマシン用のデバイスドライバを呼び出す処理を実行可能に組み込む第2の組込手段と、

該ターゲットマシンのデバイスドライバが実行される際に、該デバイスドライバが前記周辺機器との間でデータをやり取りするのに用いる第2のバッファと、

前記ターゲットマシンのデバイスドライバを呼び出す前記処理の実行を検出し、前記実行マシンのデバイスドライバを起動して、前記周辺装置へのアクセスを実行させると共に、前記第1のバッファと前記第2のバッファとの間で、前記周辺装置との間でやり取りするデータの転送を行なわせる転送手段と

を備えることを要旨とする。

【0044】

また、このエミュレーション装置に対応した第3のエミュレーション方法は、

少なくとも演算処理部、記憶部、入出力部を備えたコンピュータであり、該コンピュータに用意されたオペレーティングシステムのメモリ空間に周辺装置を利用するためのプログラムであるデバイスドライバを実行可能にロードすることで前記周辺装置を利用可能とする実行マシン上で、前記周辺装置を利用するためにオペレーティングシステムから呼び出されるデバイスドライバを利用する点では共通するオペレーティングシステムをロードして、実行マシンのアーキテクチャとは異なるアーキテクチャを備えたコンピュータであるターゲットマシン用のプログラムを実行させるエミュレーション方法であって、

前記実行マシンに接続された前記周辺機器用のデバイスドライバを、メモリ空間にロードし、前記実行マシンの前記オペレーティングシステムから呼び出せるよう関連付けることで、該デバイスドライバを実行可能な状態に組み込み、

該実行マシン用のデバイスドライバが実行される際に、該デバイスドライバが前記周辺機器との間でデータをやり取りする際に用いる第1のバッファを用意し、

前記デバイスドライバの組込の後で、前記ターゲットマシンに用意されたオペレーティングシステムを、前記メモリ空間にロードして、実行可能な状態に組み込み、

該ターゲットマシン用のオペレーティングシステムに、前記周辺機器用の該ターゲットマシン上で動作可能に作られたデバイスドライバを少なくとも前記オペレーティングシステムから呼び出せるように関連付けることで、該ターゲットマシン用のデバイスドライバを呼び出す処理を実行可能に組み込み、

該ターゲットマシンのデバイスドライバが実行される際に、該デバイスドライバが前記周辺機器との間でデータをやり取りするのに用いる第2のバッファを用意し、

10

20

30

40

50

前記ターゲットマシンのデバイスドライバを呼び出す前記処理の実行を検出し、前記実行マシンのデバイスドライバを起動して、前記周辺装置へのアクセスを実行させると共に、第1のバッファと前記第2のバッファとの間で、前記周辺装置との間でやり取りするデータの転送を行なわせること

を要旨としている。

【0045】

このエミュレーション装置またはその方法によれば、周辺機器のアクセスに必要な実行マシン用のデバイスドライバを実行可能に組み込むと共に、そのデバイスドライバによるアクセスに必要な第1のバッファを用意し、その後、ターゲットマシン用のオペレーティングシステムを組み込む。このオペレーティングシステムには、周辺機器用のターゲットマシンのデバイスドライバの少なくとも呼出側を組み込み、これと共に、ターゲットマシンのデバイスドライバを用いた前記周辺機器のアクセスに必要な第2のバッファを用意する。この状態で、ターゲットマシンがデバイスドライバを用いた前記周辺機器のアクセスを行なおうとすると、第2のデバイスドライバの呼出側が呼び出される。この呼出を取得し、実行マシンのデバイスドライバを起動して、周辺装置へのアクセスを実行させると共に、第1のバッファと第2のバッファとの間でデータの転送が行なわれる。この結果、周辺装置に対するアクセスがエミュレートされ、ターゲットマシンから実行マシン用の周辺装置をアクセスすることが可能となる。

10

【0046】

なお、第3のエミュレーション装置またその方法において、転送手段を、第1のバッファのデータ量と第2のバッファのデータ量との差異に基づいて、所定回数の転送を行なう手段とすることができ。この場合には、両デバイスドライバが管理しているバッファの大きさが異なっている、データの転送を容易に行なうことができる。

20

【発明を実施するための最良の形態】

【0047】

以上説明した本発明の構成・作用を一層明らかにするために、以下本発明の好適な実施例について説明する。図1は、本実施例のエミュレーション装置の構成を概念的に示すブロック図である。このエミュレーション装置は、実際には、PC-AT機と互換性を有するDOS/V機（PC-ATおよびDOS/Vは、IBM社の商標）を実行マシンとし、その上で、ターゲットマシンであるPC-9800シリーズ（PC-9800は日本電気株式会社の商標）のアーキテクチャを実現するものである。図2は、DOS/V機の概略構成を示すブロック図、図3は、そのメモリおよびI/Oマップを示す説明図である。

30

【0048】

説明の便宜上、まず図2に従い、実行マシンであるDOS/V機の構成について説明する。このDOS/V機は、図示するように、ローカルバス22に接続された演算処理部20、ローカルバス22を外部バスの一つであるPCIバス32に接続するPCIブリッジ30、PCIバス32を介して演算処理部20のCPU21等によりアクセスを受けるコントローラ部40、各種のI/O装置等を制御する機器が低速の外部バスであるISAバス42に接続されたI/O部60、および周辺機器であるキーボード72、スピーカ74、CRT76などから構成されている。

40

【0049】

演算処理部20は、中央演算処理装置としてのCPU21（本実施例ではインテル社製Pentium（登録商標）を使用）、キャッシュメモリ23、そのキャッシュコントローラ24およびメインメモリ25から構成されている。PCIブリッジ30は、高速のPCIバス32を制御する機能を備えたコントローラである。CPU21が扱うメモリ空間は、CPU21の内部に用意された各種レジスタ（セグメントレジスタなどのメモリマネージメント用レジスタ）により、実際の物理アドレス空間より広い論理アドレス空間に拡張されている。

【0050】

コントローラ部40は、モニタ（CRT）76への画像の表示を司るグラフィックスコ

50

ントローラ（以下、VGAと呼ぶ）44、接続されるSCSI機器とのデータ転送を司るSCSIコントローラ46、PCIバス32と下位のISAバスとのインタフェースを司るPCI-ISAブリッジ48から構成されている。VGA44は、CRT76に対して、640×480ドット、16色表示が可能である。なお、表示用のフォントを記憶したキャラクタジェネレータや所定のコマンドを受け取って所定の図形を描画するグラフィックコントローラ、さらには描画画像を記憶するビデオメモリ等は、このVGA44に実装されているが、これらの構成は周知のものなので、図2では省略した。

【0051】

PCI-ISAブリッジ48を介して接続されたISAバス42は、各種のI/O機器が接続される入出力制御用のバスであり、DMAコントローラ（以下単にDMAと呼ぶ）50、リアルタイムクロック（RTC）52、複合I/Oポート54、サウンドI/O56、キーボード72およびマウス73のインタフェースを司るキーボードインタフェース（以下KEYと呼ぶ）64、優先順位を有する割り込み制御を行なう割り込みコントローラ（以下PICと呼ぶ）66、各種の時間カウントやビープ音を発生するタイマ68等から構成されている。なお、ISAバス42には、拡張ボードが実装可能なISAスロット62が接続されている。

【0052】

複合I/Oポート54には、パラレル出力、シリアル出力の他、フロッピディスク装置82やハードディスク84を制御する信号を入出力するポートが用意されている。また、パラレル入出力には、パラレルポート86を介してプリンタ88が、シリアル入出力には、シリアルポート90を介してモデム92が、各々接続されている。また、サウンドI/O56には、上述したスピーカ74の他、マイクロフォン96が接続可能とされている。これらの構成の他、DOS/V機では、標準化されたI/Oチャンネルが用意されることも多いが、本実施例では図示および説明は省略する。

【0053】

かかる構成を有するDOS/V機において、PC-9800機のアーキテクチャを実現する手法について、以下説明する。まず最初に、本実施例の実行マシンであるDOS/V機のCPU21が有するメモリ管理機能と、実行モードについて説明する。図3は、タスクが管理するアドレス空間と物理アドレスとの関係を示す説明図である。また、図4は、リアルモード、保護（プロテクト）モード、仮想8086モード（以下、仮想86モードと略す）間の関係を示す説明図である。i80386（インテル社製）以上のCPUでは、タスクが管理するアドレスを物理アドレスに自由に割り当て、タスク内では物理アドレスではなく論理アドレスで処理を行なうことができる機能が備えられている。従って、複数のタスクを異なる物理アドレスの範囲に割り付けておけば、それぞれのタスク内で同一のアドレスをアクセスしても、実際のアクセスは、異なる物理アドレスに対して行なわれるから、他のタスクには、何の影響も与えないのである。

【0054】

また、このCPU21は、リアルモード、保護モード、仮想86モードの3つの動作モードを有する。リアルモードとは、このCPU21を高速の8086として動作させるモードあり、MS-DOS（マイクロソフト社の商標）等を立ち上げる場合には、通常このリアルモードで処理を開始する。また、保護モードは、特権レベルを設定したリング保護の機能を用いた動作モードであり、セグメント単位で、あるいは所定の命令毎に特権レベルを設定することにより、特定の処理によるハードウェアに対する直接アクセスからの保護を実現する。例えば、アプリケーションプログラムの特権レベルを最下位にしておくことで、システムとして規定されたアドレス範囲やI/Oを直接アクセスできないものとすることができる。直接のアクセスが生じた場合には、いわゆるトラップが生じて例外処理が起動され、処理を特定のタスク（カーネルと呼ばれる）に移行するものとすることができる。更に仮想86モードは、保護モードのままで8086のコードを直接実行可能とするモードであり、CPU21は、保護モードと同様に動作するが、アプリケーションプログラムにより指定されている論理番地を8086と同様に解釈して実行する。このモード

10

20

30

40

50

を利用すれば、例えば複数のMS-DOSを起動することも可能である。

【0055】

本実施例のDOS/V機に電源が投入されると、このDOS/V機は、ハードディスクのブートブロックを読み出し、ハードディスク上のOS等を読み込み可能とする。また、ROMで提供されているDOS/V機としてのBIOSを主記憶上でアクセス可能な状態とする（後述する図7符号A1）。DOS/V機のメモリマップおよびI/Oマップを図5に示す。その後、本実施例における初期化手段である初期化処理ルーチンが実行される。この処理ルーチンを図6に示す。また、図6の初期化処理ルーチンの実行の様子を図7の説明図に示す。以下、図6のフローチャートと図7の説明図を参照しつつ、エミュレーション装置が初期化される様子を説明する。

10

【0056】

図6に示した初期化処理ルーチンが起動されると、まず、エミュレーションプログラムのカーネルをハードディスク84からロードする処理を行なう（ステップS100、図7符号A2）。カーネルとは、エミュレーションを行なう処理の内、例外処理が起動された際、その原因の解析を行ない、ディスパッチャと呼ばれる処理を介して実際にエミュレーションを行なう実行モジュールを呼び出す一連の手続を言う。また、ディスパッチャとは、エミュレーションを行なう処理のうち、カーネルによる分析に基づいて実際にエミュレーションを行なう実行モジュールを呼び出す一連の手続をいう。このカーネルのロードに引き続き、カーネルをその実行位置に転送し（ステップS110、図7符号A3）、カーネルに処理を移行する（ステップS120）。従って、図6では一連の処理として示しているが、ステップS130以下は、カーネル自身の処理となる。

20

【0057】

カーネルは、プロテクトモード用のいくつかのプログラムを転送し、CPU21の動作モードをプロテクトモードに切り換えて、DOS/V用のメモリ空間およびPC-9800用のメモリ空間を、それぞれ上位のアドレスに確保する処理を行なう（ステップS130、図7符号A4）。このメモリ空間は、仮想的なDOS/V用のタスクおよびPC-9800用のタスクに対応しており、このメモリ空間を論理的には、000000Hから始まる1Mの主記憶として割り当てるのである。次に、カーネルと共に動作するディスパッチャ（図7符号A5）、更にエミュレーションを行なう各種の実行モジュールを順次転送する処理を行なう（ステップS140、図7符号A6）。この結果、先に展開された仮想DOS/VのBIOS関係のルーチンを除き、DOS/V機のためのプログラムは、主記憶上から抹消され、代わりにエミュレーションのための実行モジュールが配置される。

30

【0058】

ステップS140で、メモリ上に順次ロードされる実行モジュールは、所定の名称のファイルにテキストデータの形で書かれたリストに従って、ハードディスク84から読み出される。以下、実行モジュールの配置に伴う初期化の処理について説明するが、先に実行モジュールがすべてロードされた後の全体構成について、図1を用いて説明する。カーネルKR、ディスパッチャDPおよび各実行モジュールがロードされ初期化されると、図1に示すように、実施例のエミュレータ装置が、このDOS/V機の上に実現されることになる。このエミュレーション装置は、カーネルKR、ディスパッチャDP、および各種実行モジュールから構成されている。カーネルは、DOS/V機の上に仮想的に用意されたPC-9800機と同一の環境上（仮想PC上という）で実行されるプログラムAPPが特権命令などを実行した場合、例外の発生とみなして呼び出される。ディスパッチャDPは、カーネルKRにより特定された例外原因に基づいて呼び出され、この例外原因に基づいて、機能毎に用意された各実行モジュールを呼び出す。各種実行モジュールは、ディスパッチャDPから呼び出されて予め定められた機能のエミュレーションを行なう。

40

【0059】

実際にエミュレートを行なう実行モジュールは、PC-9800機のハードウェアをその機能に基づいて大まかに分割し、機能毎に用意されている。図1には、主要なモジュールのみ示した。メモリエミュレータMEMは、主記憶やEMSの状態をエミュレートする

50

モジュールである。マウスエミュレータMUMは、DOS/V機のマウス73の動きをPC-9800機のマウスとしてエミュレートするモジュールである。グラフィックエミュレータGEMは、PC-9800機のグラフィック画面をエミュレートするモジュールである。一方、テキストエミュレータTEMは、文字コードにより指定された文字の画像の取得をエミュレートする文字フォントエミュレータCFMと一体となって動作するモジュールであり、PC-9800機のテキスト画面をエミュレートするものである。テキストエミュレータTEMおよびグラフィックエミュレータGEMによりエミュレートされ、仮想的なPC-9800機の画像が作られた後は、表示エミュレータDEMにより実際のCRT76への画像の表示がエミュレートされる。

【0060】

タイマエミュレータTIMは、DOS/V機に内蔵されたタイマ68によりPC-9800機上のタイマ関係の処理をエミュレートするモジュールである。また、キーボードエミュレータKEMは、DOS/V機のキーボード72およびKEY64により、PC-9800機のキーボードからのキー入力をエミュレートするモジュールである。

【0061】

更に、割り込みコントローラエミュレータIRMは、DOS/V機の割り込み機能を用いて、PC-9800機の割り込みをエミュレートするモジュールである。この割り込みコントローラエミュレータIRMは、割り込みという行為を介して多くのモジュールから呼び出される関係にある。DOS/V機における割り込み処理の回路を図8に示す。PC-9800機でも同様であるが、割り込み要求は、多数のデバイスからPIC66に出されるので、これをエミュレートする実行モジュールにおいても、モジュール間で何らかのやりとりが必要となる。なお、図8では図示の都合上、インタフェース回路の一部を省略したが、専用のインタフェース回路を介して割り込み要求が出されることは当然である。例えばマウス73からの割り込み要求は、直接出力されている訳ではなく、インタフェース回路であるKEY64を介して出力されている。

【0062】

DOS/V機における現実のハードウェア割り込みは、特権命令の実行や書込保護が指定された範囲の書込等と共に、カーネルKRに処理を移す例外処理の一つとして扱われている。また、ハードウェア割り込みに起因して、あるいは各モジュール内部の処理に起因して割り込みコントローラエミュレータIRMがPC-9800機の動作をエミュレートするために起こす割り込みは、仮想割り込みの発生としてカーネルKRを呼び出す要因とされている。割り込みコントローラエミュレータIRMの呼び出しの詳細については、後述する。

【0063】

これらの各種実行モジュールは、初期化の処理において、カーネルKRにより、ハードディスク84から読み出され、メモリ上に配置される。各実行モジュールは、初期化の処理においてメモリ上に配置される度に、実行モジュールに予め用意した組み込み処理を一度実行する。このとき実行モジュールが行なう処理の一例を図9ないし図11のフローチャートに示した。また、図9の処理が行なわれる様子を図12の説明図に示した。以下、この図面を用いて、実行モジュールの組み込み時の処理について説明する。

【0064】

実行モジュール（例えば図12実行モジュールDM1）の組み込み処理に移行すると、まず、ディスパッチャDPが所定のメモリエリアに用意したディスパッチャテーブルDTを検索し（ステップS200）、その空き領域にモジュール名（英数字8文字）TM1とその呼出アドレスA1を登録する処理を行なう（ステップS210）。なお、これらの検索および登録の処理は、実際には、図12に示したように、カーネルKRおよびディスパッチャDPの機能を利用して行なわれる。即ち、カーネルKRが各実行モジュールDMからのサービスを受け付けるアドレスAAは、固定アドレスとして用意されており、各実行モジュールDMは、このアドレスAAを呼んで、モジュール名（文字列）TMとその呼出アドレスAの登録を依頼する（図12、符号B1）。すると、カーネルKRは、ディスパ

10

20

30

40

50

ッチャDPの呼出アドレスを、予めディスパッチャDP自身が登録したテーブルを参照して取得し、このアドレスADを利用して、ディスパッチャDPに登録のサービスを要求する。この要求を受けて、ディスパッチャDPがディスパッチャテーブルDTに、そのモジュールの名称（文字列）TMと呼出アドレスAとを登録するのである（図12、符号B2）。

【0065】

カーネルKRの固定アドレスAAを介してディスパッチャDPに各種のサービスを要求する上記手法は、実行モジュールの一つが他のモジュールを呼び出す場合や後述する直接呼出のエントリポイントEPを通信相手のモジュールとの間で登録し合う場合などにも用いられる。例えば、一つの実行モジュールが他の実行モジュールを呼び出す場合には、実行モジュールの呼び出しというサービスを、カーネルKRの固定アドレスAAを介してディスパッチャDPに依頼する。ディスパッチャDPは、ディスパッチャテーブルDTを参照して呼び出そうとするモジュールの呼び出しアドレスを容易に取得し、そのモジュールを呼び出すことができるのである（図12、符号B3）。従って、カーネルKR、ディスパッチャテーブルDTおよび各種実行モジュールを通じて予め固定的に管理する必要があるのは、カーネルKRの固定アドレスAAのみであり、管理が極めて容易となるという利点がある。

【0066】

次に、組込の処理を行なっている実行モジュールDM1にモジュール間通信を行なうとの登録がなされているか否かを判断する（ステップS220）。実行モジュールは、登録ファイルの内容に従って順次組み込まれるが、例えば第n番目の実行モジュールDMnが同様に起動され、自己のモジュール名TMnの登録と呼出アドレスAnの登録を済ませた後（図12符号B2）、モジュール間通信が有るとの判断がなされると（ステップS220）、実行モジュールDMn内に用意されている直接呼出用のエントリポイントEPをパラメータとして用意する処理を行なう（ステップS230）。このエントリポイントEPは、モジュール間通信の対象となる実行モジュールに教える直接呼出用のアドレスであり、教えるエントリポイントがなければ0個のエントリポイントEPを用意し、複数個あればその数のエントリポイントEPを用意するのである。エミュレーションの高速性を確保しようとする、緊密な関係を有するモジュール間では、呼出アドレスAnを利用したモジュール全体の呼出に代えて、そのモジュールが実現する機能の一部をなす処理そのものを直接呼び出したい場合が存在する。こうした場合には、各実行モジュールが対象となるモジュールにエントリポイントEPを覚えておく必要がある。

【0067】

エントリポイントEPを用意した後、次に割り込みの登録が有るか否かを判断し（ステップS240）、割り込みの登録がある場合には、登録する割り込み番号を用意する処理を行なう（ステップS250）。組み込み中の実行モジュールが割り込みを利用するものである場合には、その割り込みの番号を割り込みをエミュレートするモジュールである割り込みコントローラエミュレータIRMに予め登録しておく必要があるため、その番号を用意するのである。以上の処理の後、通信対象となる実行モジュールの呼び出しを、カーネルKRを介してディスパッチャDPに依頼する（ステップS260、図13符号C1）。この時、実行モジュールは、呼出先のモジュールをその名称（文字列）で指定し、用意した直接呼出用のエントリポイントEPや割り込みの登録番号等をディスパッチャDPにパラメータとして引き渡す。

【0068】

この依頼を受けたディスパッチャDPは、図10に示すディスパッチャ呼出ルーチンを実行する。ディスパッチャDPは、呼び出すべきモジュールの名称（文字列）を受け取っているから、ディスパッチャテーブルDTをこの文字列で検索し（ステップS300、図13符号C2）、呼出アドレスを取得してそのモジュールを呼び出す処理を行なう（ステップS310、図13符号C3）。この時、ディスパッチャDPは、呼出元のモジュールから受け取ったパラメータを呼び出したモジュールに引き渡す。

10

20

30

40

50

【0069】

こうしてディスパッチャDPから呼ばれることにより、通信対象となるべき実行モジュールが呼び出され、その実行モジュールは、図11に示したエントリポイントを取得する処理を実行する。この処理が開始されると、まず呼び出したモジュールが何であるかの判別を行なう（ステップS320）。呼び出した実行モジュールがパラメータとして直接呼出のためのエントリポイントEPを用意している場合には、これを判別し（ステップS330）、直接呼出用のエントリポイントEPを登録する処理を行なう（ステップS340）。

【0070】

次に、図11の処理を実行するモジュールが割り込みコントローラエミュレータIRMの場合には、割込の登録があるかを判別する（ステップS350）。割込番号の登録がある場合には、パラメータの一部として受け取った割込番号を登録すると共に、割込による直接呼び出しを行なう場合のエントリポイントEPを割り込む側のモジュールに教えるために準備する処理を行なう（ステップS360）。なお、割込番号の登録の処理を行なうモジュールは、割り込みコントローラエミュレータIRMに限られるから、それ以外のモジュールで図11の処理がなされる場合には、ステップS350、S360の処理は行なわれない。

【0071】

割込の登録に関する処理に続いて、呼び出した側のモジュールが、割込の呼出のためのエントリポイントEP以外に直接呼出の教示を行なうべきエントリポイントEPが存在するモジュールであるか否かの判別を行なう（ステップS370）。直接呼出のエントリポイントEPを呼び出した側のモジュールに教える必要があると判断された場合には、教示する直接呼出のエントリポイントEPを、返し値のパラメータとして用意する処理を行なう（ステップS380）。以上の処理の後、「RTN」に抜けて本ルーチンを終了する。この結果、処理は、いったんディスパッチャDPを介して（図13符号C4）、呼び出した側の実行モジュールに戻る（図13符号C5）。この際、呼び出された側の実行モジュールが返し値として用意したパラメータが存在すれば、このパラメータは、呼び出した側の実行モジュールに渡され、呼び出した側の実行モジュールは、これをエントリポイントEPとして登録する。

【0072】

以上の処理を組み込みを行なう最終の実行モジュールまで繰り返すことにより、初期化の処理は完了する。その後、仮想86モードに実行モードを切り換え、PC-9800機としてそのOSであるMS-DOSをブートする処理を開始する。具体的には、MSDOS.SYS、IO.SYS等のシステムファイルを読み込む処理や、COMMAND.COMを組み込む処理、更にはCONFIG.SYSに登録されているデバイスドライバを組み込む処理などを行なう。これらの処理は、通常のMS-DOSの組み込み処理と同一であり、CPUは仮想86モードに切り換えられているので、PC-9800機がMS-DOSを組み込む処理と何等相違しない。なお、DOS/V機がそのオリジナルな状態でMS-DOSを立ち上げるときも、MSDOS.SYS等同一名称のファイルを読み込むが、これらのファイルは、PC-9800機とDOS/V機とでは、その内容が当然異なる。これらのファイルは一般にルートディレクトリに置くことが決められているが、同一名称のファイルを同じディレクトリに置くことはできない。従って、エミュレーションを開始する以前に、即ちDOS/V機として普通に使用しているときに、専用のアプリケーションプログラムを実行させ、PC-9800機用のMSDOS.SYS等のファイルの名称を、所定の名称（例えばMSDOS.9YS等）に変更しておくのである。なお、エミュレーションを取り止めてDOS/V機として普通に使用する場合には、エミュレーションにより動作しているプログラムにより、ルートディレクトリに置かれたこれらのファイルの名称を元に戻し、PC-9800機用のシステムファイルの名称を変更しておく。

【0073】

実行モジュールは、その組み込み時に少なくとも自己のモジュール名（文字列）TMと

10

20

30

40

50

呼出アドレスAをディスパッチャテーブルDTに登録することが必要であるが、メモリを使用する実行モジュールの場合には、更に必要な実メモリをPC-9800機用に確保した仮想メモリ(98用メモリ空間)に割り当てる処理を行なう。

【0074】

以上の処理により、実行マシン(実施例ではDOS/V機)上でターゲットマシン(実施例ではPC-9800機)をエミュレートする環境が整えられた。そこで、次に図1を参照しつつ、実際にエミュレーションが行なわれる様子を大まかに説明し、更に実行モジュールがどのように処理を行なうかについて、例を挙げて詳述する。

【0075】

初期化の処理が終了してPC-9800用の各種システムファイルが読み込まれ、MS-DOSが起動されると、CRT76には、「A:」などのプロンプトが表示された状態となる。この状態では、MS-DOSが動作していることになるが、所定のアプリケーションを起動しようとして、その実行ファイル名をキーボード72から入力する際にも、キーボードのエミュレーションが実行されるのである。即ち、図1に示した「仮想PC上のプログラム」とは、アプリケーションプログラムのみならずDOSを含めたあらゆるプログラムである。この時、DOS/V機は、プロテクトモードで動作しており、CPU21に備わったリング保護機能は、特権レベル3とされている。この状態で、仮想的に用意された98用メモリ空間にアプリケーションプログラムAPPがロードされ、仮想PC上でアプリケーションプログラムが実行されている状態を取り上げてエミュレーション装置の働きについて説明する。

【0076】

エミュレーションの処理が起動されるには次の4つの要因がある。

- 〈1〉 ページフォルト(メモリが割り当てられていない領域に対するアクセス)
- 〈2〉 一般保護例外(特権命令の実行、I/Oの読み書き、書込保護されたメモリに対する書込命令の実行など)
- 〈3〉 エラー(0除算など)
- 〈4〉 ハード割り込み

これらの要因が生じると、カーネルKRが起動される。この時、各レジスタの値などはすべて保存されているから、カーネルKRは、自己が起動された原因を調べることができる。その結果、アクセスされたアドレスやI/Oが何か、あるいは生じたハード割り込みがいかなる機能に対応しているかを判断し、ディスパッチャDPを呼び出す。ディスパッチャDPを呼び出すアドレスは、図12に示したように、初期化の処理において登録されているから、カーネルKRは、この呼出アドレスADによりディスパッチャDPを呼び出すことができる。なお、カーネルKRに処理が渡った以降は、特権レベルは0に設定される。

【0077】

ディスパッチャDPは、カーネルKRから受け取った情報を元にして呼び出す実行モジュールを特定し、その呼出アドレスをディスパッチャテーブルDTから取得する。ディスパッチャDPは、この呼出アドレスを用いて、各実行モジュールを呼び出すのである。図1には、実行モジュールとして、既述したように、

- ・メモリエミュレータMEM
- ・マウスエミュレータMUM
- ・グラフィックエミュレータGEM
- ・テキストエミュレータTEM
- ・文字フォントエミュレータCFM
- ・表示エミュレータDEM
- ・タイマエミュレータTIM
- ・キーボードエミュレータKEM
- ・ハードディスクエミュレータHDM
- ・割り込みコントローラエミュレータIRM

を示したが、これ以外にも、プリンタやRS-232Cによるシリアル通信を司るエミュレータなど、各種の実行モジュールが存在する。

【0078】

ハードウェア割り込みが発生したり、アプリケーションプログラムAPPが特権命令を実行したりすると、カーネルKRに処理が移行し、カーネルKRは、特権命令の中身やハードウェア割り込みの詳細を検討し、ディスパッチャDPを呼び出す。ディスパッチャDPは、エミュレートすべき内容をカーネルKRから受け取り、ディスパッチャテーブルDTを参照して必要な実行モジュールをその呼出アドレスAnを用いて呼び出す。呼び出された実行モジュールは、自分だけで総ての処理が実行できる場合には、そのまま実行し、他の実行モジュールの処理を必要とする場合は、初期化の処理において予め得ておいたエ

10

【0079】

図14は、通常の実行モジュールの呼び出しとモジュール間通信とを例示した説明図である。図14に示した例では、実行モジュールDM2、DM3、DM5の間でモジュール間通信が行なわれるものとした。図示するように、ディスパッチャDPから呼び出される場合には、その呼出アドレスA2、A3、A5など、モジュール全体の開始アドレスが呼ばれることになるが、モジュール間通信の場合には、初期化の処理において一方的にあるいは相互に教示・取得したエントリポイントEPを利用して、各実行モジュール内の処理から必要に応じて、直接他のモジュールの内部処理を呼ぶことになる。例えば、実行モジュールDM5の内部処理において、実行モジュールDM2の内部処理を直接実行する必要が生じた場合には、エントリポイントEP21を参照し、直接実行モジュールDM2内部の処理が呼び出され、実行される（符号C21）。ディスパッチャDPを介して他の実行モジュールを呼び出す手法でも、呼び出すモジュールが必要とするサービスを示すパラメータを渡すことができれば、他の実行モジュールが用意している処理を個々に利用すること自体は可能であるが、ディスパッチャDPに対するサービスの要求、ディスパッチャDPによるディスパッチャテーブルDTの検索、呼出アドレスAによる呼び出しといった手続が必要になってしまう。これに較べて、モジュール間通信は、必要な処理を直接呼び出すから、極めて高速に必要な処理を実行できるという利点がある。

20

【0080】

各実行モジュールDMにより、アプリケーションプログラムAPPが実行しようとした処理をエミュレートした後、順次処理を復帰し、カーネルKRからアプリケーションプログラムAPPに復帰する。なお、ハードウェア割り込みにより割り込みコントローラエミュレータIRMが呼ばれた場合や、他の実行モジュールの処理から割り込みコントローラエミュレータIRMが呼ばれた場合には、割り込みコントローラエミュレータIRMは、PC-9800機における割込の発生をエミュレートするため、カーネルKRに対して仮想割り込みを発生する。この仮想割り込みによりカーネルKRが呼び出され、カーネルKRは、この仮想割り込みを判断して必要な実行モジュールを再度呼び出したり、場合によってはアプリケーションプログラムAPPに復帰する。

30

【0081】

次に、実際にDOS/V機上で行なわれるPC-9800機の各機能のエミュレーションの一例として、キーボード72からの入力のエミュレーションを詳しく説明する。DOS/V機のキーボード72およびそのインタフェースであるKEY64の構成を図15に示す。他方、PC-9800機の回路は、図16に示す通りである。両図に示したように、DOS/V機では、1チップマイクロコンピュータがキーマトリクスを読み取って、そのデータをKEY64側にクロックに同期した同期通信により、CPU21側とやり取りする構成を採っており、PC-9800機では、キーマトリクスを1チップマイクロコンピュータで読み取って、本体側に同期通信する構成を採っている。両者の通信の仕様は、異なっている。また、両機種種のキーボードは図示しないが、キーボード72そのもののキーの数なども異なっており、更に、同じキーを押し続けた場合のキーコードの発生などの

40

50

メカニズムも異なっている。

【0082】

図17、図18は、DOS/V機のキーボード72からの入力をPC-9800機におけるキーボードからの入力としてエミュレートする様子を示す説明図である。図は、上方からDOS/V機のハードウェア、エミュレーションの実行部、仮想98の各層に分けて描いてある。仮想98とは、PC-9800機用に仮想的に用意されたメモリ空間であって、タスク自身としては、あたかもMS-DOSが用意するPC-9800機の実空間で処理を行なっているように処理を実行できる空間を意味している。但し、この時点でCPU21はプロテクトモードで動作しているので、仮想98においてアプリケーションプログラムAPPが特権命令などを実行しようとする、直ちに例外処理として、タスクは切り換えられ、カーネルKRの処理に移行するのである。

10

【0083】

アプリケーションプログラムAPPの実行中にキーボード72のキーが操作されると、キーボード72からのキーコードがバッファBFに蓄えられ(図17符号D1)、同時にハードウェア上の割込が生じる(INT 09)。この割込要求は、直ちに処理をカーネルKRに移行させる(符号D2)。カーネルKRは、既述したように、ハードウェアの割込を解析してディスパッチャDPに処理を渡し(符号D3)、ディスパッチャDPは、初期化の処理において登録された呼出アドレスを用いてキーボードエミュレータKEMを呼び出す。

【0084】

20

キーボードエミュレータKEMは、バッファBFに保存されたキーコードを読み出し、このキーコードが、PC-9800機のキーコードに変換可能なものか否かを判断をする。両者は完全に一対一に対応しているものではなく、特にDOS/V機では、2つ目のキーコードが与えられて初めて一意に決まるものが存在する。従って、こうしたキーコードの場合には、PC-9800機としてのキーコードの生成は行わず、そのままディスパッチャDP、カーネルKRを介して、キーボード72からの割込が生じた状態に復帰する(符号DR)。他方、PC-9800機としてのキーコードが発生可能と判断すると、キーボードエミュレータKEMは、変換したキーコードを自分が管理するバッファDFに記憶し(符号D5)、続いてモジュール間通信により割り込みコントローラエミュレータIRMを直接呼び出す処理を行なう(符号D6)。

30

【0085】

図9ないし図14を用いて詳述したように、本実施例では、各実行モジュールは、初期化の処理における組み込み時に、モジュール間通信を行なう相手のエン트리ポイントEPを取得している。キーボードエミュレータKEMも、割り込みコントローラエミュレータIRMを直接呼び出すエン트리ポイントEPを取得しており、更に割込番号の登録も済ませている。そこで、キーボードエミュレータKEMから、直接割り込みコントローラエミュレータIRMを呼び出し、割込のキーボード72からの割込の発生を、割り込みコントローラエミュレータIRMに渡す。

【0086】

割り込みコントローラエミュレータIRMは、このモジュール間通信を受けて、割込の内容を分析し、カーネルKRに対して必要な仮想割込を起こす(符号D7)。カーネルKRはこの仮想的な割込要求を受け付け、これを解析し、アプリケーションプログラムAPPがキーボード72からのキー入力を割込処理により受け付けるものであれば、ジャンプテーブルJTを参照し(符号D8)、PC-9800機がキーボード72からの割込を受け付けたときにジャンプする先を取得して、必要な処理ルーチンに移行する(符号D9)。アプリケーションプログラムAPPが、キーボード72からの割込要求によってではなく、自分で定期的にキー入力を見に行くタイプのものであれば、処理は一旦アプリケーションプログラムAPPのレベルに戻される(符号D10)。この場合には、所定時間毎に、キーボード72からのキーコードを読み取るルーチンが呼ばれることになる(符号D11)。なお、この時点で、CPU21の特権レベルは3に戻される。

40

50

【0087】

こうして仮想98上でキーコードを読み取る処理が呼ばれた以降の処理について、図18を参照して説明する。キーコードを読み取る処理は、PC-9800機の所定のI/Oアドレスに割り当てられたキーデータを読み取るI/O命令を実行する。このI/O命令は、特権命令の実行に当たり、特権レベル3では例外処理としてトラップされ、処理はカーネルKRに移行する(図18符号E1)。カーネルKRは、この例外処理の内容を解析し、ディスパッチャDPを介してキーボードエミュレータKEMを呼び出す(符号E2)。キーボードエミュレータKEMは、呼び出された際のパラメータを認識することにより、割込要求に伴う処理かI/Oに対する読み出しに伴う処理かを判別し、バッファDFの内容を読み取る処理を行なう(符号E3)。バッファDFは、先にキーボードエミュレータKEMが書き込んだPC-9800機としてのキーコードを保持しているから、これを同じキーボードエミュレータKEMが読み取ることは容易である。

10

【0088】

ハードウェア割込による処理ではないので、キーボードエミュレータKEMは、ディスパッチャDPを介してカーネルKRへと処理を戻し、更に例外処理を引き起こした処理まで戻って行く(符号E4)。この結果、キーボード72からのキーデータを入力するPC-9800機上の処理がなされ、そのデータはアプリケーションプログラムAPPに渡されることになる。仮想98上に展開されたPC-9800機用のアプリケーションプログラムAPPは、PC-9800機のキーボード72からのキー入力があった場合と同一の動作を継続することになる。即ち、実施例のエミュレーション装置は、PC-9800機

20

【0089】

以上、キーボード72からの入力を例に採って、本実施例のエミュレーション装置の構成と働きについて説明したが、他の実行モジュールもほぼ同様に動作する。例えば、マウス73の動作をエミュレートするマウスエミュレータMUMは、次のように構成されている。PC-9800機のマウスは、所定のインターバルで割込を発生し、CPUに対しては原点位置からの絶対値をデータとして渡すのに対して、DOS/V機のマウス73は、マウス73が動いた場合に割込を発生し、しかも前回の位置からの差分のデータ(Δx , Δy)およびマウスボタンのオン・オフの情報からなるデータを引き渡すという違いがある。そこで、これをエミュレートするマウスエミュレータMUMは、マウス73が動いた時に発生するハードウェア割込により一度呼ばれ、読み取った差分のデータ(Δx , Δy)および前回のマウスの位置から現在のマウスの絶対的な位置を演算し、これを所定のバッファに記憶しておく。加えて、マウスエミュレータMUMは、予めタイマエミュレータTIMに対して所定時間毎に割込を発生するように依頼しておく。タイマエミュレータTIMから所定のインターバルで割り込みコントローラエミュレータIRMに対して割込が生じ、この割込を受け付けた割り込みコントローラエミュレータIRMは、これをPC-9800機のマウスのための割込と認識してカーネルKRに対して仮想割込を発生する。この仮想割込によりカーネルKRはMUMを再度呼び出す。マウスエミュレータMUMは、この呼び出しに応じて先に計算しておいたマウスの絶対的な位置をバッファから読み出し、これを返す。この結果、アプリケーションプログラムAPPは、所定のインターバルでマウスからの絶対位置のデータを受け取ることができる。

30

40

【0090】

以上例示したキーボードエミュレータKEMやマウスエミュレータMUMは、いずれはハードウェア割込を利用するものであるが、割込を利用しないモジュールでは、例外処理の発生→カーネルKRからディスパッチャDPを介した実行モジュールの呼び出し→各実行モジュールでのエミュレート(必要ならモジュール間通信)→処理後のパラメータを持ってアプリケーションプログラムAPPに復帰、という手順で処理がなされることになる。

【0091】

更に、実際にDOS/V機上でのハードディスク84への読み込みを仮想PC上での読

50

み込みとしてエミュレートする様子を詳しく説明する。ハードディスク84は、図2に示した複合I/Oポート54に接続されている。この複合I/Oポート54は、各種のI/O機器が接続される入出力制御用バスであるISAバス42を構成しているものの一つである。またハードディスク84の物理的な構造を図19に示す。ハードディスク84の物理的な構造は、図19に示すようなヘッド、シリンダ、セクタという単位に分かれている。このハードディスク84をアクセスするには「シリンダAとヘッダBで決まるセクタC」といった物理アドレスか、あるいはハードディスク84の先頭セクタから通し番号を付けて「先頭からD番目のセクタ」といった論理アドレスを用い、このいずれかを指定することでアクセスするセクタを決定している。ヘッド数、シリンダ数、セクタ数などの物理的な構成単位は、ハードディスクの種類によって異なっているため、論理アドレスによって読み込むセクタを決定するのであれば、ハードディスクの構成の違いを吸収することができるが、物理アドレスにより、アクセスするセクタを指定する場合には、ハードディスクの構成の違いを考慮しなければ、正しいセクタをアクセスすることはできない。これらの物理的な構成単位は、ハードディスクのシステム管理情報として、オペレーティングシステムの管理下に置かれている。

10

【0092】

図20は、ハードディスク84への読み込みを仮想PCにおけるハードディスクへの読み込みとしてエミュレートする様子を示す説明図である。図は、上方から仮想PC（ターゲットマシン）、エミュレーション実行部、DOS/V機（実行マシン）のハードウェアの各層に分けて描いてある。仮想PCとは、PC-9800機用に仮想的に用意されたメモリ空間であって、タスク自身としては、あたかもMS-DOSが用意するPC-9800機の実空間で処理を行なっているように処理を実行できる空間を意味している。但し、この時点でCPU21は仮想86モードで動作しているので、仮想PCにおいてアプリケーションプログラムAPPが特権命令などを実行しようとする、直ちに例外処理として、タスクは切り換えられ、カーネルKRの処理に移行するのである。

20

【0093】

アプリケーションプログラムAPPの実行中に仮想PCのディスク__Basic__Input__Output__System（以後DISK-BIOSと呼ぶ）を利用してハードディスク84への読み込みが行なわれると、一般保護例外が発生し、例外処理としてタスクが切り替えられてカーネルKRに処理が移行する。ここで、DISK-BIOSは本エミュレータによって供給されており、PC-9800機のBIOS-ROMを代替するように用意されている。BIOSを呼び出したときに、一般保護例外が発生してカーネルに処理を移行するには、いくつかの手法が考えられる。一つは、BIOSプログラムの中で実際にI/Oにアクセスしたとき、一般保護例外が発生してカーネルに処理が移行する構成である。もう一つは、BIOSの所定のルーチンを呼び出した時点で一般保護例外を起こしてカーネルに処理を移行する構成である。ハードウェアのレベルでエミュレーションを行なう場合（例えば上述したキーボードエミュレータKEM等）は、前者の構成を採っているが、ハードディスクのアクセスなど、実際のアクセスまでにトラックやシリンダ、セクタの指定など種々の処理を必要とする場合には、BIOS全体をエミュレートした方が良い場合が存在する。以下に説明するハードディスクエミュレーションでは、BIOSが呼び出されたときに、一般保護例外を生じさせて、エミュレーションを行なっている。この場合には、BIOSの中のハードディスクのアクセスに関する処理の先頭に「HALT」などの一般保護例外を引き起こす命令を用意しておく。アプリケーションプログラムAPPからハードディスクアクセスのためにBIOSがコールされると、呼び出されたPC-9800機用のBIOSの先頭がアクセスされたとき、直ちに処理はカーネルに移行し、エミュレータが起動される。カーネルKRは、各レジスタの値などから自己が起動された原因を調べ、アクセスされたアドレスやI/Oが何かを判断し、ディスパッチャDPを呼び出す。

30

40

【0094】

ディスパッチャDPは、カーネルKRから受け取った情報を元にして呼び出す実行モジ

50

ユーラを特定し、その呼出アドレスをディスパッチャーブルDTから取得する。ハードディスクへ84へのアクセスを行なうBIOSが呼び出された場合には、ディスパッチャDPは、この呼出アドレスを用いて、実行モジュールであるハードディスクエミュレータHDMを呼び出すのである。なお、仮想PCにおけるBIOSが呼び出されたときに、これに等価なDOS/V機用のBIOSを直接呼び出さないのは、後述するように、ハードディスクのアクセスにおいては、ハードディスクの物理アドレスの変換などの処理が必要となるからである。

【0095】

ハードディスクエミュレータHDMは、図21に示すように、ハードディスク84のパーティション領域、システム領域、DOSブート領域といった各領域のデータをDOS/V機用からPC-9800機用に作成する初期化部分、PC-9800機のDISK-BIOSのパラメータから読み込むセクタの物理アドレスを論理アドレスに変換する仮想PC用DISK-BIOS処理部、変換した仮想PC用の論理アドレスをDOS/V機用に変換するアドレス変換部、変換されたDOS/V機用のアドレスをBIOSパラメータにセットするAT-BIOS処理部を備えたHDDエミュレータ部分と、ハードディスク84のシステム領域データ、パーティション領域データ、DOSブート領域データ及びローカルデータを備えたディスク情報ブロックとから構成されている。以下、ハードディスクエミュレータHDMの処理について詳しく説明する。まず最初に、システム領域、パーティション領域、DOSブート領域のDOS/V機とPC-9800機との違いについて説明する。

【0096】

図28にPC-9800機が管理するハードディスクのシステム領域のマップを示す。図28に示すようにPC-9800機のシステム領域は、ハードディスクの先頭セクタ（シリンダ0、ヘッド0、セクタ1）に固定されており、計512バイトの領域が確保されている。システム領域の先頭から3バイト目以降の内容が#IPL1となっており、PC-9800機は、起動時にハードディスクのこの部分を参照する。一方、DOS/V機のシステム領域の位置は、PC-9800機同様ハードディスクの先頭セクタではあるが、PC-9800機のように起動時に参照されるデータが格納されているわけではなく不定である（PC-9800機のように3バイト目以降の内容が#IPL1となっている訳ではない）。よってこのようなシステム領域では、OSカーネル、アプリケーションプログラム等がどこを参照すれば良いか固定的に決めることができない。したがって、そのままでは、エミュレーション装置を構成し、DOS/V機用のハードディスク84を仮想PC上で認識させる、ということができない。そこで、この実施例のエミュレーション装置では、エミュレーションを行なう際には、専用のアプリケーションを実行することにより、DOS/V機用のシステムファイルなどの名称を変更すると共に、DOS/V機用のハードディスク84の先頭セクタから3バイト以降の内容を、強制的に図28のように書き直している。なお、書き直される以前にこのセクタに置かれていた内容は、予め別の場所に退避しておき、エミュレーションを行なわない設定を変更する場合に、専用のアプリケーションプログラムを実行することにより、システムファイルの名称を元に戻すと共に、退避したデータを書き戻している。

【0097】

次に図24、25を用いてDOS/V機のパーティションについて説明する。DOS/V機のハードディスク84の各ドライブは、図25に示すように、パーティション領域をそれぞれ持っている。それぞれのパーティション領域は、図24に示すように2つのパーティション情報を格納している。例えば図24におけるパーティション領域の上半分のパーティション情報（パーティション1）が、図25においてデータ領域であるドライブ#1の開始及び最終アドレスの情報であるとする、図24におけるパーティション領域の下半分のパーティション情報（パーティション2）は、図25において次のデータ領域であるドライブ#2のパーティション情報が格納されているパーティション領域の開始及び最終アドレスの情報になっている。それぞれのドライブのこうしたパーティション領域は

10

20

30

40

50

、図 25 に示すように、ポインタにより順次リンクされている。

【0098】

一方、PC-9800 機のパーティションについて図 26、27 を用いて説明する。PC-9800 機のハードディスクのマップは、図 27 に示すように、先頭セクタから、システム領域、パーティション領域、OS 領域、DOS ブート領域、データ領域となっている。ここで重要なのは PC-9800 機のパーティション領域は図 25 の DOS/V 機のようにパーティション領域がリンクされておらず、すべてのドライブのパーティション情報が図 27 のようにシステム領域の後に格納されていることである。更にパーティション領域に格納されているそれぞれのパーティション情報は、図 26 のようにシステムが登録されているパーティションかどうかを表すセクタ、パーティションの OS 及びファイルシステムを表す状態、BOOT プログラムのアドレス、開始アドレス、最終アドレス、OS_ID 名という順になっている。これに対して、DOS/V 機では、図 24 に示したように、パーティション情報が、起動可能なパーティションかどうかを表すブートインジケータ、開始アドレス（開始ヘッド、開始セクタ、開始シリンダ）、パーティションの OS 及びファイルシステムを表すシステムインジケータ、最終アドレス（最終ヘッド、最終セクタ、最終シリンダ）、ブートセクタの相対値、パーティション内のセクタ数という順になっており、PC-9800 機とはパーティション情報の形式や内容、配置が異なっている。また各パーティションの開始アドレスが DOS/V 機の場合はシリンダ 0、ヘッド 1、セクタ 1 となり、PC-9800 機の場合はシリンダ 1、ヘッド 0、セクタ 1 と異なっていたり、PC-9800 機で使用しているヘッド数、セクタ数が DOS/V 機で使用しているヘッド数、セクタ数の値と一致しない場合がある。

【0099】

以上のことから、仮想 PC 上で動作しているアプリケーションプログラムからハードディスクをアクセスする際、DOS/V 機のパーティション領域をそのまま利用することはできない。即ち、各々のドライブのアクセスしたいデータにアクセスできず、DOS/V 機用のハードディスク 84 を仮想 PC 上で認識させることができない。そこで DOS/V 機のパーティション領域の開始及び最終アドレスを PC-9800 機用にアドレスを変換し、図 26 に示すようなパーティション領域を作成する必要がある。

【0100】

最後に図 29 に PC-9800 機の DOS ブート領域のマップを示す。DOS/V 機の DOS ブート領域も図 29 とほぼ同じ構成になっている。しかし、PC-9800 機と DOS/V 機とでは図 29 の幾つかの項目で異なる部分があるため DOS/V 機の DOS ブート領域の幾つかの項目を書き直す必要がある。具体的には図 29 において DOS ブート領域の先頭から 18 H バイト目のトラック当りのセクタ数や 1 A H バイト目のヘッド数、1 E H 及び 4 0 H バイト目のハードディスクの先頭セクタから DOS ブート領域までのセクタ数、4 2 H バイト目の OS 領域のサイズ数、4 4 H バイト目の物理セクタ長、4 6 H バイト目のデバイスコード、4 7 H バイト目のコマンドコードである。

【0101】

以上説明したように、ハードディスクエミュレータ HDM では、前述したシステム領域データ、パーティション領域データ、DOS ブート領域データの違いから、それぞれの領域を DOS/V 機用から PC-9800 機用に作成するといったハードディスク 84 の初期化処理を、実行時の最初の 1 回のみ行なっている。以下、図 22 のフローチャートを用いてハードディスク 84 の初期化処理を説明する。

【0102】

ハードディスクの初期化処理部（ステップ S400）では、まず DOS/V 機のハードディスクの先頭セクタであるシステム領域を取得する（ステップ S410）。システム領域の先頭から 1 B E H バイト目以降に格納されているパーティション情報を取得する（ステップ S420）。取得したパーティション情報の開始及び最終アドレスを PC-9800 用の開始及び最終アドレスに変換し（ステップ S430）、図 26 に示した形式でディスク情報ブロックにパーティション領域データとして格納する（ステップ S440）。D

10

20

30

40

50

OS/V機の場合は図25に示したように各パーティション領域がリンクされているので次のパーティション情報がないとの判断がなされるまで(ステップS450)、ステップ410～ステップ440までの処理を繰り返す。以上のようにしてPC-9800機用のパーティション領域を作成する処理を、図21のブロック図におけるパーティション領域データ作成部が行なっている。

【0103】

最後に前述したようにシステム領域の先頭から3バイト目以降の内容に変更を加えて、ディスク情報ブロックにシステム領域データとして格納する処理を行なう(ステップS460)。以上のようにしてPC-9800機用のシステム領域を作成する処理を、図21のブロック図におけるシステム領域データ作成部が行なっている。

10

【0104】

図22のフローチャートには示していないが、DOSブート領域についても前述したようにDOSブート領域のトラック当たりのセクタ数、ヘッド数等に変更を加えて、ディスク情報ブロックにDOSブート領域データとして格納する。以上のようにしてPC-9800機用のDOSブート領域を作成する処理を、図21のブロック図におけるDOSブート領域データ作成部が行なっている。

【0105】

以上のようなハードディスク84の初期化処理を実行時の最初に行い、PC-9800機用に変換したシステム領域、パーティション領域、DOSブート領域をデータとしてディスク情報ブロックに持つ。以後システム領域、パーティション領域、DOSブート領域に対するアクセス時には、初期化処理で作成したデータを読み込めば良く、ハードディスクへのアクセスが速くなる。

20

【0106】

なお、システム領域データ、パーティション領域データ、DOSブート領域データをディスク情報ブロックに格納せずにシステム領域、パーティション領域、DOSブート領域に対するアクセス毎にこれを作成するという構成も可能である。この場合には、各領域データをメモリに格納する必要がないので、メモリの使用を節約することができる。しかし、システム領域、パーティション領域、DOSブート領域に対するアクセス毎にシステム領域、パーティション領域、DOSブート領域の変換および作成が行なわれるので、本実施例に比べてハードディスクへのアクセスが遅くなる。

30

【0107】

次に、ハードディスクエミュレータHDMの実際のエミュレートについてハードディスクからのデータの読み込みを例にとって説明する。まず、大まかに処理の流れを説明する。図20に示したように、仮想PCとしてのタスクで動作しているアプリケーションプログラムAPPがハードディスク84をアクセスしようとして、仮想PC上のDISK_BIOSを呼び出すと、該当するBIOSルーチンの先頭に置かれた「HALT」命令が実行され、特権命令の実行であるとして、一般保護例外によるトラップが発生する。この結果、カーネルKRおよびディスパッチャDPを介して、ハードディスクエミュレータHDMが呼び出される。DOS/V機用のDISK_BIOSには、PC-9800機用のBIOSとほぼ等価なルーチンが用意されているので、ハードディスクエミュレーションHDMは、BIOSレベルでエミュレーションを行なうものとし、後述するアドレス変換などの所定の処理の後、タスクをDOS/V機としてのタスクに切り替えて、そのDISK_BIOSを呼び出す。この結果、実際のハードディスク84に対する処理が行なわれ、ヘッドシークやセクタリードなどの処理が行なわれ、データのやり取りを伴う処理の場合には、DOS/V機のDISK_BIOSが管理するバッファとの間でデータの転送が行なわれる。ハードディスク84から読み出され、バッファに書き込まれたデータは、DOS/V機としてのタスク内に用意されたバッファに書き込まれることになる。このバッファは、ハードディスクエミュレータHDMが固定的に確保している。ハードディスクエミュレータHDMは、このデータを、仮想PC側のバッファに移し、その後、ディスパッチャDP、カーネルKRを介して、仮想PC側に処理を返す。仮想PC側のBIOSコール

40

50

からアプリケーションプログラムAPPに処理が戻った時点では、仮想PC側のタスク内のバッファに、所望のデータが用意されていることになる。なお、仮想PC側のバッファは、仮想PCのDOSが予め用意するものとしても良いし、アプリケーションプログラムAPPが、BIOSコールに先立って動的に確保するものとしても良い。

【0108】

実際に、上記の処理を行なおうとすると、PC-9800機とDOS/V機では、使用できるハードディスクのヘッド数、セクタ数の違い等があるために、仮想PC上で用意したセクタの物理アドレスをそのままDOS/V機上では利用できない。そこで、実際のハードディスクへのアクセスを示す図23に示したように、ハードディスク84に対する読み込みの処理では、まずアドレス変換の処理を行なう。

10

【0109】

この処理では、最初に、仮想PC上で実行しようとしたハードディスクの読み込みが物理アドレスによるものか論理アドレスによるものかを判定する(ステップ500)。ハードディスクの読み込みが物理アドレスによるものである場合には、PC-9800機上のハードディスク情報(ヘッド数、セクタ数、シリンダ数)を用いて論理アドレスに変換する(ステップ510)。具体的には読み込んだシリンダ番号にヘッド数を乗じたものに読み込んだヘッド番号を加える。これに1トラック当りのセクタ数を乗ずることで、今読み込もうとしているシリンダ番号までの総セクタ数が求められる。これに読み込んでいるセクタ番号を加えることでハードディスクの先頭セクタから数えたセクタ番号である論理アドレスを得ることができる。計算によらずとも、変換テーブルによっても容易に、物理アドレスを論理アドレスに変換することができる。以上のように仮想PCにおける物理アドレスをPC-9800機用のハードディスク情報を用いて論理アドレスに変換する処理を、図21のブロック図における仮想PC用DISK-BIOS処理部が行なっている。ただし、仮想PC上でのハードディスクの読み込みが論理アドレスの場合には(ステップ500)、上記のような処理は必要ない。

20

【0110】

次に、変換した論理アドレスを今度はDOS/V機上のハードディスク情報(ヘッド数、セクタ数、シリンダ数)を用いてDOS/V機上のハードディスクに対応した物理アドレスに変換する(ステップ520)。具体的には論理アドレスを最大ヘッド数に最大セクタ番号を乗じたもので割ることで読み込んでいるシリンダ番号が得られる。この除算の余りを最大セクタ番号で割ることで、読み込もうとしているヘッド番号が得られる。更に、この除算の余りに1を加えることで、読み込もうとしているアドレスのセクタ番号が得られる。以上の演算の結果、論理アドレスをDOS/V機の物理アドレスに変換することができる。仮想PCにおける論理アドレスをDOS/V機用のハードディスク情報を用いて物理アドレスに変換する以上の処理を、図21のブロック図における仮想PC用アドレスをAT用の物理アドレスへの変換部が行なっている。

30

【0111】

次に変換した物理アドレスをDOS/V機のDISK-BIOSの機能であるセクタ読み込みの入力パラメータであるセクタ数、シリンダ・セクタ番号、ヘッド番号、ドライブ番号にセットしてDISK-BIOSを実行するためにDOS/V機のDISK-BIOSに処理を渡す。DOS/V機のDISK-BIOSの実行は図20のDOS/Vハードで行なわれるので、この時点で図20におけるDOS/Vハードにタスクが切り替わる。DISK-BIOSでは入力パラメータに従ってハードディスクの指定されたセクタの内容が読み込まれ、その内容がバッファに格納される(ステップ530)。図20においてDOS/Vハードのタスクでは必要なBIOSの処理を実行し終了すると、ハードディスクエミュレータHDMの処理にタスクが切り替わる。ハードディスク84の指定されたセクタの内容がバッファに読み込まれるとタスクが切り替わりハードディスクエミュレータHDMに処理が渡されるのである。ハードディスクエミュレータHDMでは読み込んだセクタに対して以下の処理を行なう。

40

【0112】

50

読み込んだセクタの内容がシステム領域かあるいはパーティション情報であるか判定する（ステップ540）。読み込んだセクタの内容がシステム領域あるいはパーティション情報であれば、これはDOS/V機のデータであって仮想PC側が必要としているPC-9800機用のデータではないから、先程の初期化処理で作成されたPC-9800機用のシステム領域データあるいはパーティション領域データをディスク情報ブロックから読み出す（ステップ541）。読み込んだセクタの内容がシステム領域かあるいはパーティション情報でなければ、次にDOSブート領域かどうか判定する（ステップ550）。読み込んだセクタの内容がDOSブート領域であれば、これも仮想PC上の処理が必要としているPC-9800機用のDOSブートのデータではないから、先程の初期化処理で作成されたPC-9800機用のDOSブート領域データをディスク情報ブロックから読み出す（ステップ551）。上記いずれの領域情報でもなければ、ハードディスク84から読み出しておいた内容は何等変更しない。

10

【0113】

以上の判断の後、データを、仮想PC用のメモリ空間へ書き込む（ステップ560）。従って、本実施例では、システム領域、パーティション領域あるいはDOSブート領域に対するアクセスであれば、ディスク情報ブロックに記憶したデータにより書換えたデータを、そうでなければハードディスク84から読み出したデータが、仮想PC用のメモリ空間に書き込まれ、仮想PC側に引き渡される。したがって、仮想PC側からは、あたかもPC-9800機のハードディスクをアクセスしているかのようにハードディスクのアクセスを実行することができる。この実施例では、システム領域などに対するアクセスであっても、必ず実際にハードディスク84をアクセスするので、ハードディスク84に何等かのトラブルが発生した場合でも、即座にその情報を得ることができるのである。

20

【0114】

以上の読み込み処理が終了するとハードディスクエミュレータHDMは、ディスパッチャDPを介してカーネルKRへと処理を戻し、更に例外処理を引き起こした処理まで戻っていく。この結果、仮想PC上で読み込まれたセクタに該当する内容がハードディスクエミュレータHDMによってDOS/V機上のハードディスク84から読み込まれ、仮想PC用のメモリ空間に書き込まれる。その後、仮想PC上のアプリケーションプログラムAPPがこの内容を受け取り処理を行なうことになる。以上の結果、仮想PC上に展開されたPC-9800機用のアプリケーションプログラムAPPはPC-9800機上のハードディスクへ読み込みをあった場合と同一の動作をすることになる。すなわち、実施例のエミュレーション装置は、PC-9800機のハードウェアとその動作を完全にエミュレートするのである。

30

【0115】

以上のようにハードディスクエミュレータHDMは、仮想PC上のタスクの実行→BIOSコールによる例外処理の発生→カーネルKRからディスパッチャDPを介した実行モジュールの呼び出し→各実行モジュールでのDOS/V機用のタスクでのBIOSコールを含むエミュレート→処理後のパラメータを持って仮想PCタスクのアプリケーションプログラムAPPに復帰、という手順で処理がなされることになる。

【0116】

尚、本ハードディスクエミュレータHDMは、カーネルKR、ディスパッチャDP等からなるエミュレーションシステムの一部としての動作に加え、単独での動作も可能である。具体的には、初期化時にディスク情報ブロックに98用HDDの情報を作成する動作は、SMI（システム・マネジメント・インタラプト：インテルのCPUに備えられた最優先割り込み）等で本HDDエミュレータを呼び出して行ない、以降のアドレス変換はハードウェアにより高速に行なうことも可能である。即ち、エミュレーションの主操作をハードウェアによって行い、一部に本エミュレーションシステムの機能を埋めこむことも有効である。

40

【0117】

本実施例によりDOS/V機上のハードディスク84をPC-9800機用に改めてフ

50

フォーマット（初期化）することなく、DOS/V機のハードディスク84をその状態のまま利用することができる。そのため、ハードディスク上の資源を共有化することが可能である。また本実施例では、ハードディスク84の初期化処理を、ハードディスク84に対する最初のアクセス時に行い、PC-9800機用に変換したシステム領域、パーティション領域、DOSブート領域をデータとしてディスク情報ブロックに持つようにしている。したがって、システム領域、パーティション領域、DOSブート領域に対するアクセス時には初期化処理で作成したデータを読み込めばよく、前記領域へのアクセス毎にデータをPC-9800機用に変換する必要がないので、ハードディスクへのアクセスが速くなる。

【0118】

10

以上説明した本実施例のエミュレーション装置によれば、実行マシン上でターゲットマシンをエミュレーションするプログラムが、カーネルKR、ディスパッチャDP、各実行モジュールと階層化されており、極めて見通しの良い構成となっている。従って、膨大な機能をエミュレートする膨大なプログラムを効率よく開発することができる。しかも、各実行モジュールはハードウェアの機能に近いところでモジュール化されているので、物理アドレスをハードディスクを読み書きするなどハードウェアに直接アクセスするようなアプリケーションプログラムAPPであっても、エミュレーションを正しく行なうことができる。

【0119】

次に、本発明の他の実施例について説明する。この実施例もハードディスクのエミュレーションに関するものである。この実施例が、上述した実施例と異なる点をまず説明する。上述した実施例では、ハードディスク84の先頭アドレスの内容を強制的に書き直してしまうため、同一のハードディスク84から、PC-9800機として立ち上げるかDOS/V機として立ち上げるかを設定する際には、オペレーティングシステムがその組み込み時に必要とするファイルを共存させることができない。例えば、MS-DOSでは、その組み込み時に、「config.sys」や「command.com」など、決まった名称のファイルを必要とするが、PC-9800機としてのMS-DOSでも、DOS/V機としてのMS-DOSでも、同一の名称のファイルをルートディレクトリに必要とすることは変わらない。この場合、アーキテクチャの異なる機種では、これらのファイルの中身は当然異なったものとなるので、共通化を図ることはできない。

20

30

【0120】

そこで、上述した実施例では、エミュレーションを行なう場合には、セットアッププログラムを実行してオリジナルのファイルの拡張子を変更し、ターゲットマシンのためのファイルをデフォルトとし、逆にエミュレーションを行なわない場合には、セットアッププログラムにより両者の拡張子をそれぞれ元に戻していた。しかし、こうした手法では、ファイル名称などをユーザーが変更したりすると、セットアップを正常に行なうことが困難になってしまう。また、複数のハードディスク装置が接続されている場合、あるいは複数のパーティションが設けられている場合、これを仮想マシン上でどのように認識するかという点については、十分な対応がとられていなかった。以下に説明する実施例では、上記実施例におけるハードウェアエミュレーションを更に改良し、オペレーティングシステム

40

【0121】

この実施例のハードウェア構成は、上述した実施例と同様である（図1および図2参照）。この実施例では、エミュレーション装置を成立させるために、図30に示したインストールプログラムが、エミュレーション処理に先立って実行される。このプログラムは、DOS/V機がその正規のオペレーティングシステムの下で動作している状態で実行され、仮想PCとしてのPC-9800機の環境をセットアップするプログラムである。この処理が開始されると、まずDOS/V機に接続されているハードディスク84の空き容量

50

などをチェックする処理を行なう（ステップS600）。続いて、最大の連続する空き領域を取得する処理を行なう（ステップS602）。これは、ハードディスク84の空き領域のうち、連続する領域を確保するためのものであり、連続した空き領域が例えば40Mバイトと50Mバイトあれば、50Mバイトが取得される。

【0122】

次に、この連続空き領域を含む情報の表示を行ない（ステップS604）、各種の設定を入力する処理を行なう（ステップS606）。CRT76に表示されるこの時の設定画面の一例を図31に示す。仮想PCセットアッププログラムは、仮想PC用のプログラムをインストールするドライブやそのサブディレクトリの名称、およびステップS602で取得した最大の空き領域（図31では10Mバイト）等を表示する。使用者は、この画面を見ながら、カーソルキーを操作して設定しようとする項目を選択し（反転状態となるが、図示では四角の枠で囲った）、ディレクトリ名等を設定する。なお、最大空き領域の大きさを越えない範囲であれば、仮想PCのMS-DOSの領域は、必要容量以上の所望な大きさに設定することができる。

10

【0123】

所望の設定となるまで、設定処理を繰り返し、画面下の「インストールして終了」が選択されると（ステップS608）、インストールの処理が行なわれる（ステップS610）。この処理は、まず設定された連続する空き領域をMS-DOS上のファイルとして確保する処理を行なう。連続した空き領域を確保するために、ファイルアロケーションテーブル（FAT）を直接書き換え、MS-DOSが管理しているディレクトリの情報と結びつける。空き領域を確保するために作成されるこのファイルの属性は、不可視ファイル（Hidden）でリードオンリーである。更に、このファイル（以下、仮想PCシステムファイルと呼ぶ）の中に、DOSのブートセクタ情報、FAT領域、ディレクトリ領域、データ領域のディスクイメージを作成し、あたかもPC-9800機のハードディスクとしてフォーマットされたかのようにデータを書き込む。その後、このフォーマットした仮想PCシステムファイルをディスクとしてアクセスするために、仮想ドライブドライバをメモリにロードし、この仮想ドライブドライバが管理する仮想PCシステムファイルのルートディレクトリの領域に、DOSのサブディレクトリを作成する。

20

【0124】

以上の処理の後、セットアッププログラムによるセットアップの内容を、所定のファイルに書き出し、その後、PC-9800機の機能をDOS/V機上でエミュレートするのに必要な各種ファイルをハードディスク84の予め定めたサブディレクトリに転送する。続いて、DOS/V機用に用意されたMS-DOSのシステムファイルである「config.sys」というファイルの先頭に、エミュレーションを実現するデバイスドライバ（実施例では「NSELECT.SYS」）を記述する処理を行なう。従って、インストールが行なわれ後では、コンピュータが起動されると、まずDOS/V機としてのブートが行なわれてBIOSなどがロードされた後、DOS/V機用のシステムファイルである「config.sys」を参照してデバイスドライバを組み込むが、その段階で、このデバイスドライバ「NSELECT.SYS」を組み込む処理が実行されることになる。このデバイスドライバの組込の処理により、エミュレーションのための処理が開始されることになる。このデバイスドライバを組み込む処理が行なわれると、DOS/V機用のファイル「config.sys」に記述されている他のデバイスドライバは一切組み込まれないことになる。

30

40

【0125】

システムファイル「config.sys」を書き直したところで、インストールの処理をすべて終了として、インストール処理ルーチン（図30）を抜けて、DOSのプロンプト状態に戻り、待機する。

【0126】

PC-9800機のエミュレーションを行ないたい場合には、以上説明したインストールを行なった後、実施例のDOS/V機をリセットないし電源断—再投入を行なう。この

50

とき、DOS/V機は、図32に示した電源投入時処理ルーチンを実行する。この処理が開始されると、コンピュータは、DOS/V機としての通常の初期化の処理（ステップS620）、BIOSの展開の処理（ステップS622）を行なう。通常、即ちエミュレーションの機能がオンになっていなければ、コンピュータは、そのまま、システムファイル「config.sys」を参照してデバイスドライバを組み込む処理（ステップS624）に進む。しかし、実際には「config.sys」というシステムファイルの先頭には、エミュレーションの機能を記述したデバイスドライバ「NSELECT.SYS」を組み込むための情報が記載されているので、処理は、このデバイスドライバの組込の処理（図32、ステップ630ないしステップS660）に移行する。なお、このデバイスドライバ「NSELECT.SYS」を組み込む処理の中で、他の実行ファイルが呼び出すことも可能である。実施例では、「NSTART.EXE」というプログラムを呼び出して実行している。

10

【0127】

即ち、図32に示した処理のうち、右側の欄は、DOS/V機のファイル「config.sys」の先頭に書き込まれたデバイスドライバ「NSELECT.SYS」を組み込む処理および所定の実行ファイル「NSTART.EXE」の処理として実現される。まず、エミュレーション装置を構築する処理を行なう（ステップS630）。これは、どんなアーキテクチャの機種をエミュレートするかを決定するものである。次に、エミュレーションを行なうターゲットマシンの仮想BIOSを読み込み（ステップS632）、ターゲットマシン用に確保されたプロテクトメモリ上に展開する（図7参照）。ターゲットマシン用のこのメモリ領域は、タスクの切換により実アドレスに割り当てられる。したがって、DOS/V機のBIOSの展開（ステップS622）とターゲットマシンの仮想BIOSの読み込み（ステップS632）とが行なわれた後では、DOS/V機用のBIOSは実アドレスに割り当てられ、ターゲットマシンであるPC-9800機用のBIOSはリアルモード空間に仮想的に割り当てられる。両BIOSは、タスクの切換により、それぞれのタスクアドレス空間で実行可能となる。仮想86モードを用いることにより、リアルモードで動作可能に作られているこれらの各BIOSが、1Mバイトを越えるプロテクトモード空間でも使用可能となる。また、それぞれのBIOSによるファイルのアクセスなどに必要なバッファもそれぞれのタスク空間に用意される。

20

【0128】

更に、この仮想BIOSを用いて仮想システムファイルをハードディスク（HDD）としてアクセスし（ステップS634）、エミュレーションを行なって動作するシステムとしての初期化を行なう（ステップS640）。この初期化の処理については、図33以下のフローチャートを用いて、後述する。その後、ターゲットマシンとしてのブートを行なって（ステップS650）、ターゲットマシンとして立ち上がる（ステップS660）。ターゲットマシンとして立ち上がる場合には、第1実施例で説明したように、カーネルKRやディスパッチャDP、各種エミュレーションモジュールなどが組み込まれることになる。これらの詳細は、第1実施例と同様なので、説明は省略する。

30

【0129】

次に、初期化の処理（ステップS640）について説明する。初期化処理は、図33に示すように、大きくは、仮想システムファイル情報の取得および作成（ステップS700、詳細は図38）、パーティションの並び替え（ステップS720、詳細は図39）、仮想ハードディスクの作成（ステップS740、詳細は図42）、アドレス調整（ステップS760）、オフセットテーブルの作成（ステップS780、詳細は図44）から構成されている。まずこれらの処理の詳細を説明する前に、初期化の処理がなされた後の全体構成および仮想PCシステムファイルの内容などについて説明する。

40

【0130】

このハードディスクエミュレータHDMは、図34に示すように、実際にエミュレーションの処理を行なうHDDエミュレータ部と、エミュレーションのための情報を記憶しているディスク情報ブロックから構成されている。HDDエミュレータ部は、図21に拠っ

50

て第1実施例で説明したBIOS処理部などに加えて、新たに、仮想PCシステムファイルの情報を作成する仮想システムファイル情報作成部、仮想的なハードディスクを構成する仮想HDD作成部、ハードディスクのアドレス変換のためのオフセットテーブルを作成するオフセットテーブル作成部が設けられている。これらの作成部は、本実施例で、仮想的なハードディスクをファイルとして、DOS/V機用のハードディスク84の内部に確保しているため必要となったものである。図34には、この仮想的なハードディスクに相当するファイルである仮想システムファイルを確保する手段を、仮想システムファイル領域確保部VSFとして、示した。この仮想システムファイル領域確保部VSFは、ハードディスク84内の最大の連続空き領域を取得し（図30、ステップS602）、インストールの指示を受けて（同図ステップS608）、インストールすることにより（ステップS610）、ハードディスク84内に確保するものである。

10

【0131】

ハードディスクエミュレータHDMのディスク情報ブロックは、エミュレーションに必要な情報をハードディスクエミュレータHDMが管理するプロテクトメモリ上に記憶しているものであり、初期化の処理を通じて必要なデータが構成されている。ここに記憶されているデータは、ターゲットマシンとして処理を行なうために必要なデータであり、システム領域の情報、DOSのブート領域の場所に関するデータ、パーティション領域のデータ（複数のハードディスクのパーティションの並びに関するデータ）、オフセットデータ（実行マシンに実際に接続されたハードディスクと仮想ディスクとの間のアドレス変換を行なうためのデータ）、ローカルデータが記憶されている。PC-9800機用のハードディスクとしてエミュレートされるものは、実際には、DOS/V機用のハードディスク84の内部にファイルとして確保された連続領域である。この領域を仮想PCシステムファイルと呼ぶが、図35に示すように、ハードディスクのイメージでフォーマットされている。この領域の先頭には、DOSブート領域までのオフセットが記録されており、そこから実際のDOSのブート領域まではダミーデータが記録されている。ダミーデータを記録しているのは、後述するアドレス変換のためであり、DOSブート領域の先頭の論理アドレスが所定の条件を満たすように、ダミーデータの領域の大きさを調整している。DOSブート領域の後にFATとディレクトリの領域が設けられており、最後がデータ領域となっている。

20

【0132】

図35に示すように、連続領域として確保された仮想PCシステムファイルは、その先頭の2バイトに、DOSブートの先頭位置までのオフセットが記録されている。また、オフセット01BF以下に、開始ヘッド番号およびシリンダ・セクタ番号STとして、図35に示した仮想PCシステムファイル内のDOSブート領域の先頭位置の情報が、オフセット01C3以下に、終了ヘッド番号および終了シリンダ・セクタ番号EDとして、DOSブート領域の終了位置の情報が、それぞれ記録されている。従って、ブート時には、このデータを用いてDOSブート領域からの読み込みを開始することができる。DOSブート領域の内容を、図36に詳細に示した。このDOSブート領域には、この領域の基本的な情報が登録されており、オペレーティングシステムをブートする場合に用いられる。例えば、ブートインジケータには、このブート領域がアクティブか否かの情報が、またシステムインジケータには、このパーティションを使用しているオペレーティングシステムの種類（領域の休止／使用、OSの種類、12ビット／16ビットセクタアクセス等のファイルシステム種類）が記録されている。ブートローダは、これらのデータを用いて、ハードディスク84に確保された領域からデータを読み込むことでDOSのブート（図32ステップS650）を行なう。

30

40

【0133】

次に、パーティションについて説明する。DOSは、接続されているハードディスク毎にドライブデータテーブルを用意しており、このデータは、所定のBIOSファンクションにより読み取ることができる。ハードディスク84用のドライブデータテーブルの一例を、図37に示した。図示するように、ドライブデータテーブルの先頭には、次のテーブ

50

ルを示すポインタ P N T が置かれており、複数のハードディスクが接続されていたり、一つのハードディスクであっても複数のパーティションが設けられている場合には、その数だけの情報が、ポインタ P N T を介して並べられている。したがって、B I O S のファンクションコールにより、ポインタを順次辿って行くことができ、総てのハードディスクのパーティション情報を読み取ることができる。ハードディスクエミュレータ H D M は、組み込み時の初期化の処理において、このドライブデータテーブルの情報を読み取り、ターゲットマシン用のパーティション情報（図 2 6，図 2 7 参照）を作成し、ハードディスクエミュレータ H D M のディスク情報ブロック（図 3 4 参照）のパーティション領域データに記憶する。

【0134】

10

図 3 3 に示した初期化の処理の詳細について、順次説明する。図 3 8 に示したように、仮想 P C システムファイル情報作成処理では、まず仮想 P C システムファイルのパス名とファイル名を取得する処理を行なう（ステップ S 7 0 2）。これらのパス名とファイル名は、図 3 0 に示したインストールの処理において作成される所定のデータファイルに書き込まれている。このファイルからパス名とファイル名を取得し、次に仮想 P C システムファイルをアクセスし、D O S ブート領域の情報を取得する処理を行なう（ステップ S 7 0 4）。D O S ブート領域の開始アドレス S T 等は、オフセットのデータから求めても良いが、この実施例では直接取得可能に用意されていることは既に説明した。

【0135】

更に、仮想 P C システムファイルのパーティション情報（D O S / V 用）を取得する処理を行なう（ステップ S 7 0 6）。仮想 P C システムファイルの D O S ブート領域には、この仮想 P C システムファイル自身の情報があり、これを用いてパーティションの情報を作成することができる。但し、ここでは D O S / V 機のパーティション情報に合わせたフォーマットで作成する。即ち、仮想 P C システムファイルに記録されているハードディスクのドライブ番号を取得し、ディスク情報ブロック（図 3 4 参照）のオフセットテーブルデータに格納し、既述したシステムインジケータを D O S ブート領域から取得してディスク情報ブロックのパーティション領域データに格納する。更に、仮想 P C システムファイルの開始アドレス S T 等もパーティション領域データに格納する。

20

【0136】

こうして得られた仮想 P C システムファイルのパーティション情報を次に仮想 P C 用のアドレスに変換する処理を行なう（ステップ S 7 0 8）。第 1 実施例で説明したように、D O S / V 機と P C - 9 8 0 0 機では、ハードディスクの物理的なフォーマットの形式も異なっている。本実施例のエミュレーション装置は、D O S / V 機の上で P C - 9 8 0 0 機の環境を実現するものなので、まず D O S / V 機上でのパーティション情報を作成し、これを他のハードディスクのパーティション情報と共に、併せてターゲットマシンである P C - 9 8 0 0 機用のパーティション情報に変換するのである。変換したパーティション情報は、ディスク情報ブロックに格納される。

30

【0137】

以上の処理を、仮想 P C システムファイルの数だけ行ない（ステップ S 7 1 2）、総てのシステムファイルについて情報の取得・作成処理が完了すれば、「E N D」に抜けて、本処理を終了する。通常、仮想 P C システムファイルは、一つと考えられるが、複数の P C - 9 8 0 0 機の環境を共存させることも可能である。こうした環境の一つとしては、例えば P C - 9 8 0 0 機に用意されたディスクベースシステム等も含まれる。ディスクベースシステムは、M S - D O S に代えて、B A S I C プログラムを動作させる独自の環境が立ち上がるものである。この場合には、ファイルアクセスの単位である 1 セクタのバイト数も M S - D O S とは異なる。したがって、ハードディスクエミュレータ内部で、一度に読み書きするバイト数を、B A S I C のシステムのバイト数に擬似的に合わせるエミュレーションを行なえば良い。

40

【0138】

次に、図 3 9 に拠ってパーティションの並び替え処理について説明する。複数のパーテ

50

ィションが存在する場合、DOS/V機では、DOSが管理しているハードディスクについて、各パーティションの情報を格納したテーブルをポインタPNTで繋いでいる。図40に、この様子を例示する。MS-DOSでは、ハードディスクが複数のパーティションを持つ場合、少なくともその一つのパーティションにはシステムファイルが存在する必要がある。システムファイルが存在するパーティションを基本DOS領域、存在しない領域を拡張DOS領域と呼ぶ。DOSの起動時には、最初、基本DOS領域が認識され、その後、拡張DOS領域が認識される。従って、仮想PCシステムファイルをハードディスクとして認識するハードディスクエミュレータHDMでは、先頭（論理ドライブA：）をこの仮想PCシステムファイルとすると、その後のパーティションの並びを、このDOS/V起動時の並びにするのが自然である。この様子を図41に示した。同図（a）に示すように、例えば2台のハードディスク84が接続されており、内部がそれぞれ2つにパーティションされており、更に1台目のハードディスクに仮想PCシステムファイルが存在したとすると、同図（b）に示したように、仮想PCシステムファイル→基本DOS領域→拡張DOS領域の順にパーティションは配列されることになる。このように配列すれば、DOS/V機を使用していたときのハードディスクの並び（DOS/V機では、ハードディスクは論理ドライブC：から順に割り付けられる）とほぼ同一の並びとなり、DOS/V機上で、PC-9800機をエミュレートする使用者の違和感が小さくなる。

10

【0139】

このようなパーティションの並び替えは、ハードディスクエミュレータHDMの組み込み時に、パーティションの情報を読み取ることで容易に行なうことができる。ハードディスクにおけるパーティションの情報は、図37に示したドライブデータテーブルを読み取ることで取得することができる。本実施例では、電源投入直後には、DOS/V機として立ち上がり、そのBIOSを展開してから、デバイスドライバの組み込みの時点でエミュレーションに移行する。従って、タスクを切り替えてDOS/V機のBIOSのファンクションコールを使用可能とし、必要なBIOSファンクションを呼び出して、所望のデータを取得することができる。呼び出したBIOSファンクションにより得られたデータは、DOS/V機のBIOSのタスクが管理するメモリ上に格納されるが、カーネルKRを介して、このデータを初期化の処理において読み取ることは容易である。ファンクションコールの一つを用いることで、ドライブデータテーブルを取得することができる。

20

【0140】

本処理ルーチンが起動されると、図39に示すように、ファンクションコールを利用して、まずパーティションが存在するハードディスクのドライブデータテーブルが存在するブロックを取得する（ステップS722）。次に、このドライブデータテーブルの内容から、ドライブ番号を読み取って、ディスク情報ブロックのオフセットデータテーブルに格納する（ステップS724）。次にハードディスクのパーティション情報を読み取り（ステップS726）、これを仮想PC用のパーティション情報に変換する。この変換は、パーティションの開始アドレス（シリンダ番号）をドライブデータテーブルに記録された絶対シリンダ番号から取得し、その開始アドレス（ヘッド番号：1，セクタ番号：1）を用いて、仮想PC用の開始アドレス（物理アドレス）に変換するのである。更に、ドライブデータテーブルに記録されたパーティションの総セクタ数を取得し、これを上記の開始アドレス（論理アドレス）に加算する。加算されたアドレスから、仮想ハードディスクの情報（ヘッド数、セクタ数）を用いて、ターゲットマシンであるPC-9800機の終了アドレス（物理アドレス）に変換する。

30

40

【0141】

こうして変換されたアドレスとドライブデータテーブルから取得したその他の情報、例えばファイルシステムタイプなどを、ディスク情報ブロックのパーティション領域データに記憶する。その後、パーティション情報が残っている場合には（ステップS732）、テーブルの繋がりを示すポインタから次のドライブデータテーブルを検索し、同様にドライブデータテーブルの取得、パーティション情報の取得、パーティション情報の変換、パーティション情報の記憶の処理を繰り返す。

50

【0142】

総てのパーティションについて、以上の処理が完了した場合には、仮想PCシステムファイルのパーティション情報をハードディスクのパーティション情報の先頭に格納する処理を行なう（ステップS734）。この結果、図41（b）に示したように、仮想PCシステムファイルを仮想的にハードディスクの一パーティションとみなして先頭に置き、DOS/V機用のハードディスクのパーティションをその並びのままその後続くパーティションとして並べた配列に、パーティションは並べ替えられたことになる。

【0143】

次に、初期化処理（図33）に示した仮想ハードディスク作成処理について説明する。この処理は、仮想PCシステムファイルとして確保した領域を仮想ハードディスクとしてターゲットマシンに認識可能とする処理である。ここで仮想ハードディスクに関する情報としては、その最大容量が、エミュレーション装置の組み込み時に参照される情報ファイルに記録されている。最大容量の制限を加えるのは、オペレーティングシステムによっては、扱えるハードディスクの最大容量を制限しているものが存在するからである。また、仮想ハードディスクは、本実施例では、PC-9800機をエミュレーションしていることから、1セクタ当たりのバイト数を512、1トラック当たりのセクタ数を17、ヘッド数を8、最大接続台数を7としている。

【0144】

図42示した仮想ハードディスク作成処理ルーチンが起動されると、まず仮想ハードディスクの最大容量を、組み込み時に参照される情報ファイルを参照して取得する処理を行ない（ステップS742）、パーティション並び替え処理で記憶したディスク情報ブロックから仮想PC用のパーティション情報を読み出す処理を行なう（ステップS744）。このパーティション情報から、パーティションの容量を読み取り（ステップS746）、この容量が最大容量を超えているか判断する（ステップS748）。越えていない場合には、現在の仮想ハードディスクの情報に仮想PC用のパーティション情報をそのまま格納する（ステップS750）。他方、パーティションの容量が、仮想ハードディスクの最大容量を超えている場合には、次の仮想ハードディスクのディスク情報に仮想PC用のパーティション情報を格納する（ステップS752）。以上の処理を、仮想PC用のパーティション情報がなくなるまで繰り返す（ステップS754）。

【0145】

ここで言う仮想ハードディスク（仮想HDD）とは、エミュレーションにより構築されるターゲットマシンから扱うことができる論理デバイスという意味である。したがって、図41に示した例では、仮想PCシステムファイルが10Mバイト、基本DOS領域〈1〉が100Mバイト、基本DOS領域〈2〉が60Mバイト、拡張DOS領域〈1〉が30Mバイト、拡張DOS領域〈2〉が15Mバイト、仮想ハードディスクの最大容量が60Mバイトと仮定すると、上記ステップS748ないしS752の判断および処理により、各領域は、図43に示すように割り当てられる（なお、〈1〉〈2〉は図面においては丸付き数字を示す、以下本明細書において同じ）。まず、最初に仮想PCシステムファイルが、1台目の仮想ハードディスクとして割り当てられる。次に、実際に接続されている1台目のハードディスクの基本DOS領域が認識されるが、この領域は100Mバイトあり、最大容量を超えるているので、2台目の仮想ハードディスクとして割り当てられる。更に、実際に接続されている2台目のハードディスクの基本DOS領域が認識されるが、これは3台目の仮想ハードディスクとして割り当てられる。その次に、実際に接続されている1台目のハードディスクの拡張DOS領域が認識され、4台目の仮想ハードディスクとして割り当てられる。最後に、実際に接続されている2台目のハードディスクの拡張DOS領域が認識されるが、この領域を前の拡張DOS領域に加えても最大容量を超えないので、この拡張DOS領域は、先の領域に追加され、併せて4台目の仮想ハードディスクとして割り当てられる。

【0146】

初期化処理の最後の処理はアドレスの調整（図33ステップS760）とオフセットテ

10

20

30

40

50

ープルの作成（同ステップS780）である。図39に示したパーティションの並び替え処理により並び替えられた各パーティションの開始アドレス（物理アドレス）は、ハードディスクの先頭から順に並んでいるとは限らない。また、パーティションの終了アドレス（物理アドレス）と次のパーティションの開始アドレス（物理アドレス）についても、ハードディスクの先頭から順に並んでいるとは限らない。そこで、各パーティションの開始アドレスがハードディスクの先頭から順に並ぶようにアドレスの調整を行ない、その後、オフセットテーブルを作成してディスク情報ブロックに格納する処理が必要となる。これらの処理を併せてオフセットテーブル作成処理と呼ぶことにし、その詳細を図44に示した。

【0147】

10

オフセットテーブル作成処理が開始されると、まず仮想ハードディスクの情報から仮想PC用にパーティション情報を読み出す処理を行なう（ステップS782）。次に、パーティション情報のアドレスを、ハードディスクの先頭から並ぶように変更する処理を行ない（ステップS784）、これらの処理を総てのパーティションについて実行する（ステップS786）。ここで、アドレスをハードディスクの先頭から並ぶように変更する処理は、実際には次のアドレス調整処理により実現されている。

【0148】

アドレス調整処理－その1

DOS/V機の場合、ハードディスクの先頭パーティションの開始アドレス（物理アドレス）はヘッド番号1、シリンダ番号0、セクタ番号1から始まっている。このDOS/V機での開始アドレス（物理アドレス）をターゲットマシンであるPC-9800機の物理アドレスに変換し、仮想ハードディスクの先頭パーティションの開始アドレス（物理アドレス）とする。得られたこの開始アドレス（物理アドレス）を論理アドレスに変換し、このパーティションの総セクタ数を加算する。即ち、論理アドレス上でパーティションの終了アドレスを求めるのである。この終了アドレス（論理アドレス）を仮想ハードディスクのヘッド数、セクタ数を用いて物理アドレスに変換し、そのパーティションの終了アドレス（物理アドレス）とする。次のパーティションの開始アドレスは、ヘッド番号1、セクタ番号1、シリンダ番号は一つ前のパーティションの終了アドレスのシリンダ番号Nに値1を加えたN+1とする。

20

【0149】

30

アドレス調整処理－その2

ターゲットマシンであるPC-9800機の場合、ハードディスクの最初のパーティションの開始アドレス（物理アドレス）は、ヘッド番号0、シリンダ番号1、セクタ番号0から始まっているので、この開始アドレス（物理アドレス）を仮想ハードディスクの先頭パーティションの開始アドレス（物理アドレス）とする。この開始アドレス（物理アドレス）を論理アドレスに変換し、これに総セクタ数を加算する。加算されたアドレス（論理アドレス）を仮想ハードディスクのヘッド数、セクタ数を用いて物理アドレスに変換し、これをパーティションの終了アドレス（物理アドレス）とする。次のパーティションの開始アドレスは、ヘッド番号0、セクタ番号0、シリンダ番号は一つ前のパーティションの終了アドレスのシリンダ番号Mに値1を加えたM+1とする。

40

【0150】

以上のアドレス変換を総てのパーティションについて行ない、更に総ての仮想ハードディスクについて行なう（ステップS788）。複数の仮想ハードディスクが接続されている場合には、各ハードディスクについては、ヘッド番号、シリンダ番号、セクタ番号は、初期値から再開される。

【0151】

次に、オフセットテーブルの作成処理を行なう。まず、仮想ハードディスクの情報から、アドレス変更前と変更後の仮想PC用のパーティション情報を読み出し（ステップS790）、両アドレスの差を算出し、オフセットアドレスとして、これをディスク情報ブロックのオフセットテーブルデータに記憶する処理を行なう（ステップS792）。このオ

50

フセットアドレスを求める処理を総てのパーティションについて、更に総ての仮想ハードディスクについて繰り返す（ステップS794, 796）。

【0152】

このオフセットテーブルの作成の処理もアドレス調整と同様、詳細には、次のように行なわれる。

オフセットテーブル作成－その1

先にアドレス調整－その1でアドレス調整が行なわれた仮想ハードディスクのパーティションの開始アドレス（論理アドレス）とアドレス調整される前の仮想ハードディスクのパーティションの開始アドレス（論理アドレス）との差を求め、実際のハードディスクにおけるパーティションの開始アドレスからのオフセットOFS1として、オフセットテーブルデータに記憶する。なお、ハードディスクのドライブ番号の差については、仮想PCシステムファイルの情報の取得（図38）およびパーティションを並び替えた時点（図39）でオフセットテーブルデータに記憶されている。

10

【0153】

オフセットテーブル作成－その2

先にアドレス調整－その2で調整が行なわれた仮想ハードディスクのパーティションの開始アドレス（論理アドレス）とアドレス調整される前の仮想ハードディスクのパーティションの開始アドレス（論理アドレス）との差を求め、実際のハードディスクにおけるパーティションの開始アドレスからのオフセットOFS2として、先にオフセットテーブルデータに記憶したオフセットOFS1に更に加算する（OFS1+OFS2）。これが最終的なオフセットOFSとなる。なお、ドライブ番号についてのオフセットが既に記憶されている点は、上述の通りである。

20

【0154】

以上のアドレス調整の様子を、仮想PCシステムファイルを例にとって図45に簡略に示した。実際のハードディスク84を番号#0として、その基本DOS領域に仮想PCシステムファイルを作成したとする。この仮想PCシステムファイルは、仮想PC、即ちターゲットマシンでは、仮想ハードディスクとして扱われ、図41に示したように、パーティションの先頭に配置される。そこで、番号#1の仮想ハードディスクのオフセットテーブルには、仮想PCシステムファイルの先頭から実際のハードディスク上のDOSブート領域の先頭までのセクタ数が、オフセットアドレスとして記憶される。また、この仮想PCシステムファイルによる仮想ハードディスクがディスク番号#0に当たるとして、対応する番号80Hが、オフセットテーブルに、同様に記憶されている。以下に説明するハードディスクへのアクセスは、このオフセットテーブルのデータを用いて行なわれることになる。

30

【0155】

以上、本実施例におけるハードディスクエミュレータHDMが組み込まれる際の処理について詳しく説明した。かかる処理により、DOS/V機上で、ターゲットマシンとしてのPC-9800機のハードディスクアクセスをエミュレーションする準備が整ったことになる。ターゲットマシン用のシステムがブートされる仮想的なハードディスクは、DOS/V機上では、連続した領域を有するファイルとして扱われており、これが仮想PCシステムファイルとしてパーティションの情報に組み込まれ、先頭のパーティションとして、ターゲットマシンから認識およびアクセス可能となっている。また、DOS/V機において接続されていたハードディスクおよびパーティションは、エミュレーションされたターゲットマシンであるPC-9800機から認識およびアクセス可能な仮想ハードディスクとして、DOS/V機での並びと同じ並びとなっている。

40

【0156】

最後に、エミュレートされたターゲットマシンであるPC-9800機からこれらのハードディスクにアクセスする場合に処理について説明する。この処理は、図46のフローチャートに示したように、第1実施例における処理と類似の処理である（図23参照）。ターゲットマシンにおいてハードディスクへのアクセスを行なおうとすると、そのI/O

50

アクセスにより処理はカーネルKRに移行し、ディスパッチャDPを介してハードディスクエミュレータHDMが呼び出される。ハードディスクエミュレータHDMは、図46に示す処理を開始し、まずターゲットマシンにおけるハードディスクアクセスが、物理アドレスによるものか否かを判定する(ステップS800)。物理アドレスによりアクセスの場合にはこれを論理アドレスに変換し(ステップS802)、続いてオフセットテーブルデータからオフセットアドレスとドライブ番号とを読み出す処理を行なう(ステップS804)。このオフセットアドレスをアクセスしようとした論理アドレスに加え(ステップS806)、更にドライブ番号も勘案して、DOS/V機に実際に接続されているハードディスク84の物理アドレスに変換する処理を行なう(ステップS808)。この物理アドレスを用いて、DOS/V機のBIOSを呼び出し、ディスクアクセスを実行させる(ステップS810)。呼び出されるDOS/V機のBIOSは、仮想PC上で実行しようとしたBIOSと等価な機能を有するBIOSである。このBIOSは、CPU21のタスクをDOS/V機のDOSのタスクに切り替えることにより実行される。

10

【0157】

以下、データの読み出しの場合について説明すると、読み出したデータがシステム領域もしくはパーティション情報から読み出されたものか否かを判断し、システム領域またはパーティション情報から読み出しものであると判断された場合には、DOS/V機のデータを読み出しても無意味であることから、仮想PC用のシステム領域またはパーティション情報を読み出して(ステップS814)、これを仮想PCのメモリに書き込み処理を行なう(ステップS820)。また、変換して得られた物理アドレスによるデータの読み込みが、DOSのブート領域に当たっている場合にも(ステップS816)、そのまま読み込むのではなく、仮想PCのDOSブート領域の対応するデータを読み出し(ステップS818)、これを仮想PCのメモリに書き込む(ステップS820)。これら以外の場合には、単なるデータであることから、DOS/V機のBIOSの実行により得られたデータ(ステップS810)をそのまま仮想PCのメモリに書き込むことになる(ステップS820)。

20

【0158】

第一実施例で詳しく説明したように、読み出しの場合、実際にハードディスク84からデータを読み出すのはDOS/V機のBIOSである。このBIOSの実行は、DOS/V機のDOSのタスク上で実行され、ハードディスクの該当セクタからデータが読み出され、バッファ上にロードされる。ステップS820では、このデータまたは仮想PC用のシステム領域またはパーティション情報あるいは仮想PC用のブート領域のデータを、仮想PC上のBIOSが管理しているバッファ領域に転送する。この結果、処理が、図20に示したように、ハードディスクエミュレータHDMから、仮想PC上のBIOSに戻ったとき、仮想PCのタスクで実行されるBIOSは、自分が管理しているバッファ領域にハードディスクからデータを読み出したものとして、自分を呼び出したアプリケーションプログラムに処理を戻すことになる。

30

【0159】

以上説明した本実施例のエミュレーション装置によれば、実行マシンであるDOS/V機においてターゲットマシンであるPC-9800機のハードディスクのアクセスを、そのパーティションの違いなどを含めて完全にエミュレートすることができる。しかも、DOS/V機のBIOSをそのまま利用し、DOS/V機のファイルとしてターゲットマシン用のハードディスクの領域を確保する方式を採用したことから、ターゲットマシンのためのハードディスクを別に確保する必要がない。しかも、一旦実行マシンのBIOSを読み込んでから、ターゲットマシンの環境に移行し、実行マシンのBIOSおよびDOSもタスクを切り替えることで実行可能であるため、ハードディスクアクセスのエミュレーションが容易になった。DOS/V機のシステムファイルの名称などを書き換える必要がなく、システム運用の使い勝手、信頼性も向上する。

40

【0160】

次に、本発明の第3、第4実施例について説明する。この実施例では、CD-ROMな

50

どの周辺装置をエミュレートする構成を示す。周辺装置には、様々なものが存在するから、これをすべてエミュレーションできるように、予めエミュレータを用意することは極めて困難である。特に、ハードウェアに付属する専用のデバイスドライバを用いてアクセスするものでは、その内容をいちいち解析してエミュレータモジュールを用意することは困難である。そこで、基本構成として、図47に示すように、CPU21に、実行マシンに接続された周辺機器用のデバイスドライバを実行可能に組み込む第1の組込手段KK1、実行マシン用のデバイスドライバを用いた周辺機器のアクセスに必要な第1のバッファBF1、第1の組込手段KK1による組込の後で、ターゲットマシンに用意されたオペレーティングシステムを組み込むオペレーティングシステム組込手段KOS、組み込まれたオペレーティングシステムに周辺機器用のターゲットマシンのデバイスドライバの少なくとも呼出側を実行可能に組み込む第2の組込手段KK2、ターゲットマシンのデバイスドライバを用いた周辺機器のアクセスに必要な第2のバッファBF2、ターゲットマシンのデバイスドライバの呼出を取得し、実行マシンのデバイスドライバを起動して、周辺装置へのアクセスを実行させると共に、第1のバッファBF1と第2のバッファBF2との間でデータの転送を行なわせる転送手段TRSを、備える。

10

【0161】

次に、これらの手段の具体的な構成について説明する。第3実施例では、図48に示すように、周辺装置850をアクセスする構成において、実行マシンであるDOS/V機に用意され、その周辺装置を本来制御すべく用意されたデバイスドライバ852を一旦読み込んでから処理をターゲットマシンである仮想PC（ここではPC-9800機）に移行する第2実施例と同様の手法を適用する。エミュレーションにおいて実行マシンのデバイスドライバ852をそのまま利用するのである。DOS/V機のデバイスドライバ852が組み込まれた後で、エミュレーションのための初期化の処理に移行し（図32参照）、その後、仮想PCとしてブートして、DOSを立ち上げる。この過程で、周辺装置850用として、オペレーティングシステムを介して呼び出される仮想のデバイスドライバ860を用意し、これを組み込む。

20

【0162】

この仮想のデバイスドライバ860は、仮想PCが、周辺装置850をアクセスするとき、呼び出すものであり、アプリケーションプログラムAPP側から呼び出された時、一般保護例外のトラップを起こすよう構成すれば足りる。いわば、呼出側の入り口の機能さえ用意しておけば足りるのである。仮想のデバイスドライバ860が呼び出されると、一般保護例外のトラップが生じ、カーネルKRがこれをフックしてエミュレーション装置に制御を移行する。カーネルKRは、ブリッジシステムBSを介して、メモリ上のリアル空間に呼び出される処理が実行マシンのデバイスドライバ852、BIOSとなるようタスクを切り替え、実行マシン用に用意されたデバイスドライバ852をそのまま利用して処理を行なう。

30

【0163】

周辺装置850が単純なアクチュエータなどである場合には、デバイスドライバ852を介して周辺装置850に所定の信号を出力した後、処理を、上述した流れを逆に辿って、アプリケーションプログラムAPPに返せば良い。この結果、DOS/V機側では、周辺装置850用のデバイスドライバをそのまま流用することができ、製品の開発、製造が極めて容易となった。したがって、エミュレーション装置の全体構成、見通しも極めて良好である。

40

【0164】

次に、この第3実施例の構成・手法を、特にCD-ROMに適用した構成を取り上げ、説明する。図49は、周辺装置としてCD-ROM870を実行マシンであるDOS/V機（図2参照）に接続した場合に、仮想PC（実施例ではPC-9800機）からこれをアクセスする構成を示すブロック図である。このCD-ROM870用に、DOS/V機では、専用のデバイスドライバ872が用意されている。このデバイスドライバ872は、DOS/V機においてDOSを立ち上げる過程で、DOS/V機のDOSを専用のメモ

50

リ空間に展開した後、システムファイル「`config.sys`」の記述に従い、組み込まれる。このデバイスドライバ872には、CD-ROM870から読み出したデータを記憶する第1バッファ874が必要となるが、この第1バッファ874は、DOS/V機のBIOSの展開後に行なわれるエミュレーション装置の組込時（詳しくはブリッジシステムBSの組込時）に、DOS/V機のDOSのタスク空間の一部に固定的に確保される。ブリッジシステムBSが、DOS/V機のデバイスドライバ872を呼び出すとき、第1バッファ874の位置を、デバイスドライバ872に教える。バッファ874の容量は、通常固定的に確保されるが、エミュレーション装置のオプションスイッチにより指定する構成とすることもできる。データの受け渡しのためのバッファを、デバイスドライバを呼び出す側が確保することは、周知の手法である。

10

【0165】

CD-ROM用のデバイスドライバ872など必要なデバイスドライバを組み込んだ後、図32に示したように、エミュレータの構築が行なわれる。その構築の手法については、詳しく説明したので繰り返さないが、本実施例では、カーネルKR、ディスパッチャDPを組み込んだ後、他のエミュレーションモジュール群と共にブリッジシステムBSというエミュレータが組み込まれる。このブリッジシステムは、第3実施例で説明したように、仮想PCのデバイスドライバと実行マシンであるDOS/V機のデバイスドライバとを繋ぐものである。その機能については、後述する。

【0166】

エミュレータの構築の後、仮想PC側のBIOSを展開し、第2実施例で詳しく説明したように、DOS/V機のハードディスク84内に連続した領域として確保された仮想システムファイルなどをアクセスし、初期化の処理を実行し、その後、仮想システムファイルのブート領域からデータを読み出して、ターゲットマシンであるPC-9800機としてのブートを実行する。この結果、図49に示したように、仮想PCのDOSが、仮想PC用のメモリ空間に展開され、更に種々のデバイスドライバが組み込まれる。こうして組み込まれるデバイスドライバの一つとして、CD-ROMのアクセスを担当する仮想デバイスドライバ880も組み込まれる。また、MS-DOSでは、CD-ROMをアクセスするためのデバイスドライバとDOSとのインタフェースを定義したMSCDEX882というモジュールも組み込まれる。アプリケーションプログラムAPPは、このMSCDEX882を介してCD-ROMをアクセスしても良いし、直接仮想デバイスドライバ880をアクセスしても差し支えない。仮想デバイスドライバ880は、MSCDEX882が定める入出力を用意しているので、この入出力をアプリケーションプログラムAPPからアクセスすることは容易である。なお、仮想デバイスドライバ880とのデータのやり取りに必要な第2バッファ884は、仮想デバイスドライバ880を呼び出す側（ここではアプリケーションプログラムAPP）が、仮想PCのメモリ空間に動的に確保する。アプリケーションプログラムAPPは、CD-ROMをアクセスしようとする、メモリ空間に第2バッファ884を確保した後、その位置を仮想デバイスドライバ880に連絡する。

20

30

【0167】

以上の組み込み作業やバッファの確保などを行なった後、仮想PCは、アプリケーションプログラムAPPを実行可能な状態となる。この状態で、アプリケーションプログラムAPPからCD-ROM870をアクセスする場合の各部の動作について説明する。アプリケーションプログラムAPPがMSCDEX882を介して、あるいは直接にCD-ROM870からデータを読み出す処理を呼び出すと、仮想デバイスドライバ880は、この呼出を受けて、「HALT」命令などの特権命令を実行する。仮想PCは、プロテクトモードで動作しており、かかる特権命令の実行は、カーネルKRによりトラップされる。カーネルKRは、システムの状態をチェックしてトラップの要因を特定し、この場合には、ディスパッチャDPを介してブリッジシステムBSを呼び出す。ブリッジシステムBSは、変換部890と転送部892とを備える。変換部890は、仮想PCとDOS/V機とでCD-ROMのアドレスの指定の方法が異なる場合にアドレス変換を行ない、更に仮

40

50

想PCからの命令についてもDOS/V機のCD-ROM870に対する命令に変換する処理を行なう。転送部892は、後述する第1バッファ874から第2バッファ884へのデータ転送を制御するものである。

【0168】

ブリッジシステムBSは、CPU21の実行するタスクをDOS/V機のタスクに切り替えると共に、変換されたアドレスと命令を、DOS/V用に用意されたデバイスドライバ872に引き渡す。デバイスドライバ872は、このアドレスと命令により、CD-ROM870にアクセスする。このデバイスドライバ872は、DOS/V機用のCD-ROM特有のハードウェア（モータ、ヘッドの仕様等）を考慮して作られているから、CD-ROM870を正しくアクセスし、必要なデータをDOS/V機のタスク内の第1バッファ874に格納する。これはDOS/V用デバイスドライバが自分が動作しているタスク内にバッファを持つものとして作られているからである。

10

【0169】

以上の処理の後、CD-ROMデバイスドライバ872は、処理をカーネルKRに戻す。カーネルKRはタスクをエミュレータ管理下（プロテクトモード）に戻し、ブリッジシステムBSを呼び出す。ブリッジシステムBSでは、その転送部892が処理の戻りを判別し、第1バッファ874のデータを第2バッファ884に転送する処理を行なう。第2バッファ884は、仮想PCのタスク内のメモリ領域である。バッファ874、884間のデータの転送は、第1バッファ874のデータを一旦ブリッジシステムBSが吸い上げ、これを第2バッファ884に書き出すことにより行なわれる。この後、処理は、ブリッジシステムBSからディスパッチャDP、カーネルKRを介して、仮想デバイスドライバ880へと戻って行くが、仮想PC側が予定しているデータ量（第2バッファ884に用意されるべきデータ量）とCD-ROMデバイスドライバ872が読み出して第1バッファ874に用意したデータ量とが異なる場合には、次のように処理が行なわれる。

20

【0170】

仮想PCが予定しているデータ量より、第1バッファ874に用意されるデータ量が少ない場合には、ブリッジシステムBSの転送部892は、仮想PCが予定しているデータ量となるまで、CD-ROMデバイスドライバ872に処理を戻し、第1バッファ874から第2バッファ884へのデータ転送を繰り返す。第2バッファ884に必要なデータが用意されてから、ブリッジシステムBSは、処理をディスパッチャDPに戻すことになる。なお、この機能を積極的に利用して、DOS/V機側で一回に読み込むデータ量を小さくして、第1バッファ874の容量を小さくすることも可能である。

30

【0171】

他方、仮想PCが予定しているデータ量より第1バッファ874に用意されるデータ量の方が大きい場合には、ブリッジシステムBSの転送部892は、第1バッファ874の一部のみ第2バッファ884に転送する。仮想PC側からのCD-ROM870に対するアクセスが連続セクタリード命令等の場合は、ブリッジシステムBSは、2回目以降のセクタリードに対しては、第1バッファ874にデータが残っている限りは、CD-ROMデバイスドライバ872を呼び出すことなく、直ちに転送部892を起動して、第1バッファ874の残余のデータを、第2バッファ884に転送する。この場合には、CD-ROM870のアクセス回数を減らすことができるので、データ転送の速度を上げることができる。

40

【0172】

処理がディスパッチャDPからカーネルKRを介して、仮想PC側で実行されているアプリケーションプログラムAPPに戻ると、アプリケーションプログラムAPPは、第2バッファ884をアクセスして所望のデータを読み取ることができる。なお、本実施例の場合、CD-ROM870をアクセスするエミュレータの全体は、仮想PC上のDOSとのインタフェースを司るMSCDEX882と、ハードウェアとの接点であるDOS/V用デバイスドライバとの間にブリッジシステムを設けた構成となっている。MSCDEX882とCD-ROMデバイスドライバ872とのインタフェースは共通化が図られてい

50

る。この位置に仮想デバイスドライバ８８０を組み込んでいるのでＤＯＳあるいはハードウェアが変更されても仮想デバイスドライバ８８０を継続使用することが可能になっている。

【０１７３】

以上説明した実施例では、周辺装置がＣＤ－ＲＯＭ８７０のようにデータの転送を必要とするものである場合に、各デバイスドライバが実行されるタスク内に用意されたバッファにデータを読み書きし、これをブリッジシステムＢＳにより転送しており、データ転送を伴う周辺装置のエミュレーションを極めて容易に実現することができる。

【０１７４】

上記実施例では、ＤＯＳ／Ｖ機上でＰＣ－９８００機を実現する構成を例に挙げて説明したが、ＰＣ－９８００機上でＤＯＳ／Ｖ機を実現することも容易である。またＤＯＳ／Ｖ機はＩＢＭ－ＡＴ互換機と読み替えてもよい。更に、他のアーキテクチャに適用することも可能である。また、図１に示した実行モジュールの分け方は一例に過ぎず、モジュールを更に細分したり、併合したりすることも可能である。

【図面の簡単な説明】

【０１７５】

【図１】本発明の一実施例であるエミュレーション装置の概略構成を示すブロック図である。

【図２】実行マシンであるＤＯＳ／Ｖ機の概略構成図である。

【図３】タスクアドレスと物理メモリとの関係を示す説明図である。

【図４】実行モードとその切り換えの関係を示す説明図である。

【図５】ＤＯＳ／Ｖ機が立ち上がった直後のメモリマップおよびＩ／Ｏマップを示す説明図である。

【図６】エミュレータ装置の初期化処理ルーチンを示すフローチャートである。

【図７】初期化の処理の様子を示す説明図である。

【図８】ＤＯＳ／Ｖ機の割込関係の回路を示すブロック図である。

【図９】実行モジュールを組み込む際の処理ルーチンを示すフローチャートである。

【図１０】ディスパッチャ呼出ルーチンを示すフローチャートである。

【図１１】エントリポイント取得ルーチンを示すフローチャートである。

【図１２】実行モジュールの組み込み時における呼出アドレスの登録の様子を示す説明図である。

【図１３】実行モジュールがその登録時に他の実行モジュールを呼び出す様子を示す説明図である。

【図１４】実行モジュール間の直接呼出（モジュール間通信）の様子を示す説明図である。

【図１５】ＤＯＳ／Ｖ機におけるキーボード周りの回路を示すブロック図である。

【図１６】ＰＣ－９８００機におけるキーボード周りの回路を示すブロック図である。

【図１７】実施例のエミュレーション装置において、キーボードからの入力をエミュレートする様子を示す説明図である。

【図１８】同じくキーボードからの入力のエミュレーションの後半の様子を示す説明図である。

【図１９】ハードディスクの物理的構造図である。

【図２０】本実施例のエミュレーション装置において、ハードディスクへの読み込みをエミュレートする様子を示す説明図である。

【図２１】ハードディスクへの読み込みをエミュレートするハードディスクエミュレータの構成図である。

【図２２】ハードディスクの初期化を示すフローチャートである。

【図２３】ハードディスクへの読み込みを示すフローチャートである。

【図２４】ＤＯＳ／Ｖ機におけるパーティション領域のマップを示す説明図である。

【図２５】ＤＯＳ／Ｖ機におけるハードディスクのマップを示す説明図である。

10

20

30

40

50

【図 2 6】 P C - 9 8 0 0 機におけるパーティション領域のマップを示す説明図である。
 【図 2 7】 P C - 9 8 0 0 機におけるハードディスクのマップを示す説明図である。
 【図 2 8】 P C - 9 8 0 0 機におけるシステム領域のマップを示す説明図である。
 【図 2 9】 P C - 9 8 0 0 機における D O S ブート領域のマップを示す説明図である。
 【図 3 0】 エミュレーション用のシステムをインストールするための処理ルーチンを示すフローチャートである。

【図 3 1】 インストール処理における設定画面を例示する説明図である。
 【図 3 2】 エミュレーション装置における電源投入時のシーケンスを示すフローチャートである。

【図 3 3】 初期化の処理の概要を示すフローチャートである。 10

【図 3 4】 ハードディスクエミュレータ H D M の概略構成を示す説明図である。

【図 3 5】 仮想 P C システムファイルの構成を例示する説明図である。

【図 3 6】 仮想 P C システムファイルの D O S ブート領域の構成を例示する説明図である。
 。

【図 3 7】 ドライブデータテーブルに一例を示す説明図である。

【図 3 8】 仮想 P C システムファイル情報作成処理を示すフローチャートである。

【図 3 9】 パーティションの並び替え処理を示すフローチャートである。

【図 4 0】 パーティションの並びをポインタ P N T で参照する様子を示す説明図である。

【図 4 1】 ハードディスクの各領域を並び替える様子を示す説明図である。

【図 4 2】 仮想ハードディスクの作成処理を示すフローチャートである。 20

【図 4 3】 仮想ハードディスクの作成の様子を示す説明図である。

【図 4 4】 オフセットテーブル作成処理を示すフローチャートである。

【図 4 5】 オフセットデータを用いたアクセスの様子を簡略に示す説明図である。

【図 4 6】 ハードディスクのアクセスの様子を読み込み処理により示すフローチャートである。

【図 4 7】 第 3, 第 4 実施例に相当する基本構成を示す説明図である。

【図 4 8】 本発明の第 3 実施例として、周辺機器のアクセスをエミュレートする構成を示すブロック図である。

【図 4 9】 本発明の第 4 実施例として、 C D - R O M 8 7 0 のアクセスをエミュレートする構成を示すブロック図である。 30

【符号の説明】

【 0 1 7 6 】

2 0 … 演算処理部

2 1 … C P U

2 2 … ローカルバス

2 3 … キャッシュメモリ

2 4 … キャッシュコントローラ

2 5 … メインメモリ

3 0 … P C I ブリッジ

3 2 … P C I バス

4 0 … コントローラ部

4 2 … I S A バス

4 4 … V G A

4 6 … S C S I コントローラ

4 8 … P C I - I S A ブリッジ

5 0 … D M A コントローラ

5 2 … リアルタイムクロック

5 4 … 複合 I / O ポート

5 6 … サウンド I / O

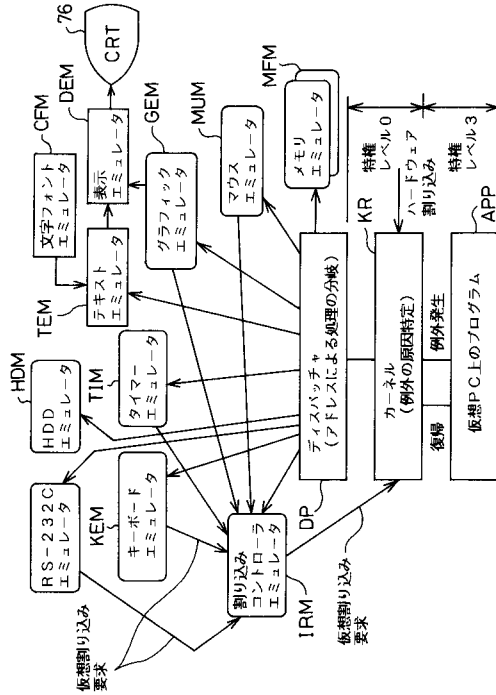
6 0 … I / O 部

40

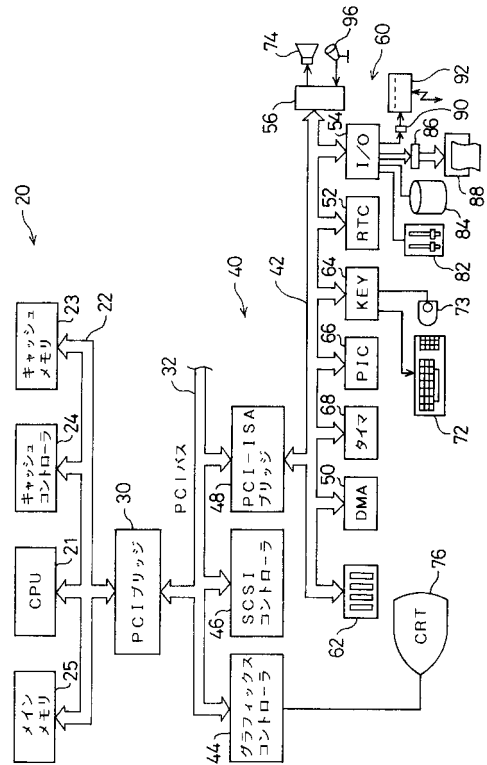
50

6 2 … 拡張スロット	
6 4 … K E Y	
6 6 … P I C	
6 8 … タイマ	
7 2 … キーボード	
7 3 … マウス	
7 4 … スピーカ	
7 6 … C R T	
8 2 … フロッピディスク装置	
8 4 … ハードディスク	10
8 6 … パラレルポート	
8 8 … プリンタ	
9 0 … シリアルポート	
9 2 … モデム	
9 6 … マイクロフォン	
8 5 0 … 周辺装置	
8 5 2 … デバイスドライバ	
8 6 0 … デバイスドライバ	
8 7 0 … C D - R O M R O M	
8 7 2 … C D - R O M デバイスドライバ	20
8 7 4 … 第 1 バッファ	
8 8 0 … 仮想デバイスドライバ	
8 8 2 … M S C D E X	
8 8 4 … 第 2 バッファ	
8 9 0 … 変換部	
8 9 2 … 転送部	
A P P … アプリケーションプログラム	
A n … 呼出アドレス	
B F 1 … 第 1 のバッファ	
B F 2 … 第 2 のバッファ	30
B S … ブリッジシステム	
C F M … 文字フォントエミュレータ	
D E M … 表示エミュレータ	
D M n … 実行モジュール	
D P … ディスパッチャ	
D T … ディスパッチャテーブル	
G E M … グラフィックエミュレータ	
I R M … 割込コントローラエミュレータ	
H D M … ハードディスクエミュレータ	
K E M … キーボードエミュレータ	40
K K 1 … 第 1 の組込手段	
K K 2 … 第 2 の組込手段	
K O S … オペレーティングシステム組込手段	
K R … カーネル	
M E M … メモリエミュレータ	
M U M … マウスエミュレータ	
T E M … テキストエミュレータ	
T I M … タイマエミュレータ	
T R S … 転送手段	
V S F … 仮想システムファイル領域確保部	50

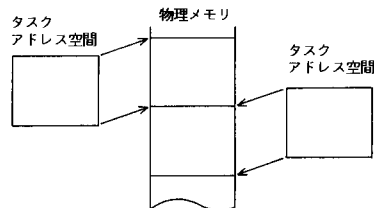
【 図 1 】



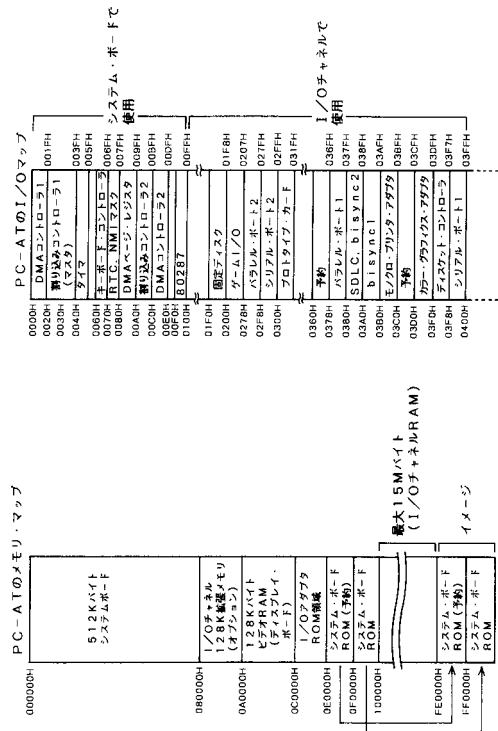
【図 2】



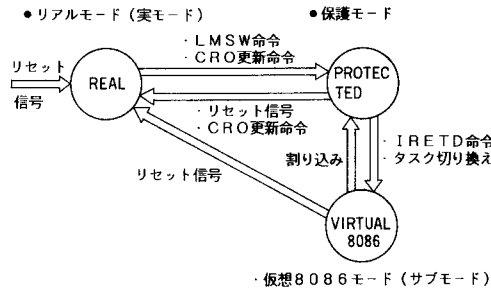
【 図 3 】



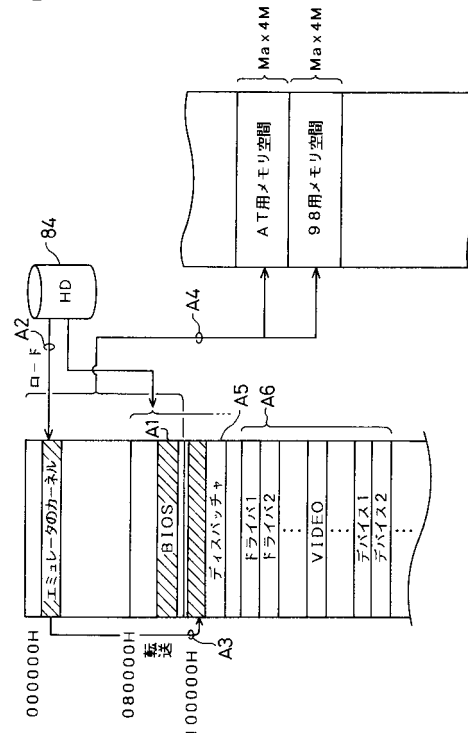
【図 5】



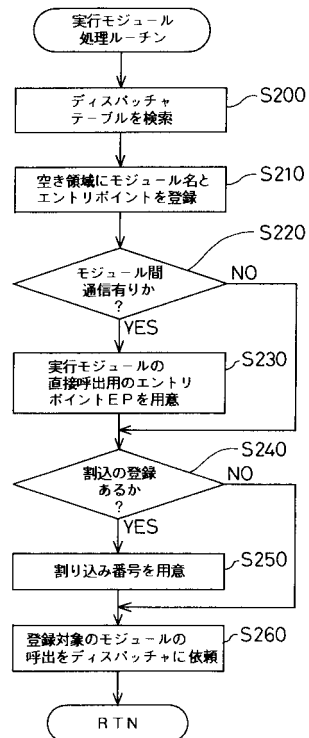
【図 4】



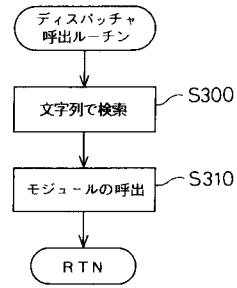
【图 7】



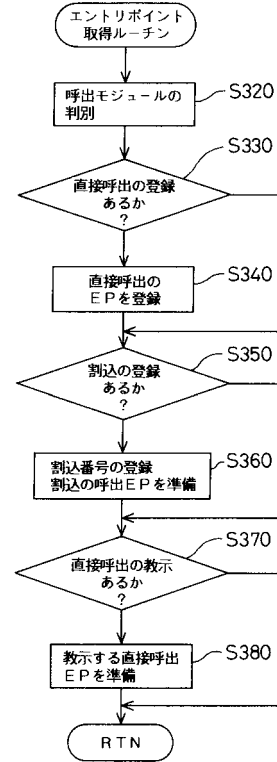
【图 9】



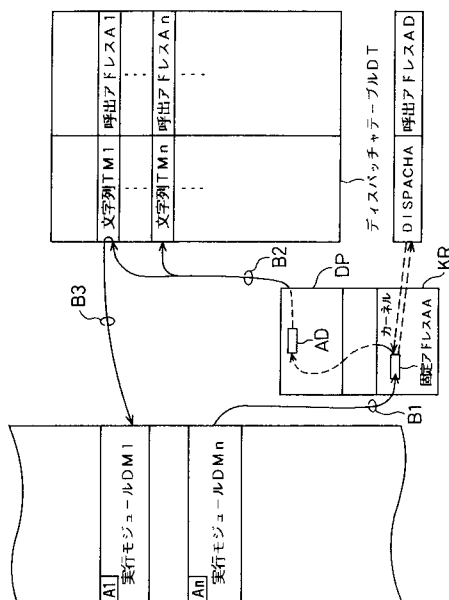
【図 10】



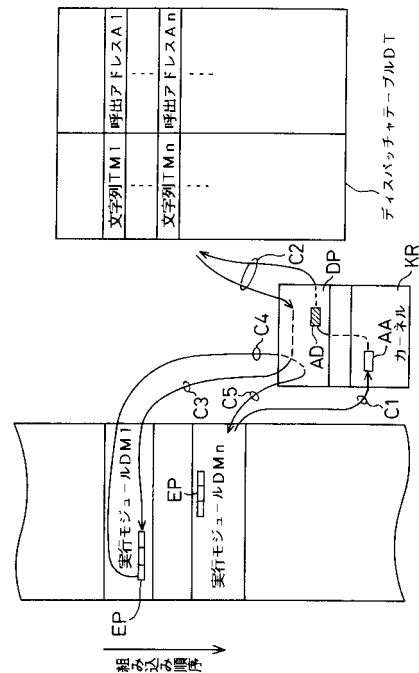
【図 11】



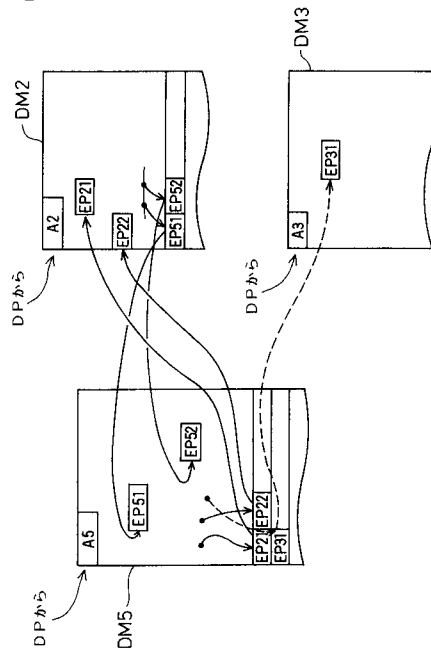
【図 12】



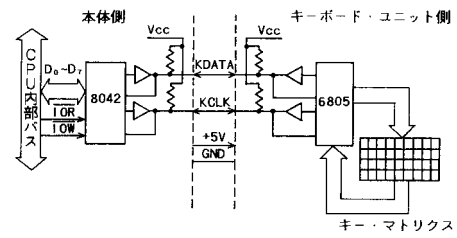
【図 13】



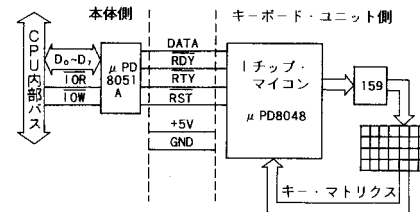
【図 14】



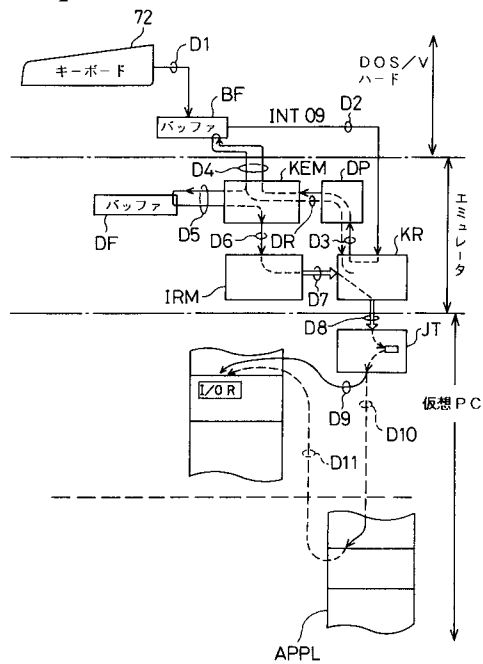
【図 15】



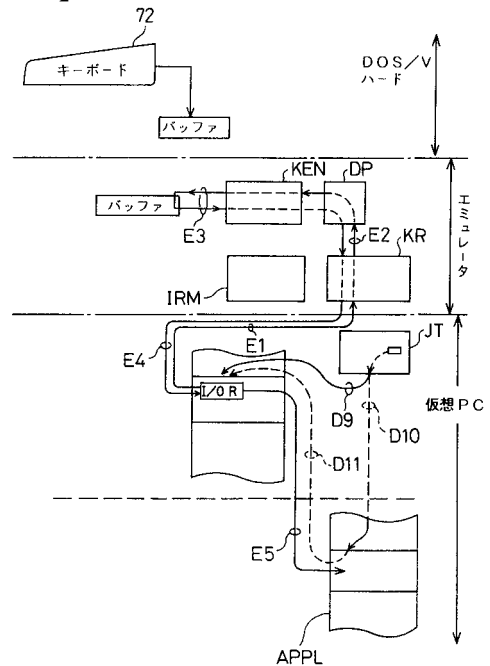
【図 16】



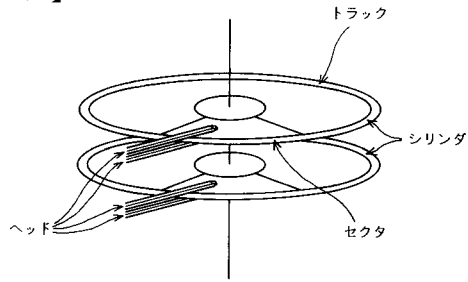
【図 17】



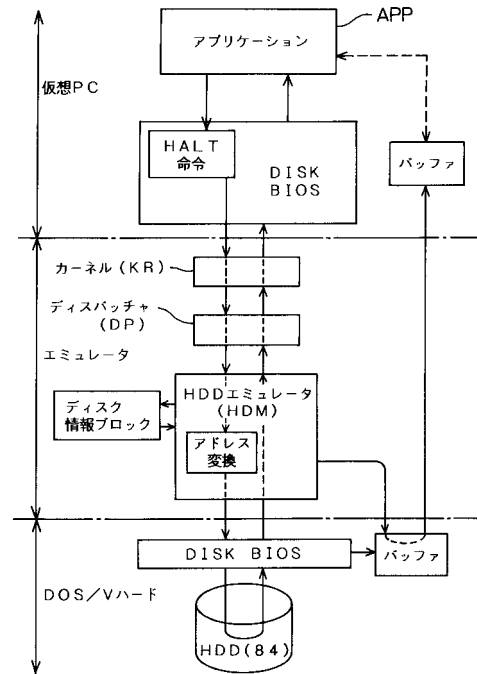
【図 18】



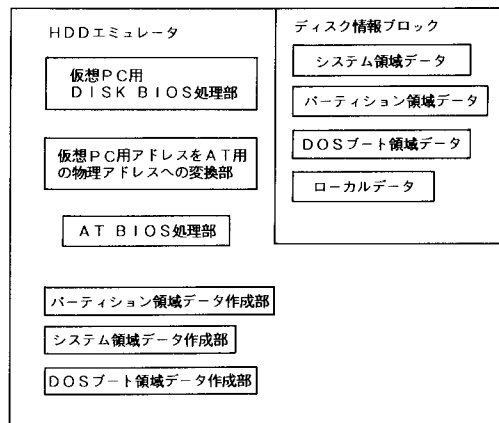
【図 19】



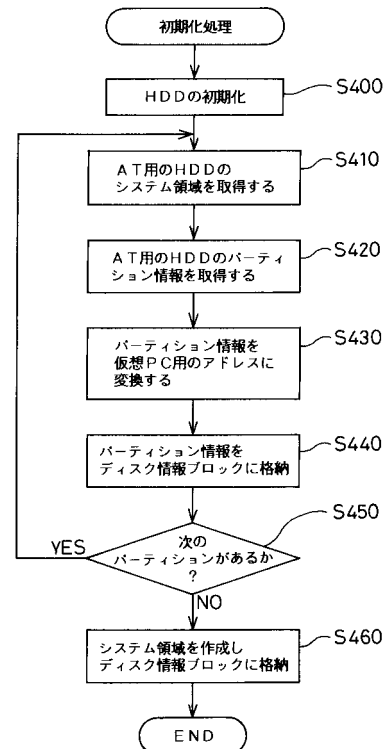
【図 20】



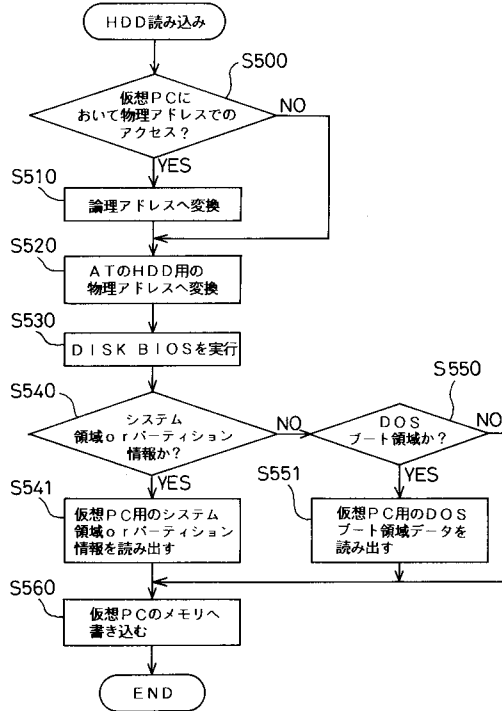
【図 21】



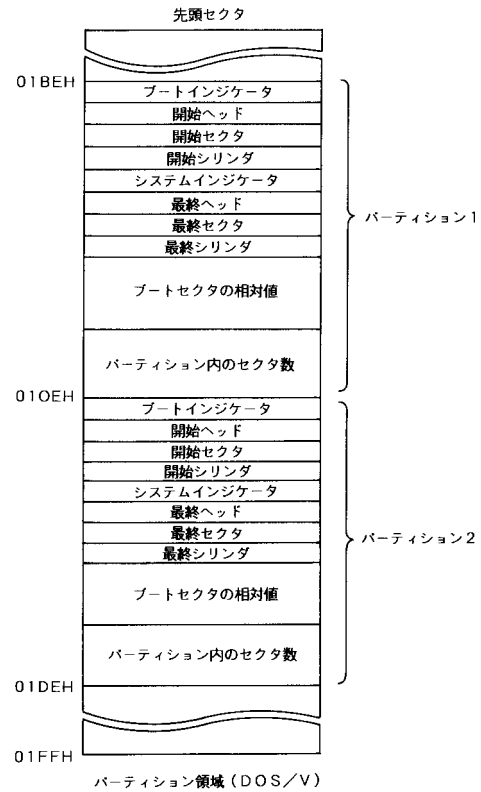
【図 22】



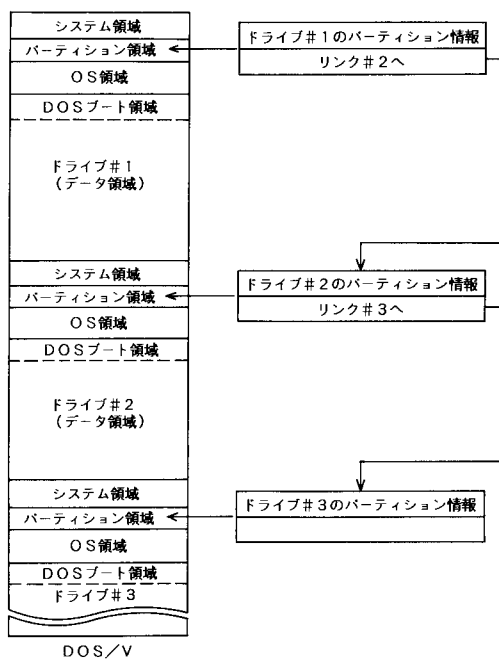
【図 2 3】



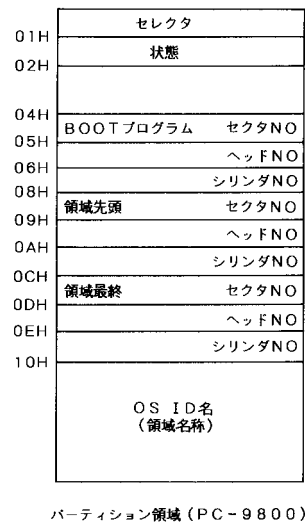
【図 2 4】



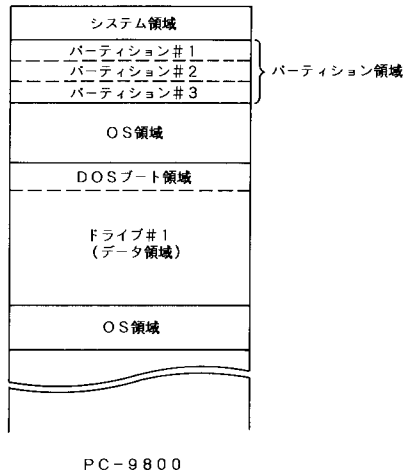
【図 2 5】



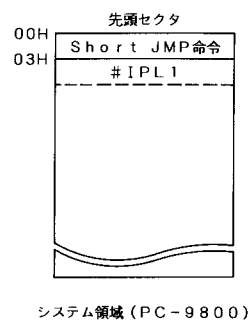
【図 2 6】



【図 27】



【図 28】

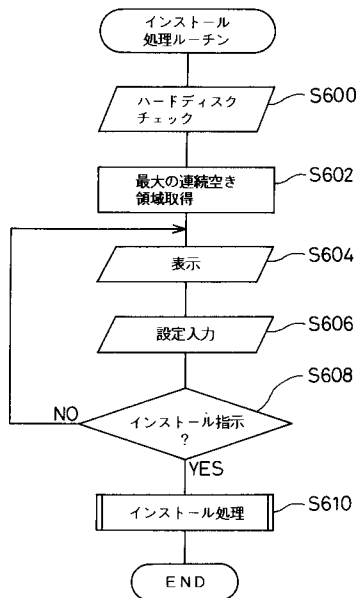


【図 29】

00H	Short JMPプログラム先頭90H
03H	OEM名
0BH	論理セクタ長
0PH	アロケーションユニットサイズ
0EH	予備領域サイズ
10H	FAT数
11H	ディレクトリ数
13H	領域サイズ
15H	FAT1D
16H	FAT領域サイズ
18H	セクタ数/トラック
1AH	ヘッド数
1CH	MS-DOS領域先頭相対セクタ(下位)
1EH	MS-DOS領域先頭相対セクタ(上位)
20H	領域サイズ
24H	0080H
26H	拡張BOOTレコード識別子
27H	ボリュームシリアルNO
2BH	ボリュームラベル
36H	ファイルシステム名
3FH	MS-DOS領域先頭相対セクタ(下位)
40H	MS-DOS領域先頭相対セクタ(上位)
42H	OS領域サイズ
44H	物理セクタ長
46H	デバイスコード
47H	コマンドコード
48H	

DOSブート領域 (PC-9800)

【図 30】

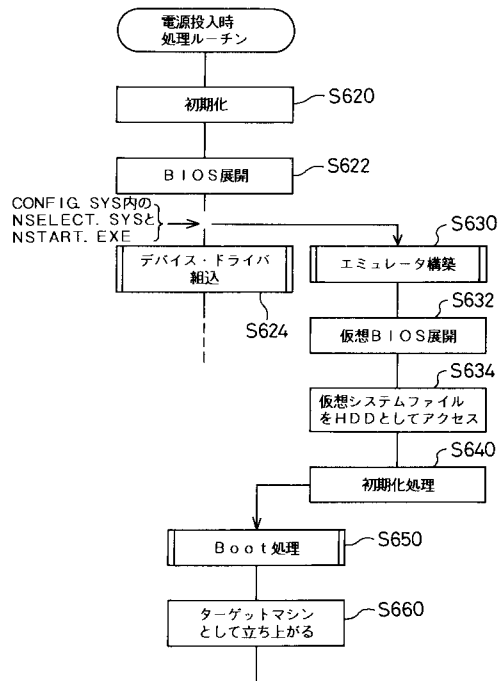


【図 31】

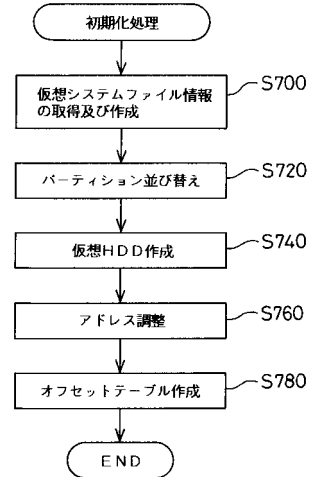
76

仮想PC セットアップ		
インストール元ドライブ	A	
インストール先ドライブ	C	
インストール先ディレクトリ	98V	
起動ドライブ	C	
仮想PC MS-DOS領域(MB単位)	10	(最大XXXXMB)
仮想PC DISK BASIC領域(MB単位)	0	(最大XXXXMB)
各種設定		
インストールして終了		
設定変更を保存して終了		
処理を選択して下さい		
[F1]:ヘルプ [F3]:セットアップ中止 [ESC]:入力放棄 [Enter]:選択確定		

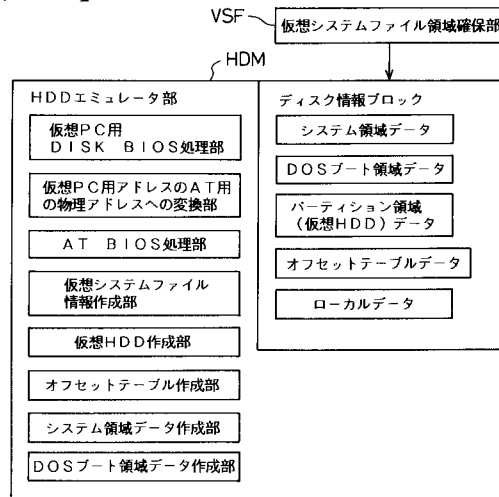
【図 3 2】



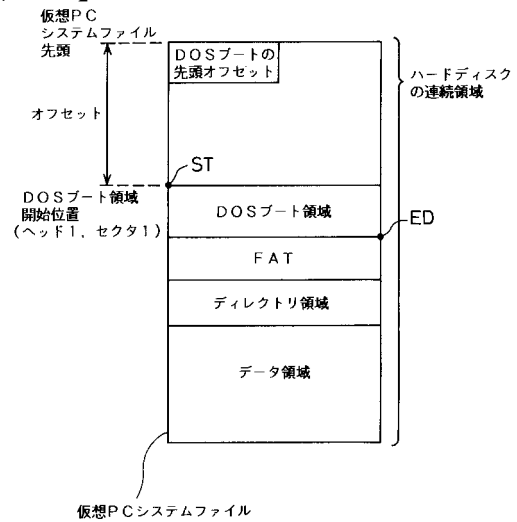
【図 3 3】



【図 3 4】

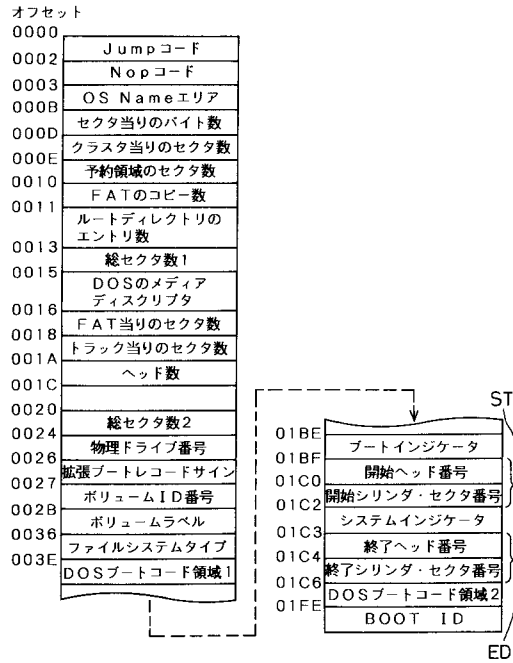


【図 3 5】

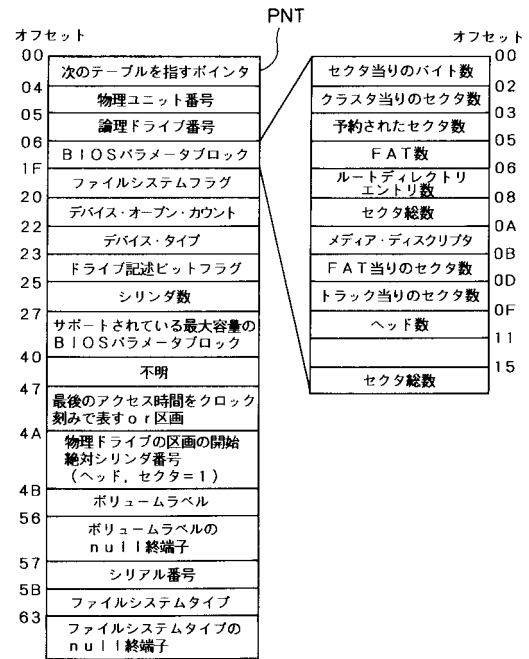


【図 36】

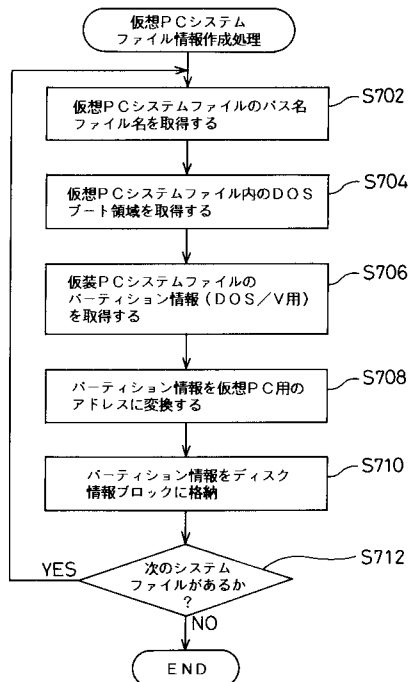
仮想PCシステムファイルのDOSブート領域の構築



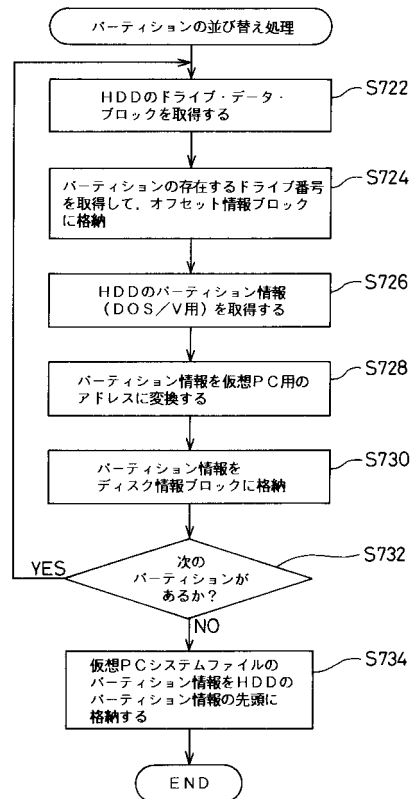
【図 37】



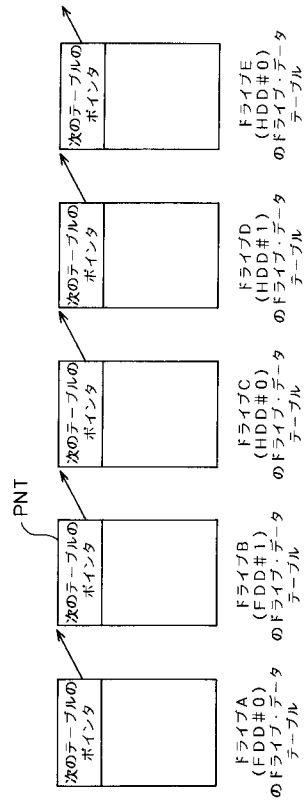
【図 38】



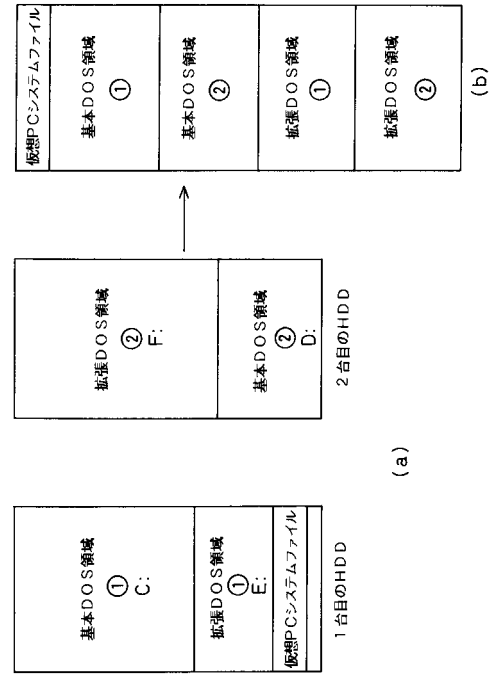
【図 39】



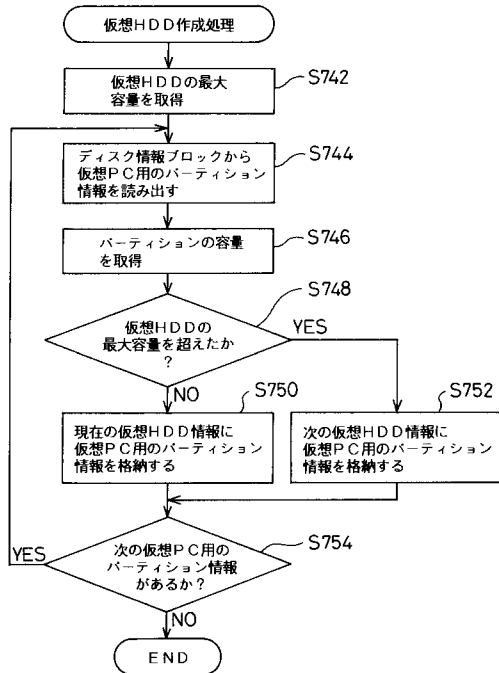
【図 40】



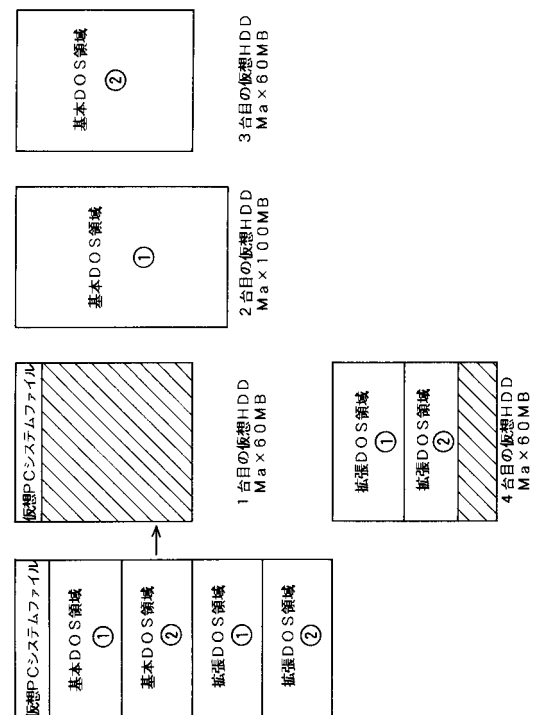
【図 41】



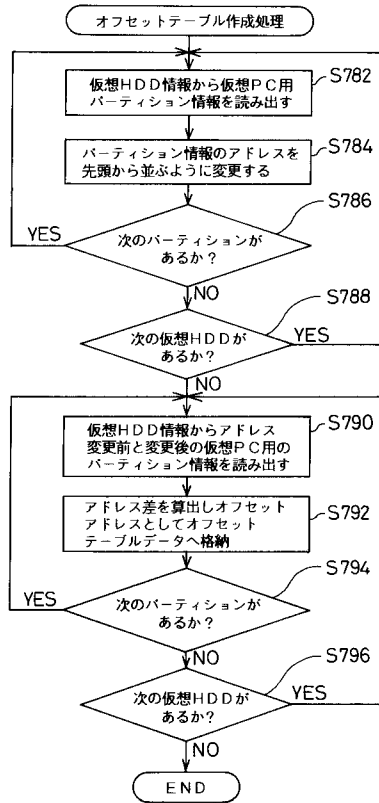
【図 42】



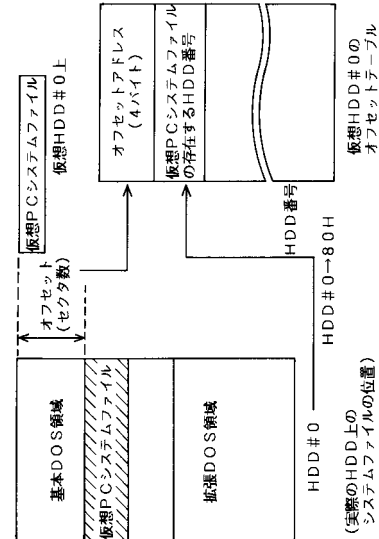
【図 43】



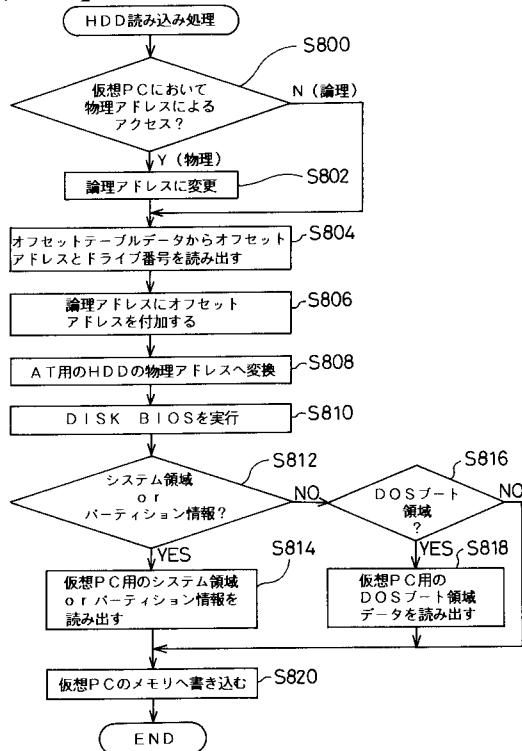
【図 4 4】



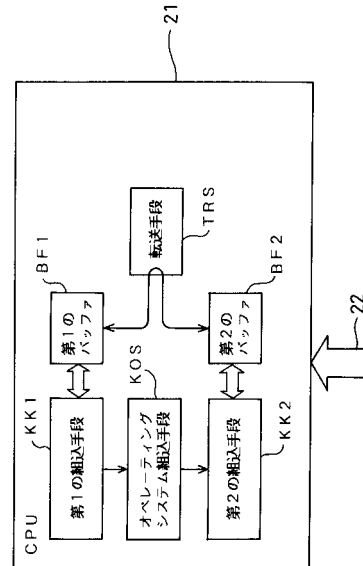
【図 4 5】



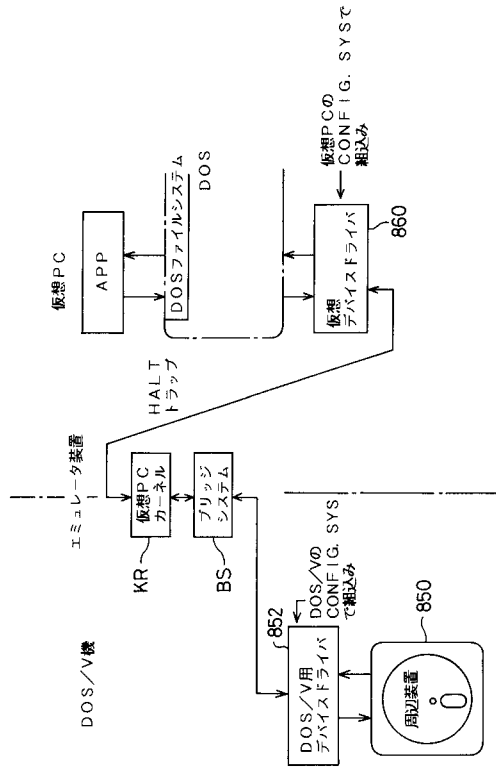
【図 4 6】



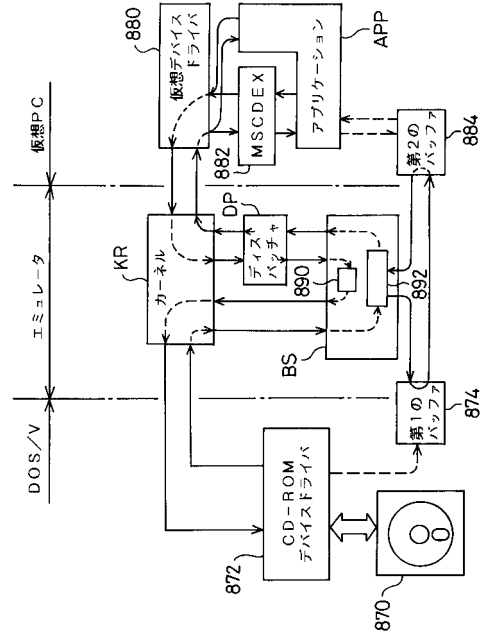
【図 4 7】



【図 48】



【図 49】



フロントページの続き

- (72)発明者 小島 政典
長野県諏訪市大和三丁目3番5号 セイコーエプソン株式会社内
(72)発明者 赤堀 豊
長野県諏訪市大和三丁目3番5号 セイコーエプソン株式会社内

審査官 石川 正二

- (56)参考文献 特開平06-242974 (JP, A)
特開平01-250150 (JP, A)

- (58)調査した分野(Int.Cl., DB名)
G06F 9/455