お客様各位

カタログ等資料中の旧社名の扱いについて

2010年4月1日を以ってNECエレクトロニクス株式会社及び株式会社ルネサステクノロジが合併し、両社の全ての事業が当社に承継されております。従いまして、本資料中には旧社名での表記が残っておりますが、当社の資料として有効ですので、ご理解の程宜しくお願い申し上げます。

ルネサスエレクトロニクス ホームページ（http://www.renesas.com）

2010年4月1日
ルネサスエレクトロニクス株式会社

【発行】ルネサスエレクトロニクス株式会社（http://www.renesas.com）

【問い合わせ先】http://japan.renesas.com/inquiry

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   標準水準： コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせないようお客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

注 1. 本資料において使用されている「当社」とは、ルネサスエレクトロニクス株式会社およびルネサスエレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

アプリケーション・ノート

RENESAS

保守／廃止

μPD72070

フロッピィ・ディスク・コントローラ

1 M/2 Mバイト FDD，
垂直磁化記録方式 4 Mバイト FDD編

資料番号　IEA-751（第1版）
発行年月　February 1997　N

保守／廃止

保守／廃止

CMOSデバイスの一般的注意事項

①静電気対策（MOS全般）

注意　MOSデバイス取り扱いの際は静電気防止を心がけてください。

MOSデバイスは強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には，NECが出荷梱包に使用している導電性のトレーやマガジン・ケース，または導電性の緩衝材，金属ケースなどを利用し，組み立て工程にはアースを施してください。プラスチック板上に放置したり，端子を触ったりしないでください。

また，MOSデバイスを実装したボードについても同様の扱いをしてください。

②未使用入力の処理（CMOS特有）

注意　CMOSデバイスの入力レベルは固定してください。

バイポーラやNMOSのデバイスと異なり，CMOSデバイスの入力に何も接続しない状態で動作させると，ノイズなどに起因する中間レベル入力が生じ，内部で貫通電流が流れて誤動作を引き起こす恐れがあります。プルアップかプルダウンによって入力レベルを固定してください。また，未使用端子が出力となる可能性（タイミングは規定しません）を考慮すると，個別に抵抗を介してV_{DD}またはGNDに接続することが有効です。

資料中に「未使用端子の処理」について記載のある製品については，その内容を守ってください。

③初期化以前の状態（MOS全般）

注意　電源投入時，MOSデバイスの初期状態は不定です。

分子レベルのイオン注入量等で特性が決定するため，初期状態は製造工程の管理外です。電源投入時の端子の出力状態や入出力設定，レジスタ内容などは保証しておりません。ただし，リセット動作やモード設定で定義している項目については，これらの動作ののちに保証の対象となります。

リセット機能を持つデバイスの電源投入後は，まずリセット動作を実行してください。

保守／廃止

本資料に掲載の応用回路および回路定数は，例示的に示したものであり，量産設計を対象とするものではありません。

○本資料の内容は，後日変更する場合があります。
○文書による当社の承諾なしに本資料の転載複製を禁じます。
○本資料に記載された製品の使用もしくは本資料に記載の情報の使用に際して，当社は当社もしくは第三者の知的所有権その他の権利に対する保証または実施権の許諾を行うものではありません。上記使用に起因する第三者所有の権利にかかわる問題が発生した場合，当社はその責を負うものではありませんのでご了承ください。
○当社は品質，信頼性の向上に努めていますが，半導体製品はある確率で故障が発生します。当社半導体製品の故障により結果として，人身事故，火災事故，社会的な損害等を生じさせない冗長設計，延焼対策設計，誤動作防止設計等安全設計に十分ご注意願います。
○当社は，当社製品の品質水準を「標準水準」，「特別水準」およびお客様に品質保証プログラムを指定して頂く「特定水準」に分類しております。また，各品質水準は以下に示す用途に製品が使われることを意図しておりますので，当社製品の品質水準をご確認の上ご使用願います。

標準水準：コンピュータ，OA機器，通信機器，計測機器，AV機器，家電，工作機械，パーソナル機器，産業用ロボット
特別水準：輸送機器（自動車，列車，船舶等），交通用信号機器，防災／防犯装置，各種安全装置，生命維持を直接の目的としない医療機器
特定水準：航空機器，航空宇宙機器，海底中継機器，原子力制御システム，生命維持のための医療機器，生命維持のための装置またはシステム等

当社製品のデータ・シート／データ・ブック等の資料で，特に品質水準の表示がない場合は標準水準製品であることを表します。当社製品を上記の「標準水準」の用途以外でご使用をお考えのお客様は，必ず事前に当社販売窓口までご相談頂きますようお願い致します。
○この製品は耐放射線設計をしておりません。

巻末にアンケート・コーナを設けております。このドキュメントに対するご意見をお気軽にお寄せください。

保守／廃止

はじめに

対象者　このアプリケーション・ノートは，μPD72070（Floppy Disk Controller）の機能を理解し，それを用いた1 M/ 2 MバイトFDD用，垂直磁化記録方式4 MバイトFDD用のアプリケーション・システムを設計するユーザを対象とします。

目　的　このアプリケーション・ノートは，次の構成に示すようなμPD72070の持つハードウエア機能およびソフトウエア機能の応用方法をユーザに理解していただくことを目的とします。

構　成　このアプリケーション・ノートは，次の内容で構成されております。

- ●システム構成例
- ●ハードウエア
- ●ソフトウエア
  - 1 M/2 MバイトFDD用デバイス・ドライバ編
  - 垂直磁化記録方式4 MバイトFDD用デバイス・ドライバ編
- ●コンパイル／リンク方法
- ●CONFIG. SYSへの登録方法

読み方　このアプリケーション・ノートを読むにあたっては，電気，論理回路，マイクロコンピュータに関する一般知識を必要とします。さらに理解を深めていただくために，**μPD72070　ユーザーズ・マニュアル**をお読みください。

凡　例

データ表記の重み	：左が上位桁，右が下位桁
アクティブ・ロウの表記	：$\overline{\times\times\times}$（端子，信号名称に上線）
注	：本文中に付けた注の説明
注意	：気をつけて読んでいただきたい内容
備考	：本文中の補足説明
数の表記	：２進数…××××または××××B
	10進数…××××
	16進数…××××H

関連資料　○μPD72070に関する資料

- ●データ・シート　（IC - 8587）
- ●ユーザーズ・マニュアル　（IEU - 816）

保守／廃止

保守／廃止

目　　次

保守／廃止

図の目次（1/2）

図の目次（2/2）

保守／廃止

表の目次（1/2）

保守／廃止

表の目次（2/2）

第１章　システム構成例

図１－１にこのアプリケーション・ノートで説明するシステム構成例を示します。

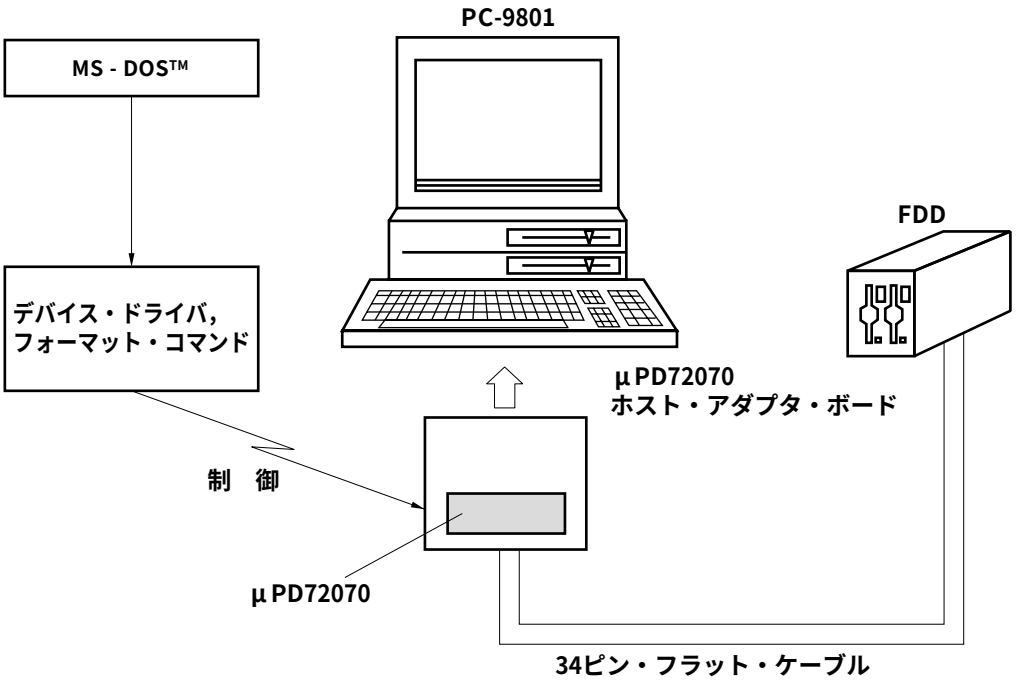
ホスト・システムにはPC-9801シリーズ（以下PC-9801），フロッピィ・ディスク・ドライブ（FDD）には，1 M/2 M バイトFDDまたはIBM社の仕様の垂直磁化記録方式4 MバイトFDDを使用します。

ホスト・システムの対象機種を表１－１に示します。

μPD72070を搭載したμPD72070ホスト・アダプタ・ボードによって，PC-9801とFDDを接続します。

μPD72070ホスト・アダプタ・ボードの制御は，1 M/2 MバイトFDD用，垂直磁化記録方式4 MバイトFDD用それぞれのデバイス・ドライバおよびフォーマット・コマンド・プログラムにより行います。

図１－１　システム構成例



保守／廃止

表１－１　PC - 9801対象機種

機　　種	CPU	動作クロック
PC-9801UX	80286	12 MHz
PC-9801RX		
PC-9801EX		
PC-9801DX		
PC-9801RS	80386SX	16 MHz
PC-9801ES		
PC-9801DS		
PC-9801FX		12 MHz
PC-9801FS		20 MHz
PC-9801US		16 MHz
PC-9801RA	80386	16/20 MHz
PC-9801DA		20 MHz
PC-9801FA	80486SX	16 MHz

備考　最新機種の参考にしてください。

第２章　ハードウエア

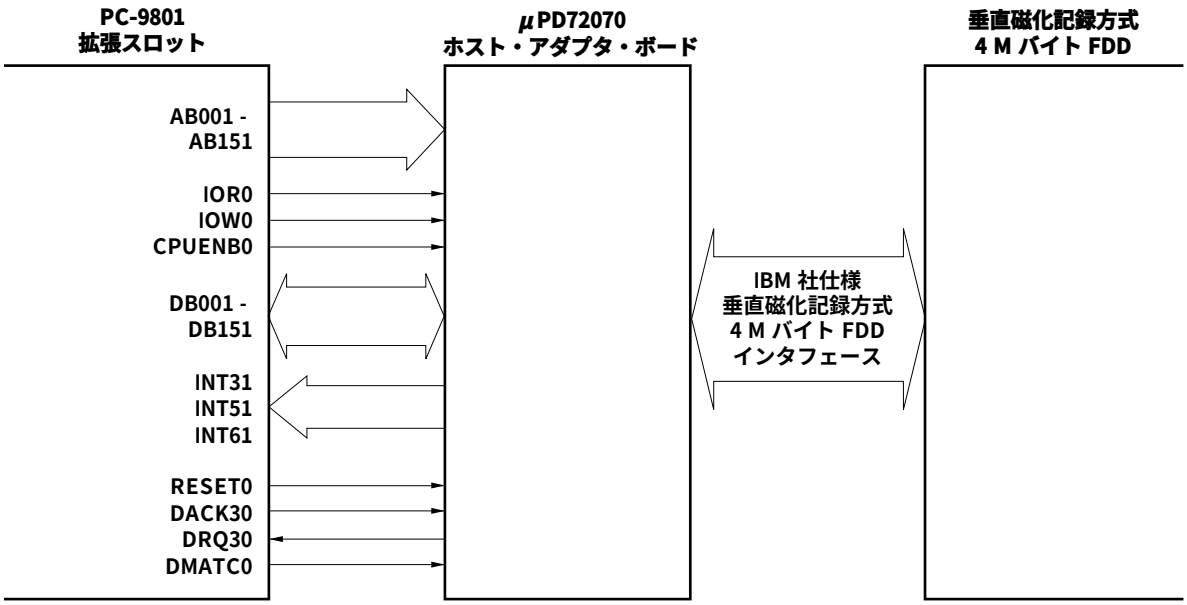
２.１　ハードウエア概要

図２－１にPC-9801とμPD72070ホスト・アダプタ・ボード，IBM社の仕様の垂直磁化記録方式4 MバイトFDDの接続ブロック図を示します。

μPD72070ホスト・アダプタ・ボードは，PC-9801の拡張スロットに差し込んで使用します。

また，μPD72070ホスト・アダプタ・ボードには，IBM社の仕様の垂直磁化記録方式4 MバイトFDDインタフェースを搭載しています。

図２－１　接続ブロック図



備考　1 M/2 MバイトFDDを接続した場合は，1 M/2 MバイトFDD用インタフェースを搭載したμPD72070ホスト・アダプタ・ボードを接続してください。

２.２　μPD72070ホスト・アダプタ・ボード

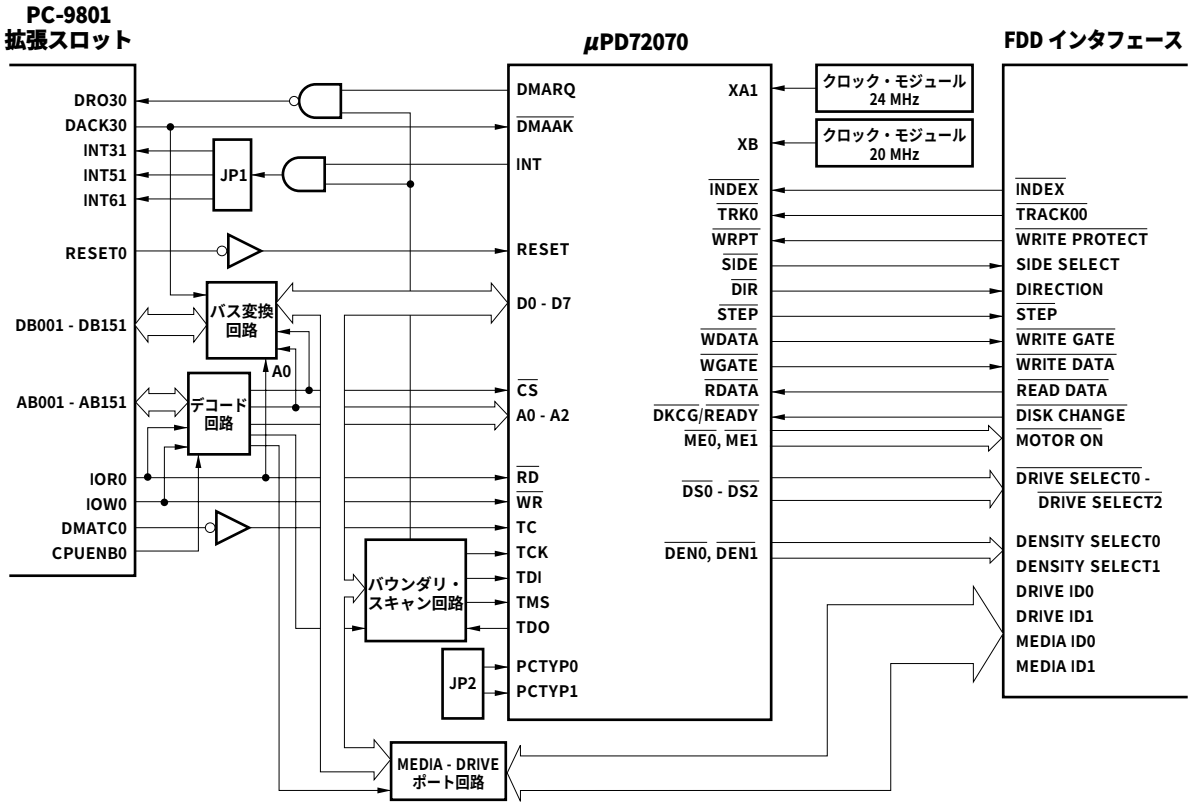
ここでは，μPD72070を使用したμPD72070ホスト・アダプタ・ボードの説明をします。

図２－２にμPD72070ホスト・アダプタ・ボードのブロック図を示します。

μPD72070ホスト・アダプタ・ボードは，割り込み処理およびDMA（Direct Memory Access）処理に，PC-9801内にあるμPD71059（マスタ），μPD8237を使用します。

なお，**付録A　回 路 図**にこのボードの回路図を添付します。

図２－２　μPD72070 ホスト・アダプタ・ボードのブロック図



保守／廃止

２.２.１　μPD72070のインターフェース・モード設定

μPD72070のホスト・インタフェースのモードは，ジャンパJP2によって選択できます。JP2については**２.２.６　ディップ・スイッチおよびジャンパ**を参照してください。

ドライブ・インタフェースのモードは，スタンダードFDDモードです。FDDはIBM社の仕様の垂直磁化記録方式4 MバイトＦＤＤまたは，1 M/2 MバイトＦＤＤを使用します。システム・クロックは，XA1端子に24 MHzを，XB端子に20 MHzを入力します。

なお，FDDへのデータ転送レートは表２－１のようになります。

表２－１　データ転送レート

FDD	データ転送レート
1 M	250 kbps
2 M（1.6 M）	500 kbps
4 M	1 Mbps

注意　μPD72070のDKCG/READY端子をディスク・チェンジ信号で使用する場合，μPD72070の内部でREADY信号が常にアクティブになるため，μPD72070をリセット（ハードウエア／ソフトウエア・リセット）した場合，ドライブ４台分の状態遷移割り込みが発生します。
イニシャライズ時は，SENSE INTERRUPT STATUSコマンドを発行し，コマンドがINVALID COMMAND（ST0 = 80H）になるまで発行を繰り返してください。

２.２.２　I/Oアドレス

μPD72070ホスト・アダプタ・ボード内にあるI/Oポート（ライト・ポート１，リード・ポート１，２）とμPD72070のI/Oアドレスを表２－２に示します。アドレス・ビット15 - 10はディップ・スイッチで決定します。ディップ・スイッチについては**２.２.６　ディップ・スイッチおよびジャンパ**を参照してください。

注意　μPD72070ホスト・アダプタ・ボードのI/Oポートは，ワード・アクセスできません。

表２－２　μPD72070ホスト・アダプタ・ボードのI/Oアドレス

レジスタ名	R/W	I/Oアドレス															
		A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
ステータス・レジスタA	R	表２－３参照						0	0	1	1	0	1	0	0	0	0
ステータス・レジスタB	R							0	0	1	1	0	1	0	0	1	0
ディジタル・アウトプット・レジスタ	R/W							0	0	1	1	0	1	0	1	0	0
テープ・ドライブ・レジスタ	R/W							0	0	1	1	0	1	0	1	1	0
ステータス・レジスタ	R							0	0	1	1	0	1	1	0	0	0
データ・レート・レジスタ	W																
データFIFOレジスタ	R/W							0	0	1	1	0	1	1	0	1	0
使用禁止								0	0	1	1	0	1	1	1	0	0
ディジタル・インプット・レジスタ	R							0	0	1	1	0	1	1	1	1	0
コンフィギュレーション・コントロール・レジスタ	W																
ライト・ポート１	W							1	0	1	1	0	1	0	0	0	0
リード・ポート１	R																
リード・ポート２	R							1	0	1	1	0	1	0	0	1	0

２.２.３　バウンダリ・スキャン回路

この回路には，ライト・ポート１とリード・ポート１があります。

ライト・ポート１の構成を次に説明します。

TCK	μPD72070のバウンダリ・スキャン回路へのクロック入力信号をライトします。
TMS	μPD72070のバウンダリ・スキャン回路のTAPコントローラの状態遷移入力信号をライトします。
TDI	μPD72070のバウンダリ・スキャン回路のデータ入力信号をライトします。
MASK	μPD72070のINT出力とDMARQ出力のマスク解除を行います。 （マスク解除 =１をセット）
ACTIVE READY	μPD72070へ強制的に入力するREADY信号をライトします。 （READY =１をセット）
MOTOR ON	μPD72070のホスト・インタフェース・モードがGeneralモードのときに，FDDのモータの制御（ON/OFF）を行います。

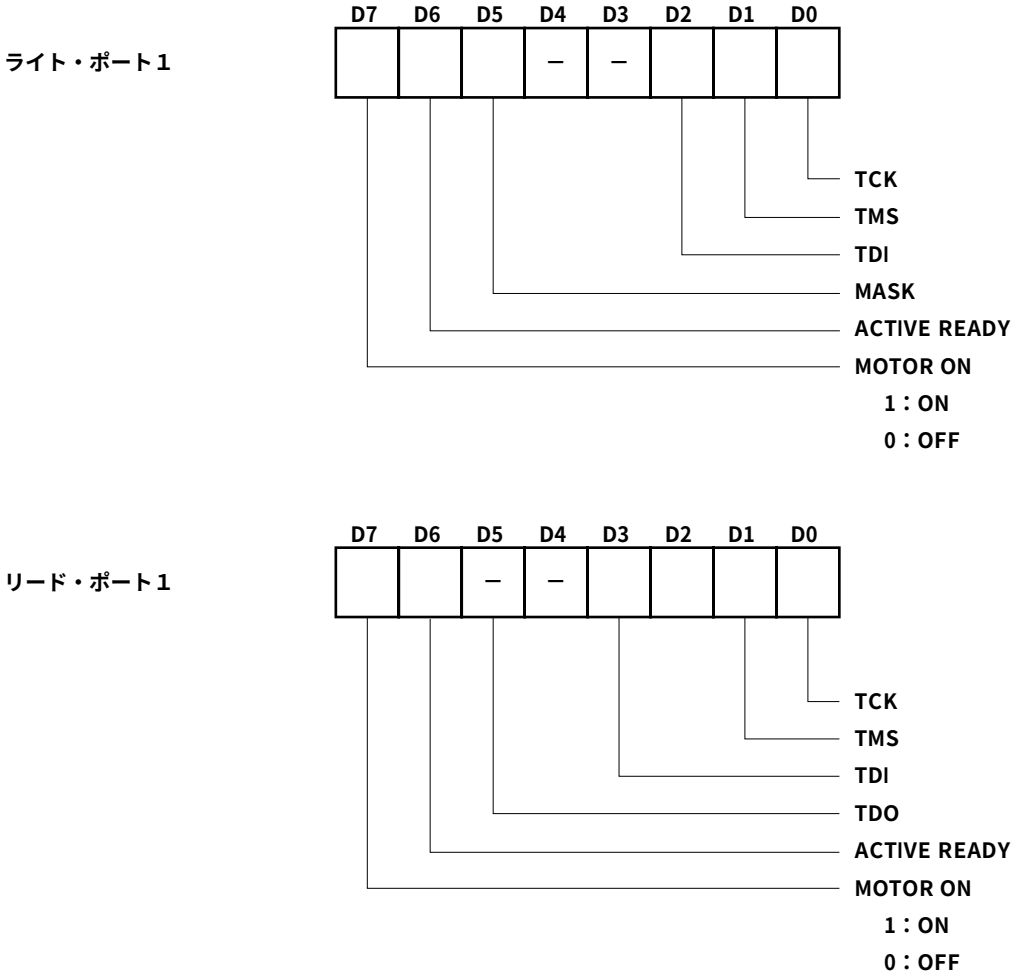
リード・ポート１の構成を次に説明します。

TCK	μPD72070のバウンダリ・スキャン回路へのクロック入力信号をリードします。
TMS	μPD72070のバウンダリ・スキャン回路のTAPコントローラの状態遷移入力信号をリードします。
TDI	μPD72070のバウンダリ・スキャン回路のデータ入力信号をリードします。
TDO	μPD72070のバウンダリ・スキャン回路のデータ出力信号をリードします。
ACTIVE READY	μPD72070へ強制的に入力されているREADY信号の状態をリードします。
MOTOR ON	μPD72070のホスト・インタフェース・モードがGeneralモードのときに，FDDのモータの制御（ON/OFF）状況をリードします。

図２－３に各ポートの割り当て状態を示します。

注意　μPD72070ホスト・アダプタ・ボードは，ハードウエア・リセット後にμPD72070のINT出力とDMARQ出力にマスクがかかります。
MASKビットは，マスクの解除以降，ハードウエア・リセットを行うまで１をライトしてください。

図２－３　ライト・ポート１，リード・ポート１



－：Don't care

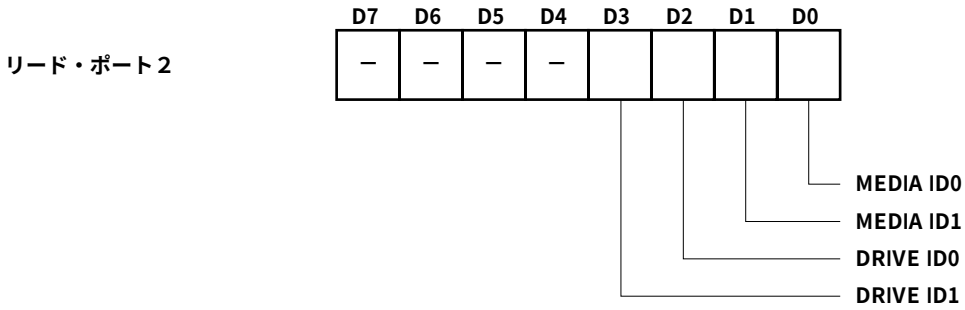
２.２.４　MEDIA - DRIVEポート回路

この回路には，リード・ポート２があります。

リード・ポート２は，FDDインタフェースのDRIVE ID0, DRIVE ID1, MEDIA ID0, MEDIA ID1の信号をリードできます。

図２－４にリード・ポート２の割り当て状態を示します。

図２－４　リード・ポート２



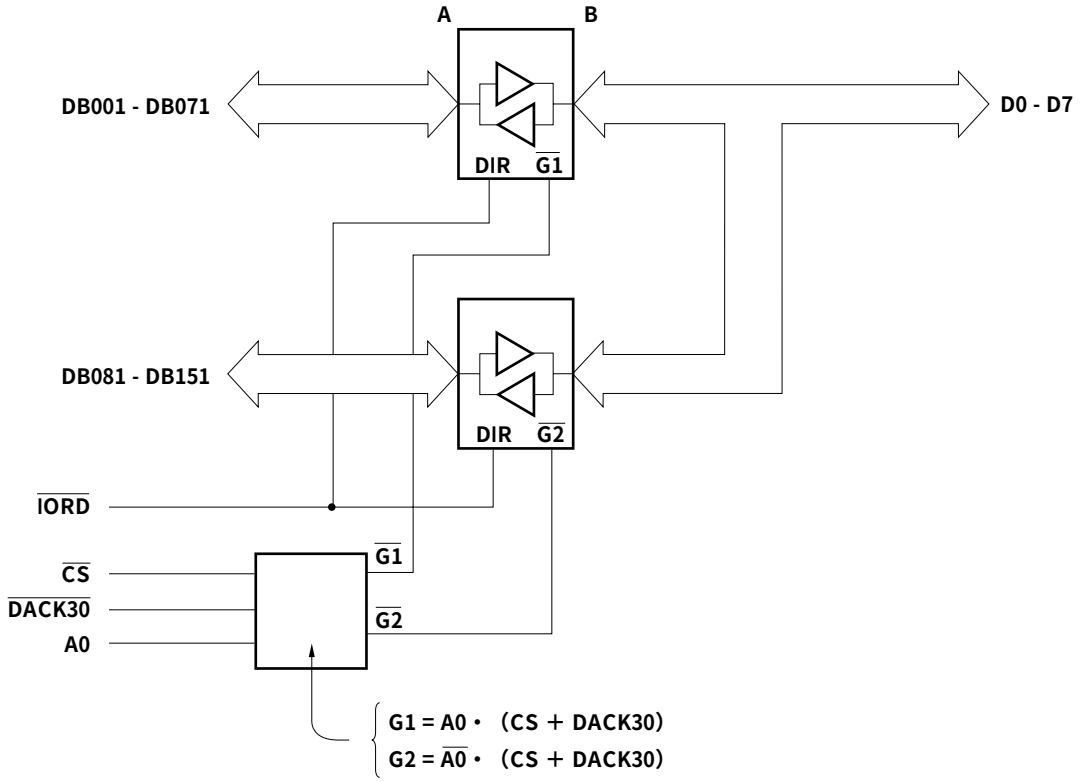
MEDIA TYPE ID1	MEDIA TYPE ID0	メディア・タイプ
L	L	Reserved
L	H	4 MB（2ED）
H	L	2 MB（2HD）
H	H	1 MB（2DD）

DRIVE TYPE ID1	DRIVE TYPE ID0	ドライブ・タイプ
L	L	3.5インチ，1.44 MB
L	H	3.5インチ，2.88 MB
H	L	5.25インチ，1.2 MB
H	H	Reserved

２．２．５　バス変換回路

この回路は，PC-9801側のデータ・バスとボード側のデータ・バスを接続するインタフェース回路です。図２－５にブロック図を示します。

図２－５　バス変換回路



・IORDがアクティブになるとB → Aにデータが出力されます。

・IORDがインアクティブのときはA → Bにデータが出力されます。

A：PC-9801データ・バス

B：ボード上のデータ・バス

注意　図２－５の中の論理式は，図中の信号名のアクティブ・レベルを示すバーとは関係なく，信号がインアクティブの場合にバーが付けてあります。

保守／廃止

２.２.６　ディップ・スイッチおよびジャンパ

ここでは，μPD72070ホスト・アダプタ・ボードに搭載されるディップ・スイッチおよびジャンパについて説明します。

（１）I/Oアドレスの設定

μPD72070の各レジスタとホスト・アダプタ・ボード上のI/Oポート（ライト・ポート１，リード・ポート１，２）のI/Oアドレスの上位６ビット分をディップ・スイッチSW1で設定します。表２－３に設定状態を示します。

表２－３　I/Oアドレス設定

SW1								μPD72070各レジスタ I/Oアドレス		I/Oポート I/Oアドレス	
8	7	6	5	4	3	2	1	A15 - A8	A7 - A0	A15 - A8	A7 - A0
－	－	OFF	OFF	OFF	OFF	OFF	OFF	00H		02H	
－	－	OFF	OFF	OFF	OFF	OFF	ON	04H		06H	
－	－	OFF	OFF	OFF	OFF	ON	OFF	08H		0AH	
－	－	OFF	OFF	OFF	OFF	ON	ON	0CH		0EH	
		・ ・ ・			・ ・ ・			・ ・ ・	表２－２ 参照	・ ・ ・	表２－２ 参照
－	－	ON	ON	ON	ON	OFF	ON	F4H		F6H	
－	－	ON	ON	ON	ON	ON	OFF	F8H		FAH	
－	－	ON	ON	ON	ON	ON	ON	FCH		FCH	

（２）割り込み要求信号の選択

μＰＤ７２０７０からの割り込み信号は，ジャンパＪＰ1により拡張スロット上の割り込み要求信号（INT31, INT51, INT61）を選択できます。

図２－６にJP1の接続図を，表２－４に選択される割り込み要求信号を示します。

図２－６　JP1

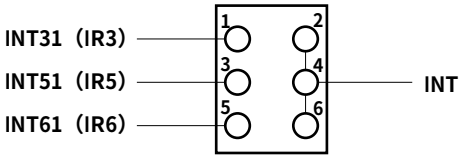


表２－４　選択割り込み要求信号

接　続	割り込み要求信号
1 - 2	INT31（IR3）
3 - 4	INT51（IR5）
5 - 6	INT61（IR6）

（３）μPD72070のホスト・インタフェース・モードの選択

μPD72070のホスト・インタフェースのモードは，ジャンパJP2で選択できます。図２－７にJP2の接続図を，表２－５に選択するホスト・インタフェースのモードを示します。

図２－７　JP2

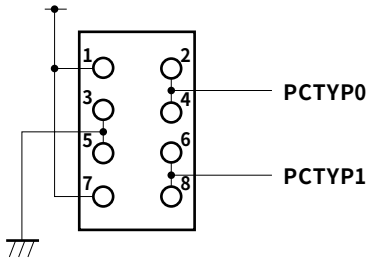


表２－５　ホスト・インタフェース・モード

ホスト・インタフェースのモード	PCTYP0	PCTYP1
Generalモード	1 - 2	5 - 6
Appleモード	1 - 2	7 - 8
PS/2モード	3 - 4	5 - 6
PC-ATモード	3 - 4	7 - 8

（４）MOTOR ON信号の選択

ジャンパJP3は，μPD72070のホスト・インタフェースにGeneralモードを選択した場合に，ライト・ポート１からFDDへMOTOR ON信号を入力できるようにするジャンパです。

PC-AT，PS/2モードではμPD72070からのMOTOR ON信号をFDDへ入力できるようにジャンパを選択します。

図２－８にJP3の接続図を，表２－６に選択する信号を示します。

図２－８　JP3

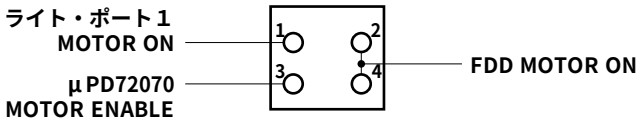


表２－６　MOTOR ON信号

ホスト・インタフェースのモード	ジャンパ
Generalモード	1 - 2
PC-AT, PS/2モード	3 - 4

（５）DISK CHANGE/READY信号選択

ジャンパJP4はREADY信号を使用する場合，イニシャライズ時，強制的にFDCへREADY信号が入力できるようにするジャンパです。

またジャンパJP5は，FDCのDKCG/READY端子をDISK CHANGE信号，またはREADY信号のどちらで使用するか選択するジャンパです。

図２－９，２－10にJP4, JP5の接続図を，表２－７に接続表を示します。

図２－９　JP4

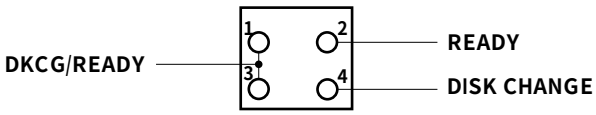


図２－10　JP5

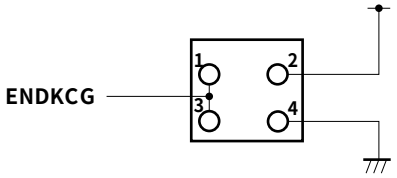


表２－７　DISK CHANGE/READY信号選択

FDDの信号	JP4	JP5
READY	1 - 2	1 - 2
DISK CHANGE	3 - 4	3 - 4

保守／廃止

２.３　割り込みコントローラ

PC-9801内の割り込みコントローラ（μPD71059）は，マスタを使用し，割り込み要求信号（IR3, IR5, IR6）をジャンパJP1で選択できます。JP1については**２.２.６　ディップ・スイッチおよびジャンパ**を参照してください。

割り込みコントローラのI/Oアドレス，割り込みベクタ番号をそれぞれ表２－８，表２－９に示します。

PC-9801割り込みコントローラ・レジスタIMW（OCW1），PFCW（OCW2），MCW（OCW3）は，I/Oポートに書き込むことで制御を行います。ほかのレジスタはPC - 9801の初期設定値を使用します。

表２－８　割り込みコントローラのI/Oアドレス

μPD71059 レジスタ名	PC-9801割り込み コントローラ・レジスタ名	I/Oアドレス	初期値
IW1	ICW1（マスタ）	00H	11H
IW2	ICW2（マスタ）	02H	08H
IW3	ICW3（マスタ）	02H	80H
IW4	ICW4（マスタ）	02H	IDH
IMW	OCW1（マスタ）	02H	－
PFCW	OCW2（マスタ）	00H	－
MCW	OCW3（マスタ）	00H	－

表２－９　割り込みベクタ番号

割り込み要求信号	ベクタ番号
IR3	0BH
IR5	0DH
IR6	0EH

保守／廃止

２.４　DMAコントローラ

PC-9801内のDMAコントローラ（μPD8237）は，チャネル３を使用します。

表２－10にDMAコントローラのI/Oアドレスとリセット後の初期値を示します。

なお，バンク・アドレス・オートインクリメント・モードは，16 Mバイト・インクリメント・モードを使用します。

また，転送モードはシングル・モードとし，イニシャライズを行うときには必ずチャネル３をマスクしてから行います。

表２－10　DMAコントローラのI/Oアドレス

μPD8237のレジスタ名	PC-9801DMAコントローラのレジスタ名	I/Oアドレス	初期値
デバイス・コントロール	コマンド・レジスタ	11H	40H
モード・コントロール	モード・レジスタ	17H	－
リクエスト・コントロール	ライト・リクエスト	13H	－
マスク・コントロール（１チャネル）	ライト・シングル・マスク	15H	－
マスク・コントロール（全チャネル）	ライト・オール・マスク	1FH	－
チャネル３セット・アドレス	アドレス・レジスタ	0DH	－
チャネル３セット・カウント	カウント・レジスタ	0FH	－
クリア・オール・マスク	クリア・マスク	1DH	－
ソフトウエア・リセット	マスタ・クリア	1BH	－
アドレス・ロウ・バイト	クリア・バイト・ポインタF/F	19H	－
－	チャネル３バンク	25H	－
－	バンク・アドレス・オートインクリメント・モード	29H	－

注意　このマニュアルで使用しているμPD72070ホスト・アダプタ・ボードは，DMA転送タイミングを保証していません。

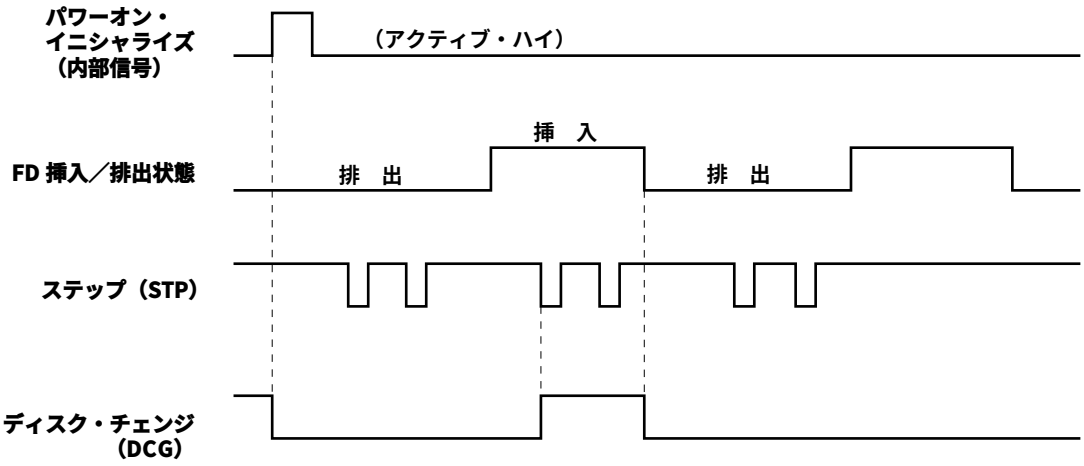
保守／廃止

２.５　ディスク・チェンジ信号の処理

ここでは，フロッピィ・ディスクの挿入／排出を判断するディスク・チェンジ信号の処理方法について説明します。

ディスク・チェンジ信号は，フロッピィ・ディスクがドライブに挿入されステップ信号を判断するとインアクティブ状態になります。フロッピィ・ディスクが排出されるとただちにアクティブ状態になります。図２－11にディスク・チェンジ信号のタイミングを示します。

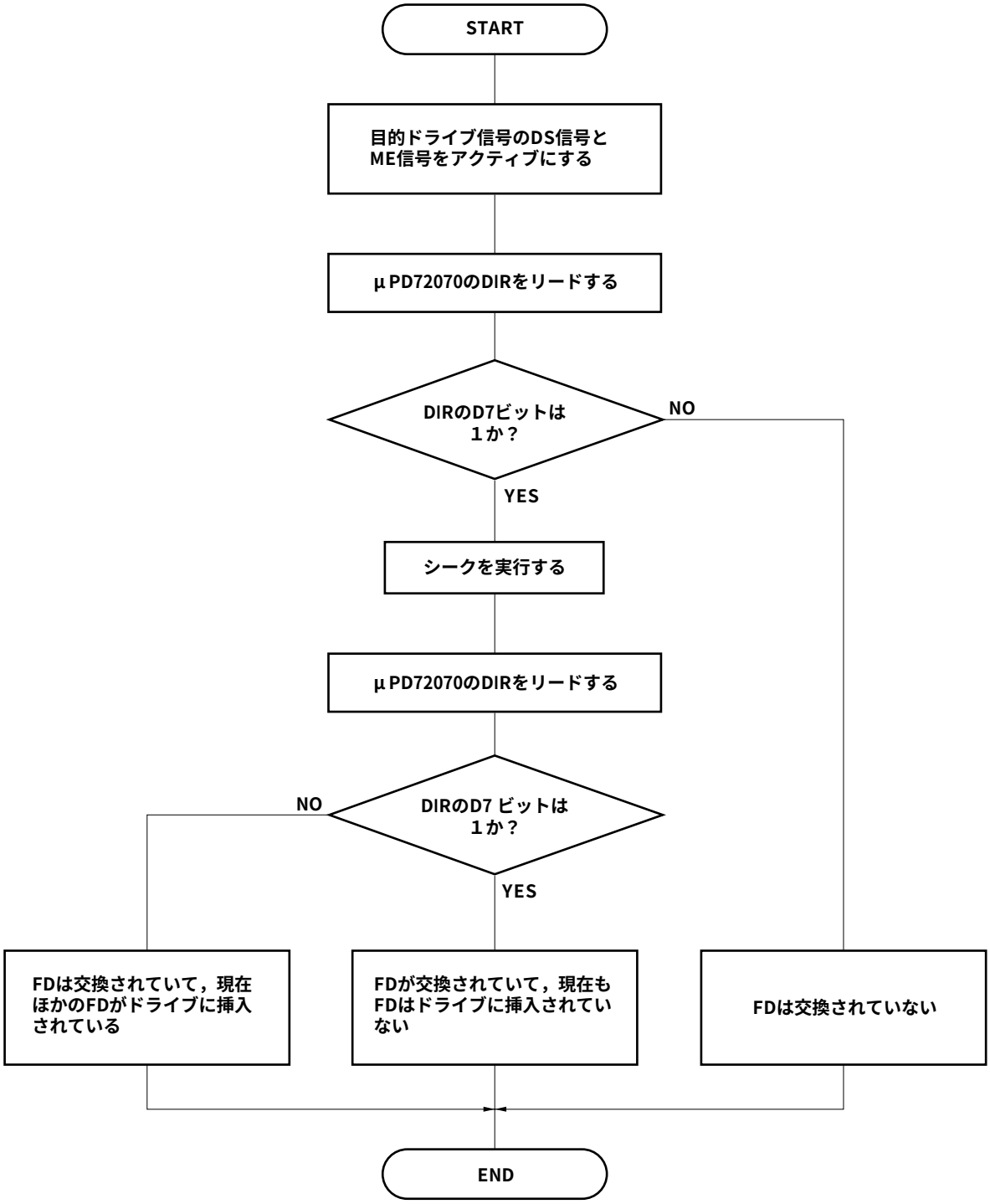
図２－11　ディスク・チェンジ信号のタイミング



μPD72070では，ホスト・インタフェースのモードにPS/2またはPC-ATモードを選択することで，ディジタル・インプット・レジスタ（DIR）が使用できます。DIRはμPD72070のDKCG端子入力の状態を見ることができます。このDKCG端子にディスク・チェンジ信号を入力し，DIRをリードすることでフロッピィ・ディスクの交換状態が検出できます。

図２－12にディスク・チェンジ信号を使用したフロッピィ・ディスク交換状態の検出方法のフロー・チャートを示します。

図２－12　フロッピィ・ディスク交換状態の検出フロー・チャート



保守／廃止

第３章　ソフトウエア

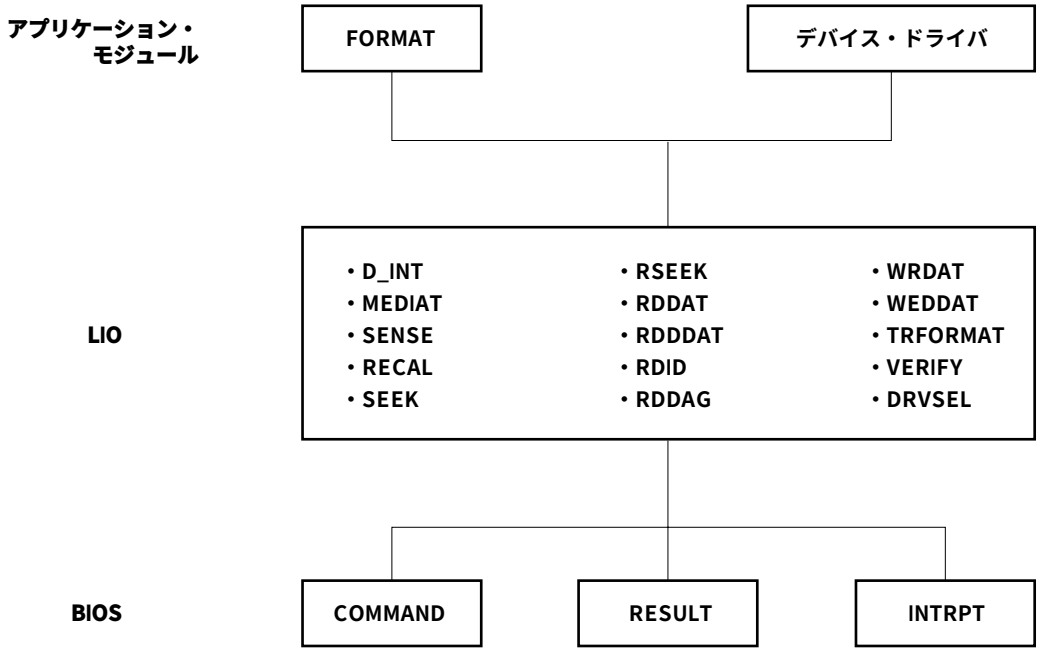
第３章では，1 M/2 MバイトFDDまたは垂直磁化記録方式の4 MバイトFDDにアクセスするためのデバイス・ドライバとフォーマット・コマンドについて説明をします。

３.１　ソフトウエア概要

図３－１にソフトウエアのモジュール構成図を示します。

モジュール構成は，1 M/2 MバイトFDD用も垂直磁化記録方式4 MバイトFDD用も同じです。

図３－１　モジュール構成



３.２　１M/2 M バイトFDD用デバイス・ドライバ編

３.２.１　BIOS部

BIOSモジュールはFDCへ直接アクセスするモジュールです。

表３－１にBIOSモジュールの機能を示します。

表３－１　BIOSモジュール機能

モジュール名	機　　能	ページ
COMMAND	FDCへのコマンドの書き込み処理	p.21
RESULT	FDCからのリザルト・ステータスの読み込み処理	p.22
INTRPT	FDCの割り込み処理	p.23

（1）BIOSモジュールの説明

ここではBIOSモジュールの説明をします。表３－２に示す内容に従って各モジュールを説明し，SPD（Structured programming diagrams）チャートを示します。SPDチャートについては，**付録C SPDチャートの説明**を参照してください。

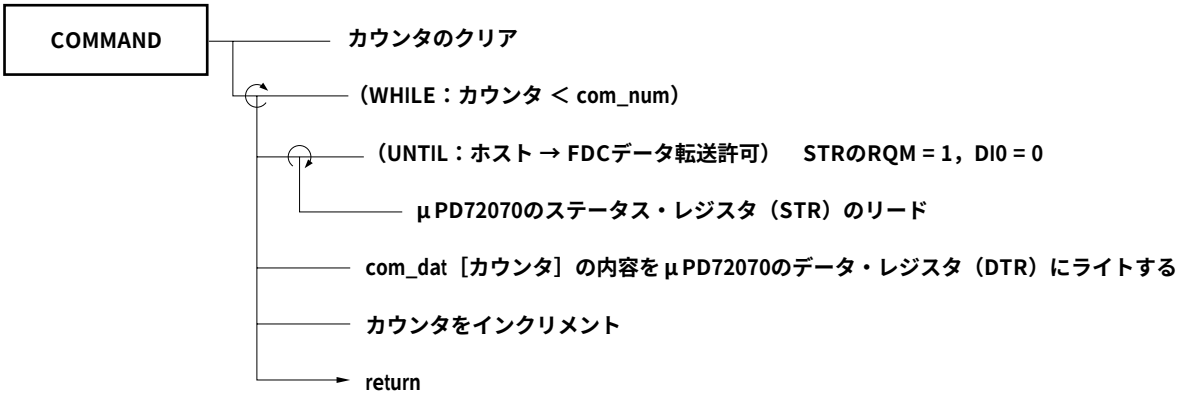
表３－２　各モジュールの説明内容

モジュール名	モジュール名を示します。
言　　語	関数がどの言語で作成されているかを示します。
入　　力	入力時の変数およびレジスタを示します。
出　　力	出力時の変数およびレジスタを示します。
［機　　能］ 関数の処理内容を示します。	

（a）コマンド・ライト処理

モジュール名	COMMAND
言　　語	C言語
入　　力	com_dat[ ]，com_num
出　　力	なし
［機　　能］ FDCへcom_datの内容をcom_numが示すバイト数だけライトします。	

SPD-1



（b）リザルト・リード処理

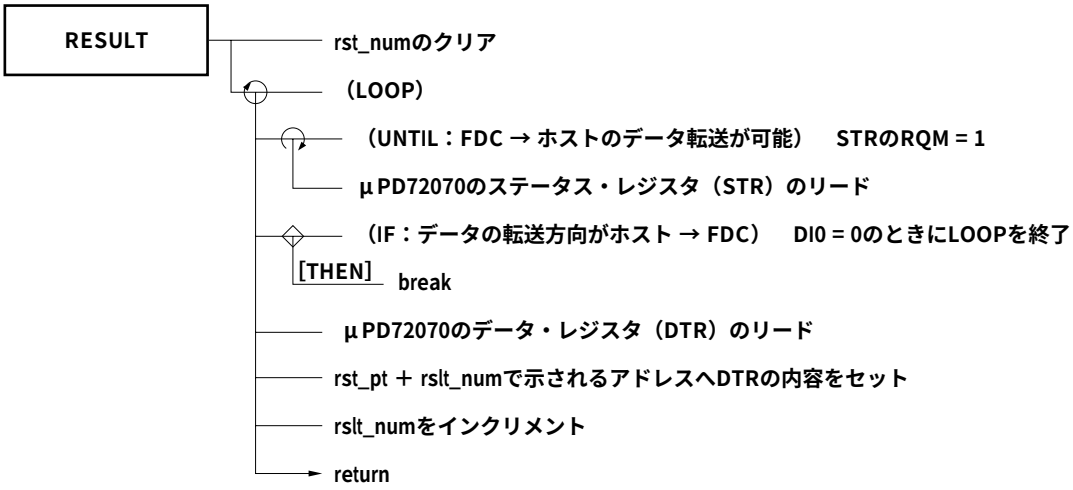
モジュール名	RESULT
言　　語	C言語
入　　力	なし
出　　力	rslt_pt [ ], rslt_num

［機　　能］

（ｉ）rslt_ptの示す領域へFDCからリードしたリザルト・ステータスをセットします。

（ii）リードしたリザルト・ステータスのバイト数をrslt_numへセットします。

SPD-2



（ｃ）割り込み処理

モジュール名	INTRPT
言　　語	C言語
入　　力	なし
出　　力	int_s, seek_r [ ], ready

［機　　能］

μPD72070からの割り込み要因（リード／ライト系，シーク系，状態遷移系）を解析して，各要因ごとに割り込み処理を行います。

（ｉ）リード／ライト系

リザルト・リード処理を実行し，FDCからのリザルト・ステータスをリードします。結果はRESULTモジュールによってrslt_ptとrslt_numにセットされます。

（ii）シーク系

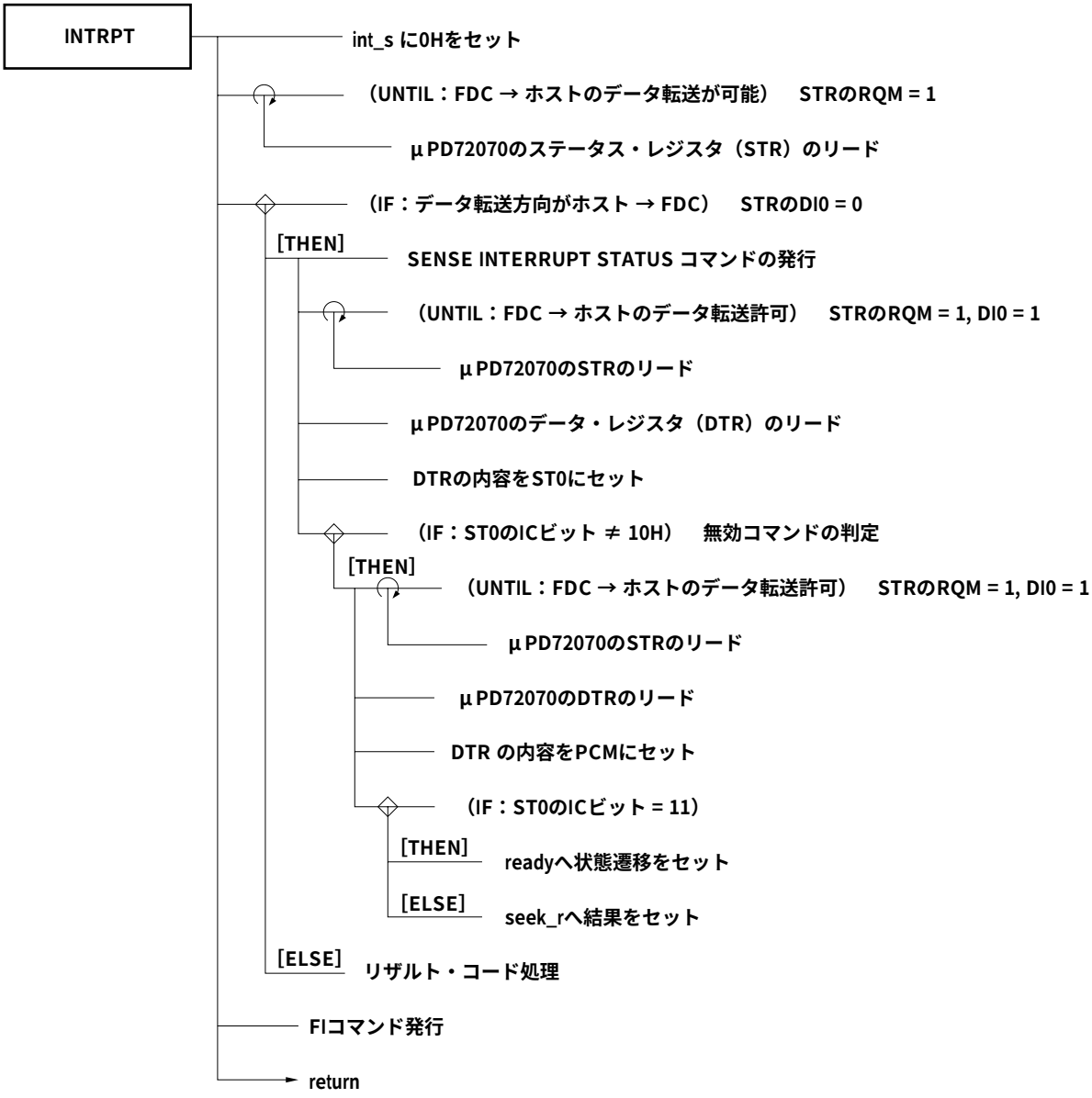
SENSE INTERRUPT STATUSコマンドをFDCに発行し，FDCからリードしたリザルト・ステータスをseek_r [ ] にセットします。

（iii）状態遷移系

SENSE INTERRUPT STATUSコマンドをFDCに発行し，FDCからリードしたリザルト・ステータスを元に，readyに情報をセットします。

注意　INTRPTモジュールは，int_sに01Hをセットして，アプリケーション・モジュールに割り込みが発生したことを知らせます。アプリケーション・モジュールでは，int_sの01Hを確認しint_sをリセット（00Hをセット）します。

SPD-3

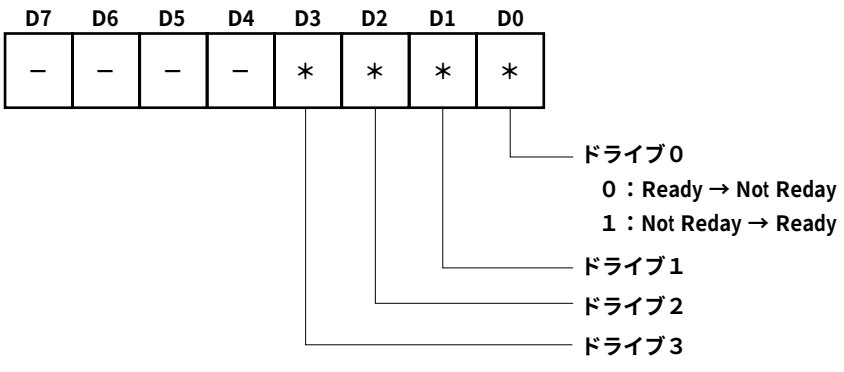

INTRPT
int_s に0Hをセット
（UNTIL：FDC → ホストのデータ転送が可能）　STRのRQM = 1
μPD72070のステータス・レジスタ（STR）のリード
（IF：データ転送方向がホスト → FDC）　STRのDI0 = 0
[THEN]
SENSE INTERRUPT STATUS コマンドの発行
（UNTIL：FDC → ホストのデータ転送許可）　STRのRQM = 1, DI0 = 1
μPD72070のSTRのリード
μPD72070のデータ・レジスタ（DTR）のリード
DTRの内容をST0にセット
（IF：ST0のICビット ≠ 10H）　無効コマンドの判定
[THEN]
（UNTIL：FDC → ホストのデータ転送許可）　STRのRQM = 1, DI0 = 1
μPD72070のSTRのリード
μPD72070のDTRのリード
DTR の内容をPCMにセット
（IF：ST0のICビット = 11）
[THEN]
readyへ状態遷移をセット
[ELSE]
seek_rへ結果をセット
[ELSE]
リザルト・コード処理
FIコマンド発行
return

（２）BIOSの変数

ここでは，BIOSが使用する変数について説明します。

表３－３に変数と内容を示します。

表３－３　BIOSの変数

変数名	機　　能
com_dat［］ コマンド・データ （１バイト）	μPD72070のC - Phaseで書き込むコマンド・パラメータをセットする変数です。
com_num コマンド・ナンバ （１バイト）	コマンド・パラメータのバイト数をセットする変数です。
rslt_pt［］ リザルト・ポインタ （１バイト）	μPD72070のR - Phaseで受け取ったリザルト・ステータス・バイトとパラメータをセットする変数です。
rslt_num リザルト・ナンバ （１バイト）	受け取ったリザルト・ステータス・バイトとパラメータのバイト数をセットする変数です。
int_s 割り込みステータス （１バイト）	割り込みが発生したことを示す変数です。 割り込みが発生すると01Hがセットされます。
seek_r［］ シーク・リザルト （構造体）	シーク系コマンドのリザルト・ステータスとヘッドの位置をドライブごとにセットする変数です。
ready ドライブ・レディ （１バイト）	ドライブに状態遷移が発生するとINTRPTモジュールでその情報をセットする変数です。  －：Don't care

保守／廃止

３．２．２　LIO部

LIOモジュールはFDCへの基本的なコントロールを行うモジュールです。

図３－２にLIOの基本動作を，表３－４にLIOの機能を示します。

図３－２　LIOの基本動作

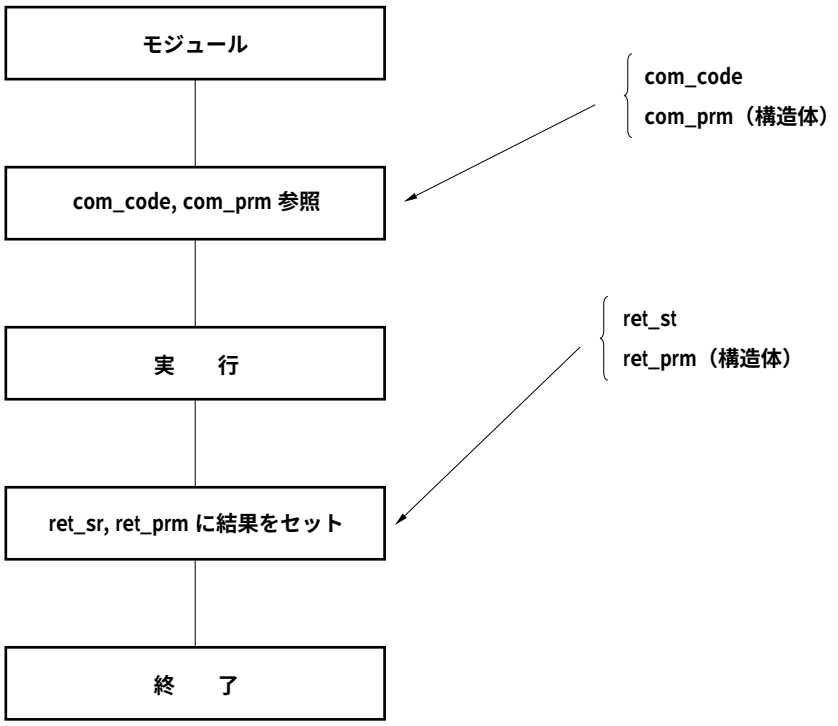


表３－４　LIO機能

モジュール名	機　　能	ページ
D_INT	FDCのイニシャライズと，接続されているFDDを調べます。パワーオン時に実行します。	p.28
MEDIAT	メディア・タイプ（2DD, 2HD）を変更します。	p.30
SENSE	指定されたドライブの情報を得ます。	p.32
RECAL	指定されたシリンダ番号へヘッドを移動します。	p.34
SEEK	指定されたシリンダ数だけヘッドを移動します。	p.36
RSEEK	シリンダ０へヘッドを移動します。	p.38
RDDAT	DAM付きデータをリードします。	p.40
RDDDAT	DDAM付きデータをリードします。	p.41
RDID	トラック上の任意のID情報をリードします。	p.43
RDDAG	指定トラックのデータをリードし，トラックに書き込まれているデータの状態を診断します。	p.45
WRDAT	DAM付きデータをライトします。	p.47
WRDDAT	DDAM付きデータをライトします。	p.48
TRFORMAT	１トラック分のフォーマットします。	p.50
VERIFY	指定したセクタのベリファイ（CRCチェック）します。	p.52
DRVSEL	指定したドライブ番号のDS信号とME信号をアクティブにします。	p.54
TIME	タイマの割り込みがあったことを示すフラグを立てます。	p.55

（1）LIOモジュール説明

ここではLIOモジュールの説明をします。表３－５に示す内容に従って各モジュールを説明し，SPDチャートを示します。

表３－５　各モジュール説明内容

モジュール名	モジュール名を示します。
言　　語	関数がどの言語で作成されているかを示します。
入　　力	入力時の変数およびレジスタを示します。
出　　力	出力時の変数およびレジスタを示します。
［機　　能］ 関数の処理内容を示します。また，関数の入出力変数の内容も示します。	

（a）デバイス・イニシャライズ処理

モジュール名	D_INT
言　　語	C 言語
入　　力	なし
出　　力	d_exist

［機　　能］

（ｉ）割り込みコントローラのマスクを行い，割り込みベクタの設定をします。

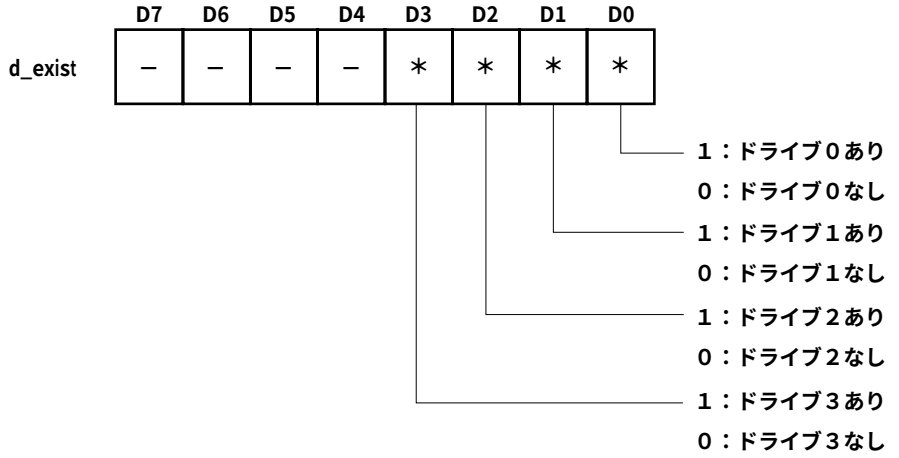
（ii）DMAコントローラのチャネル３のマスクを行い，DMAコントローラの設定をします。

（iii）μPD72070のリセット後の割り込み（ドライブの状態遷移）をクリアします。

（iv）割り込みコントローラのマスクを解除します。

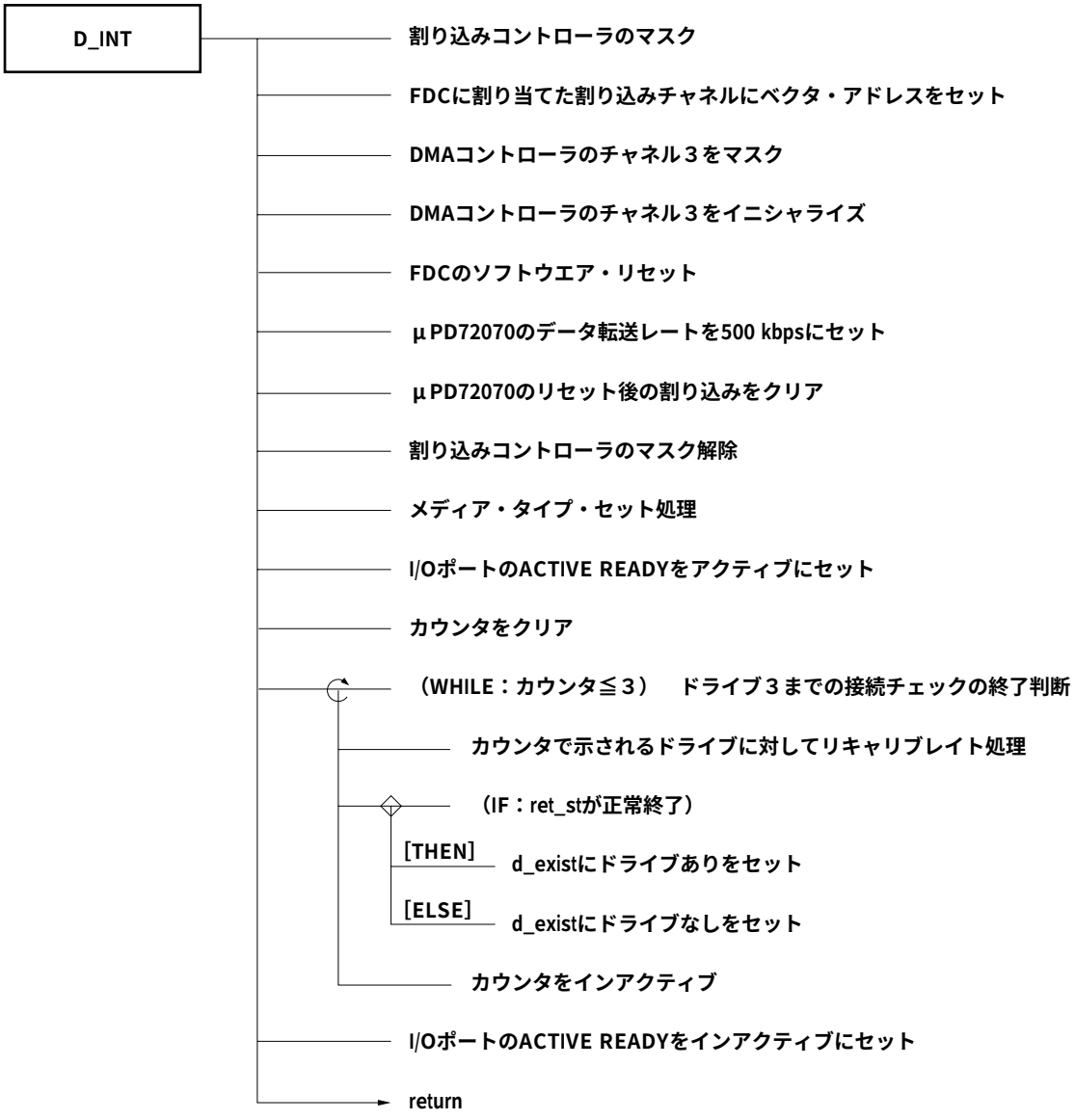
（ｖ）FDCの初期設定（メディア：2HD）を行います。

（vi）リキャル・コマンドを４つのドライブに発行し，ドライブの接続状態をチェックします。チェックした接続状態をd_exist にセットします。



－：Don't care

SPD-4


D_INT
割り込みコントローラのマスク
FDCに割り当てた割り込みチャネルにベクタ・アドレスをセット
DMAコントローラのチャネル３をマスク
DMAコントローラのチャネル３をイニシャライズ
FDCのソフトウエア・リセット
μPD72070のデータ転送レートを500 kbpsにセット
μPD72070のリセット後の割り込みをクリア
割り込みコントローラのマスク解除
メディア・タイプ・セット処理
I/OポートのACTIVE READYをアクティブにセット
カウンタをクリア
（WHILE：カウンタ≦３）　ドライブ３までの接続チェックの終了判断
カウンタで示されるドライブに対してリキャリブレイト処理
（IF：ret_stが正常終了）
[THEN]
d_existにドライブありをセット
[ELSE]
d_existにドライブなしをセット
カウンタをインアクティブ
I/OポートのACTIVE READYをインアクティブにセット
return

（b）メディア・タイプ・セレクト処理

モジュール名	MEDIAT
言　　語	C言語
入　　力	com_code
出　　力	なし

［機　　能］

com_codeに示されたメディア・タイプにμPD72070を設定します。設定には次に示すμPD72070のレジスタとコマンドを使用します。

●レジスタ

コンフィギュレーション・コントロール・レジスタ（CCR）――転送レートの設定をします。

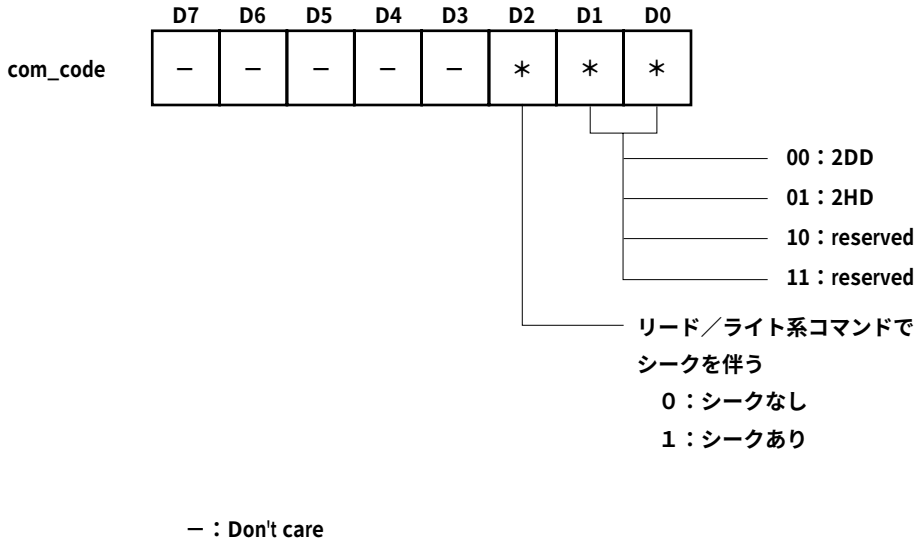
●コマンド

（ｉ）SPECIFY

（ii）CONFIGURE

（iii）PERPENDICULAR MODE

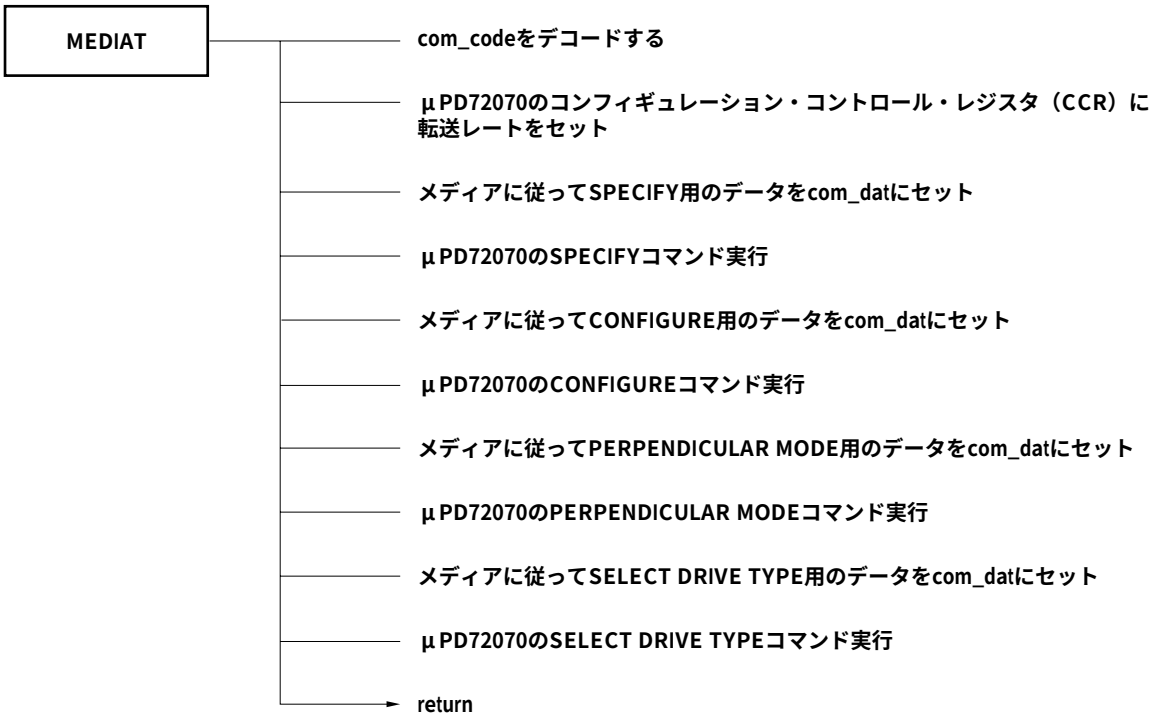
（iv）SELECT DRIVE TYPE



－：Don't care

保守／廃止

SPD-5

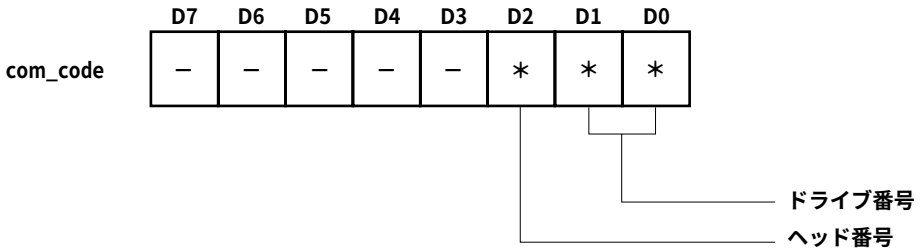

MEDIAT
com_codeをデコードする
μPD72070のコンフィギュレーション・コントロール・レジスタ（CCR）に転送レートをセット
メディアに従ってSPECIFY用のデータをcom_datにセット
μPD72070のSPECIFYコマンド実行
メディアに従ってCONFIGURE用のデータをcom_datにセット
μPD72070のCONFIGUREコマンド実行
メディアに従ってPERPENDICULAR MODE用のデータをcom_datにセット
μPD72070のPERPENDICULAR MODEコマンド実行
メディアに従ってSELECT DRIVE TYPE用のデータをcom_datにセット
μPD72070のSELECT DRIVE TYPEコマンド実行
return

（ｃ）ドライブ・センス処理

モジュール名	SENSE
言　　語	C言語
入　　力	com_code
出　　力	sense_r

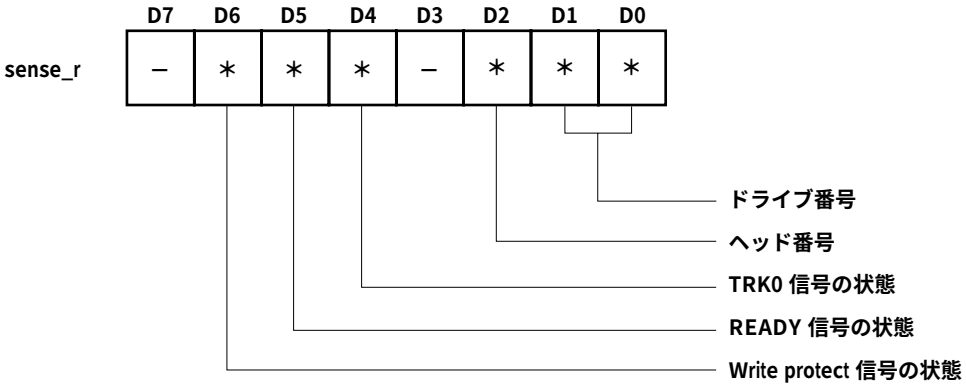
［機　　能］

（ｉ）com_codeで示されるドライブに対して，SENSE DRIVE STATUSコマンドを発行します。



－：Don't care

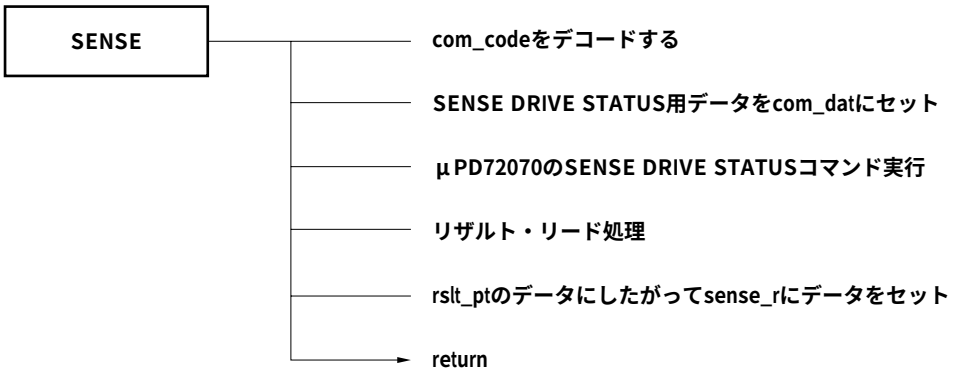
（ⅱ）リザルト・リード処理でコマンドの実行結果を受け取り，sense_r に結果をセットします。



－：Don't care

保守／廃止

SPD-6

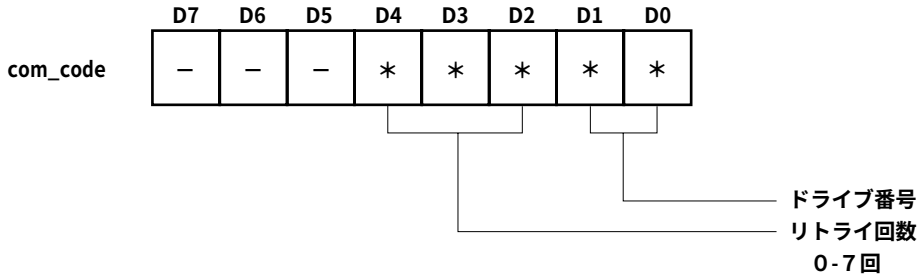

SENSE
com_codeをデコードする
SENSE DRIVE STATUS用データをcom_datにセット
μPD72070のSENSE DRIVE STATUSコマンド実行
リザルト・リード処理
rslt_ptのデータにしたがってsense_rにデータをセット
return

（d）リキャリブレイト処理

モジュール名	RECAL
言　　語	C言語
入　　力	com_code
出　　力	ret_st, ret_prm

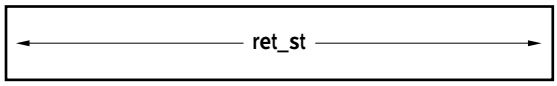
［機　　能］

（ｉ）com_codeで示されるドライブに対して，RECALIBRATE コマンドを発行します。



－：Don't care

（ii）コマンドの実行結果をret_st, ret_prmにセットします。



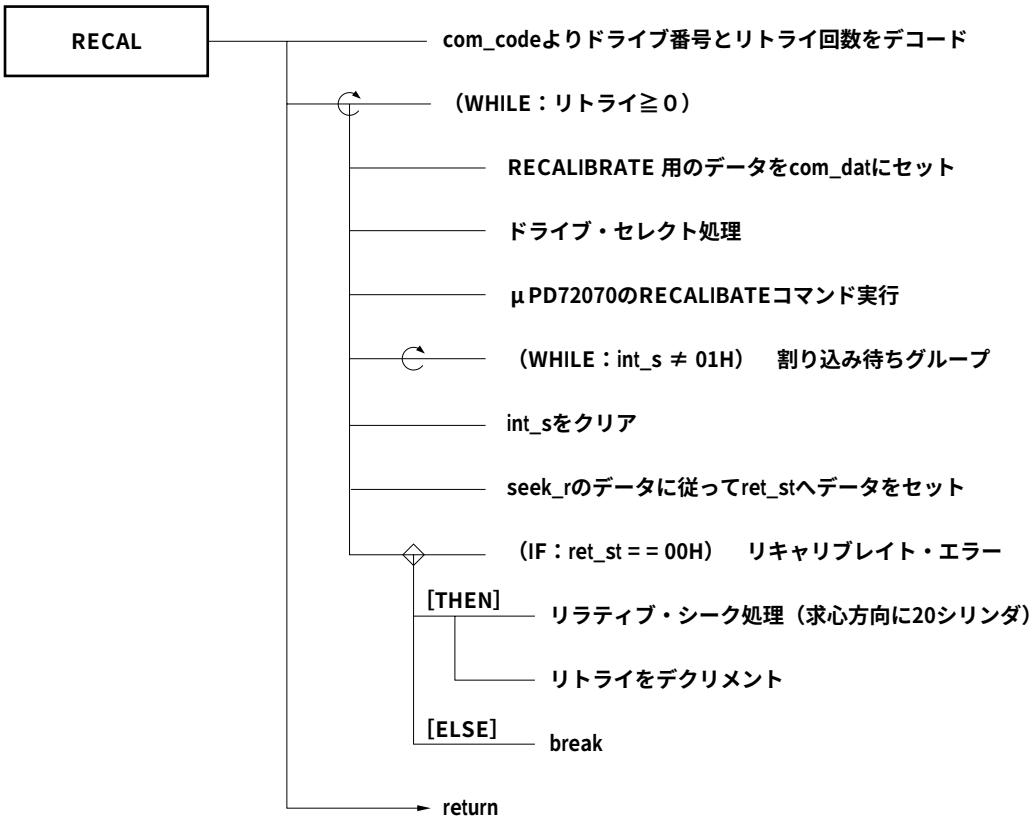
00H：正常終了
01H：Not Ready
0DH：リキャリブレイト・エラー

シーク後のPCNをセット

（iii）実行結果がリキャリブレイト・エラーで，リトライ回数が１以上のときには，リラティブ・シーク処理（求心方向へ20シリンダ）を行い，リキャリブレイトを指定回数だけリトライします。リトライ後，最終結果をret_st, ret_prmにセットして終了します。

保守／廃止

SPD-7


RECAL
com_codeよりドライブ番号とリトライ回数をデコード
（WHILE：リトライ≧０）
RECALIBRATE 用のデータをcom_datにセット
ドライブ・セレクト処理
μPD72070のRECALIBATEコマンド実行
（WHILE：int_s ≠ 01H）　割り込み待ちグループ
int_sをクリア
seek_rのデータに従ってret_stへデータをセット
（IF：ret_st = = 00H）　リキャリブレイト・エラー
[THEN]
リラティブ・シーク処理（求心方向に20シリンダ）
リトライをデクリメント
[ELSE]
break
return

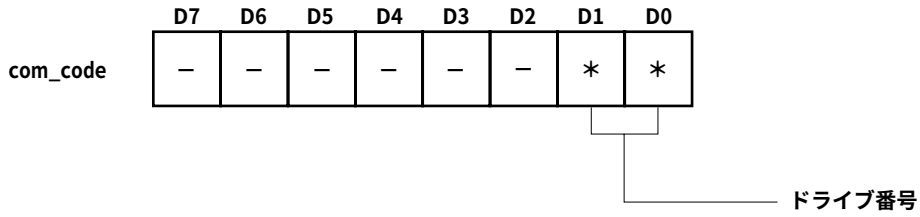
保守／廃止

（ｅ）シーク処理

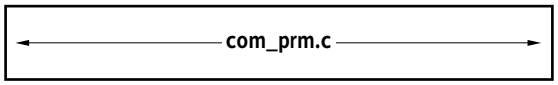
モジュール名	SEEK
言　　語	C言語
入　　力	com_code，com_prm
出　　力	ret_st，ret_prm

［機　　能］

（ｉ）com_code，com_prmの内容に従って，SEEKコマンドを発行します。

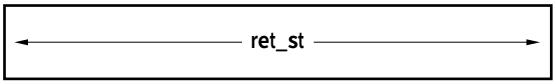


－：Don't care



シーク先のシリンダ番号をセット

（ii）コマンドの実行結果をret_st，ret_prmにセットします。

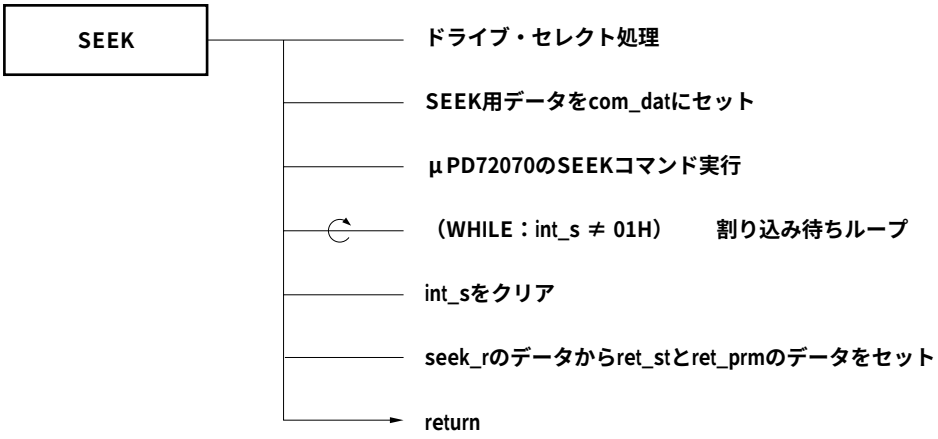


00H：正常終了
01H：Not Ready

シーク後の PCN をセット

保守／廃止

SPD-8

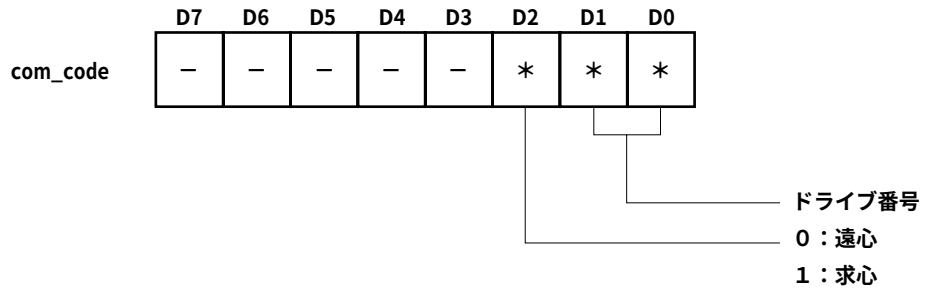

SEEK
ドライブ・セレクト処理
SEEK用データをcom_datにセット
μPD72070のSEEKコマンド実行
（WHILE：int_s ≠ 01H）　割り込み待ちループ
int_sをクリア
seek_rのデータからret_stとret_prmのデータをセット
return

（ｆ）リラティブ・シーク処理

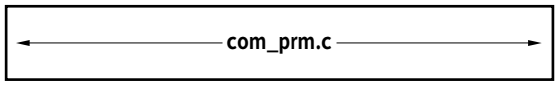
モジュール名	RSEEK
言　　語	C言語
入　　力	com_code, com_prm
出　　力	ret_st, ret_prm

［機　　能］

（ｉ）com_code, com_prmの内容に従って，RELATIVE SEEKコマンドを発行します。

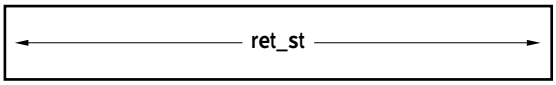


－：Don't care



シーク先のシリンダ番号をセット

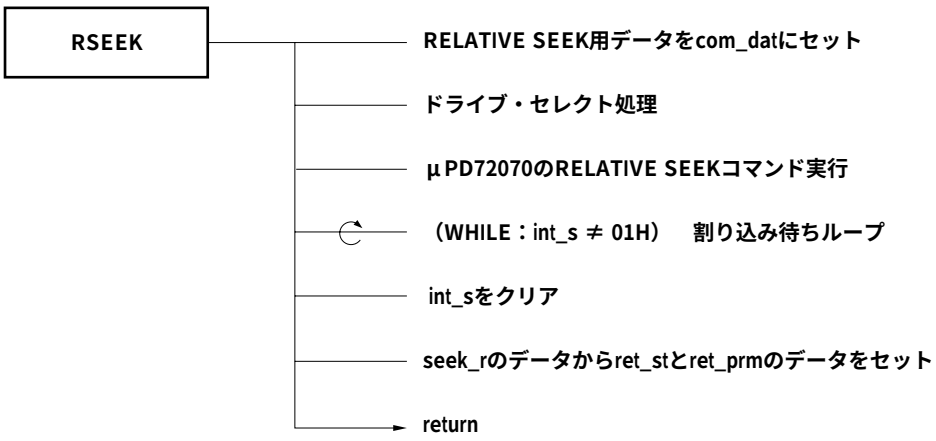
（ⅱ）コマンドの実行結果をret_st, ret_prmにセットします。



00H：正常終了
01H：Not Ready

シーク後の PCN をセット

SPD-9


RSEEK
RELATIVE SEEK用データをcom_datにセット
ドライブ・セレクト処理
μPD72070のRELATIVE SEEKコマンド実行
（WHILE：int_s ≠ 01H）　割り込み待ちループ
int_sをクリア
seek_rのデータからret_stとret_prmのデータをセット
return

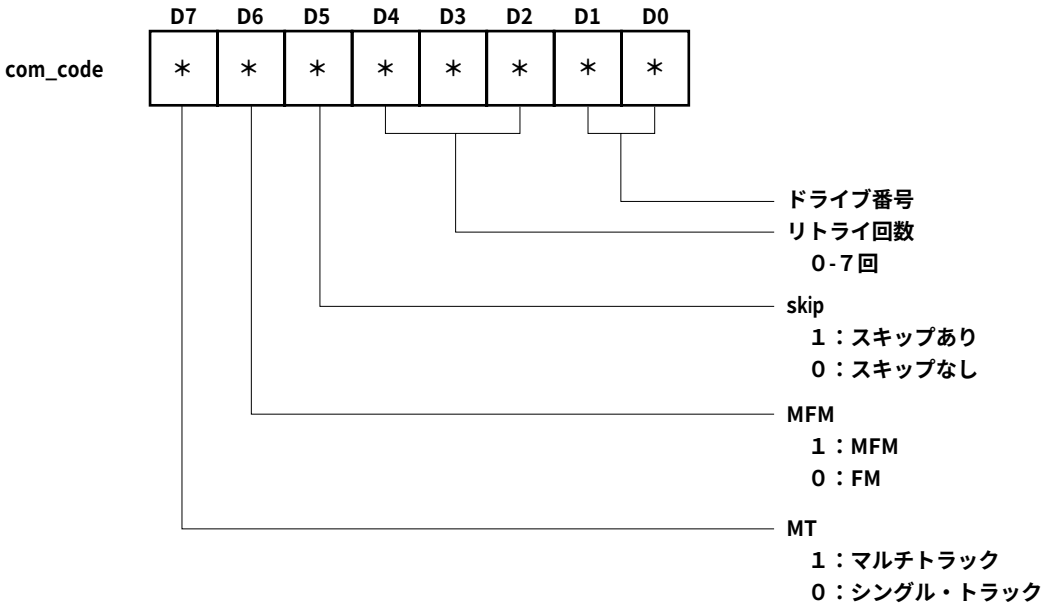
（g）リード・データ処理

モジュール名	RDDAT
言　　語	C言語
入　　力	com_code, com_prm
出　　力	ret_st, ret_prm

［機　　能］

（ｉ）com_prmの内容に従ってDMAコントローラを設定します。

（ii）com_code, com_prmの内容に従ってREAD DATAコマンドを発行し，DAM付きデータをリードします。



（iii）コマンドの実行結果をret_st, ret_prmにセットします。

（iv）ret_stが02H, 05H, 06H, 07H, 08H, 09H, 0AH, 0BHでリトライ回数が１以上のときは，リードを指定回数だけリトライします。

リトライ後，最終結果をret_st, ret_prmにセットし終了します。

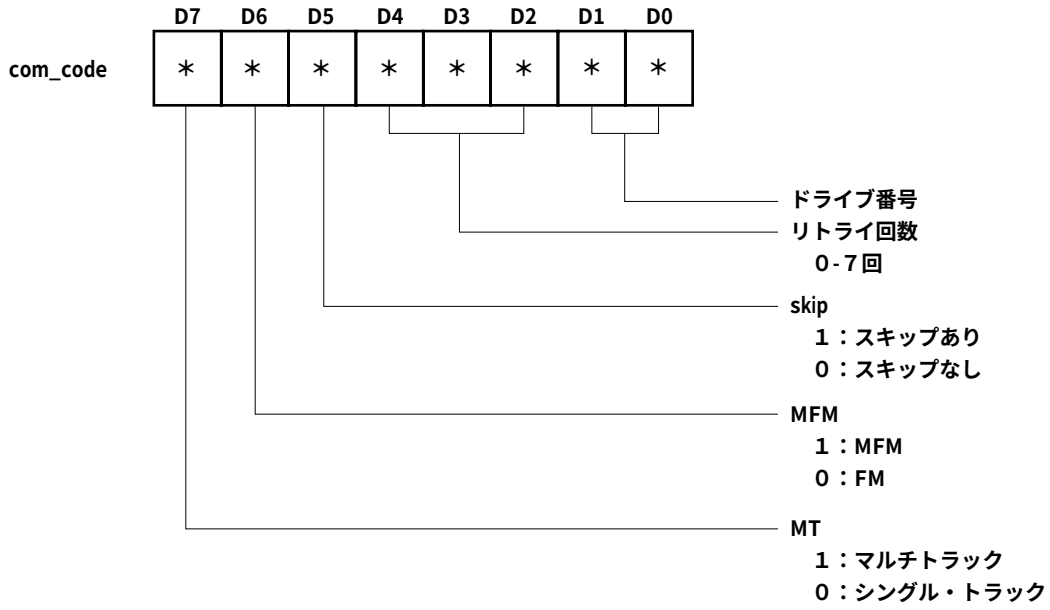
（h）リード・デリーテッド・データ処理

モジュール名	RDDDAT
言　　語	C言語
入　　力	com_code, com_prm
出　　力	ret_st, ret_prm

［機　　能］

（ｉ）com_prmの内容に従ってDMAコントローラを設定します。

（ii）com_code, com_prmの内容に従ってREAD DELETED DATAコマンドを発行し，DDAM付きデータをリードします。



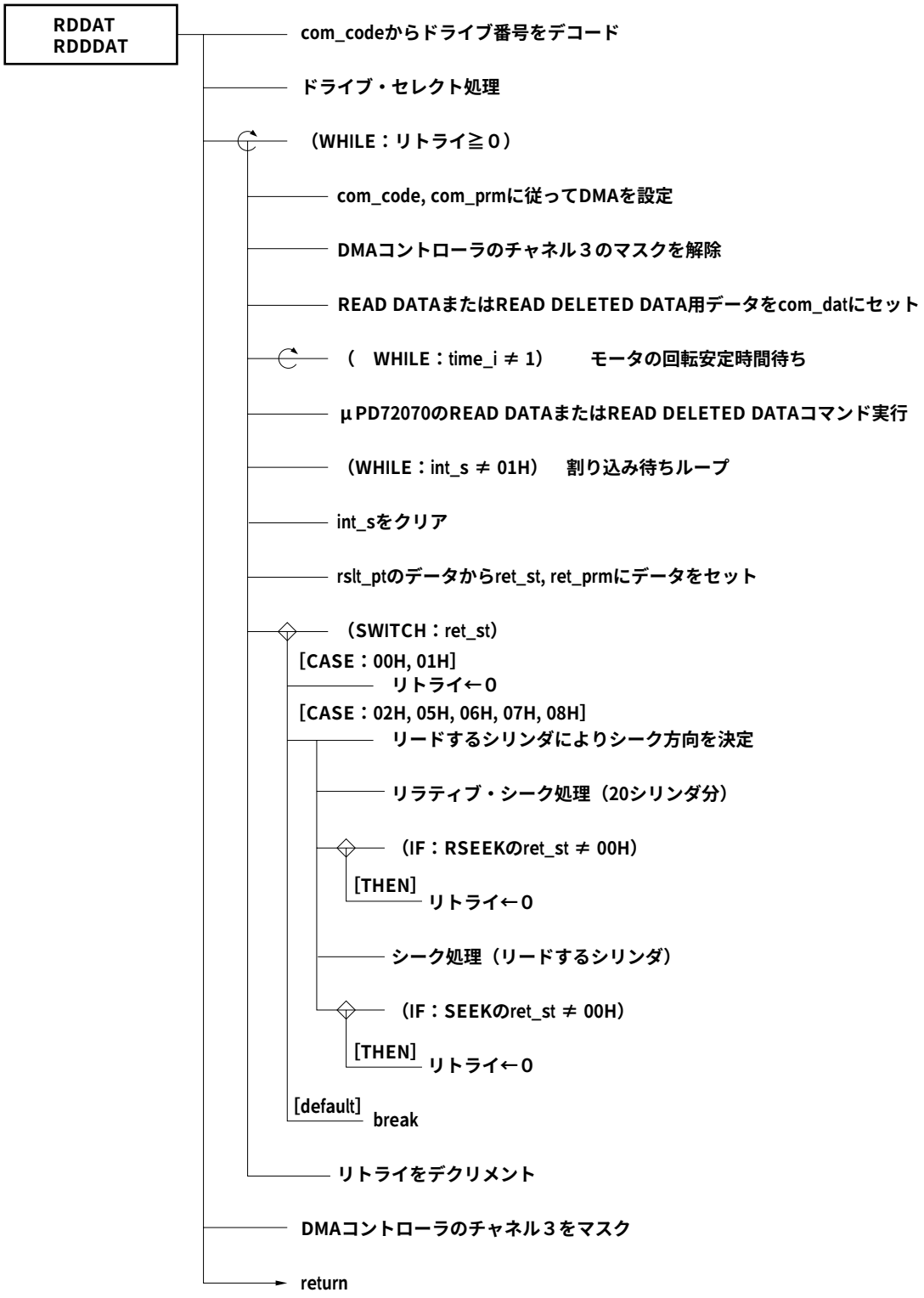
（iii）コマンドの実行結果をret_st, ret_prmにセットします。

（iv）ret_stが02H, 05H, 06H, 07H, 08H, 09H, 0AH, 0BHでリトライ回数が１以上のときは，リードを指定回数だけリトライします。

リトライ後，最終結果をret_st, ret_prmにセットし終了します。

保守／廃止

SPD-10

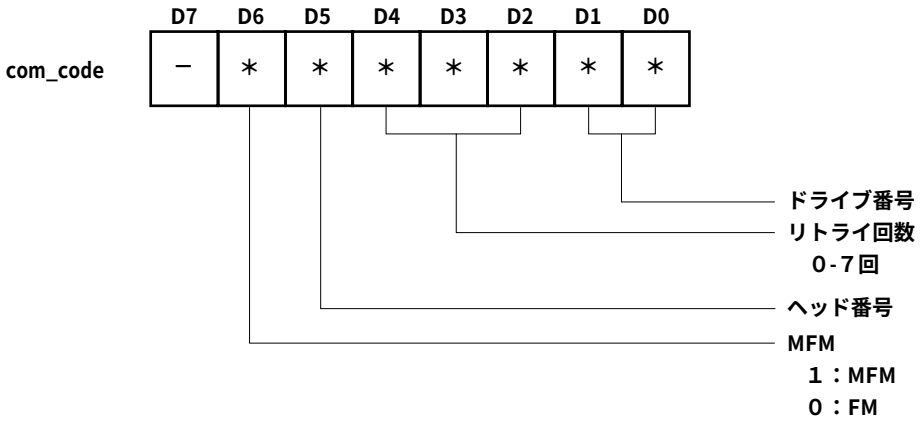

RDDAT
RDDDAT
com_codeからドライブ番号をデコード
ドライブ・セレクト処理
（WHILE：リトライ≧０）
com_code, com_prmに従ってDMAを設定
DMAコントローラのチャネル３のマスクを解除
READ DATAまたはREAD DELETED DATA用データをcom_datにセット
（　WHILE：time_i ≠ 1）　モータの回転安定時間待ち
μPD72070のREAD DATAまたはREAD DELETED DATAコマンド実行
（WHILE：int_s ≠ 01H）　割り込み待ちループ
int_sをクリア
rslt_ptのデータからret_st, ret_prmにデータをセット
（SWITCH：ret_st）
[CASE：00H, 01H]
リトライ←０
[CASE：02H, 05H, 06H, 07H, 08H]
リードするシリンダによりシーク方向を決定
リラティブ・シーク処理（20シリンダ分）
（IF：RSEEKのret_st ≠ 00H）
[THEN]
リトライ←０
シーク処理（リードするシリンダ）
（IF：SEEKのret_st ≠ 00H）
[THEN]
リトライ←０
[default]
break
リトライをデクリメント
DMAコントローラのチャネル３をマスク
return

（ｉ）リードID処理

モジュール名	RDID
言　　語	C言語
入　　力	com_code
出　　力	ret_st, ret_prm

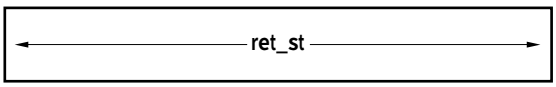
［機　　能］

（ｉ）com_codeの内容に従ってREAD IDコマンドを発行し，任意のIDをリードします。



－：Don't care

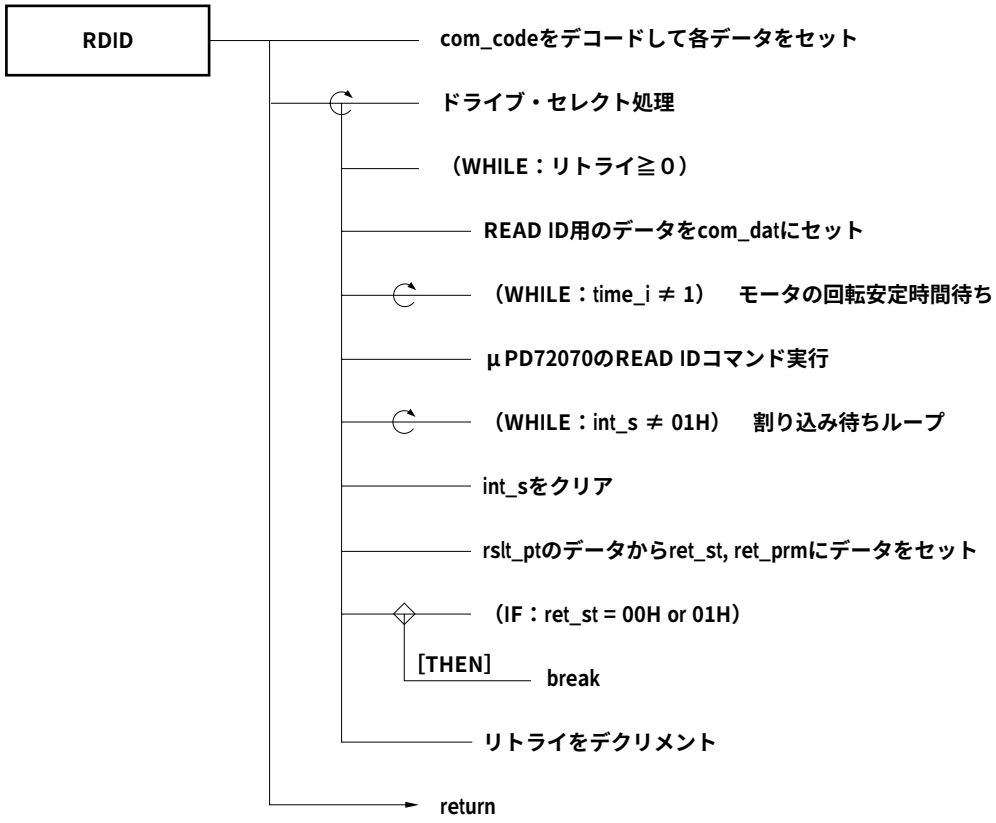
（ⅱ）コマンドの実行結果をret_st, ret_prmにセットします。



00H：正常終了
01H：Not Ready
02H：IDAM 非検出
05H：H, R, N 非検出

（ⅲ）ret_stが02H，05Hで，リトライ回数が１以上のときは，IDリードを指定回数だけリトライします。
リトライ後，最終結果をret_st, ret_prmにセットし終了します。

SPD-11


RDID
com_codeをデコードして各データをセット
ドライブ・セレクト処理
（WHILE：リトライ≧０）
READ ID用のデータをcom_datにセット
（WHILE：time_i ≠ 1）　モータの回転安定時間待ち
μPD72070のREAD IDコマンド実行
（WHILE：int_s ≠ 01H）　割り込み待ちループ
int_sをクリア
rslt_ptのデータからret_st, ret_prmにデータをセット
（IF：ret_st = 00H or 01H）
[THEN]
break
リトライをデクリメント
return

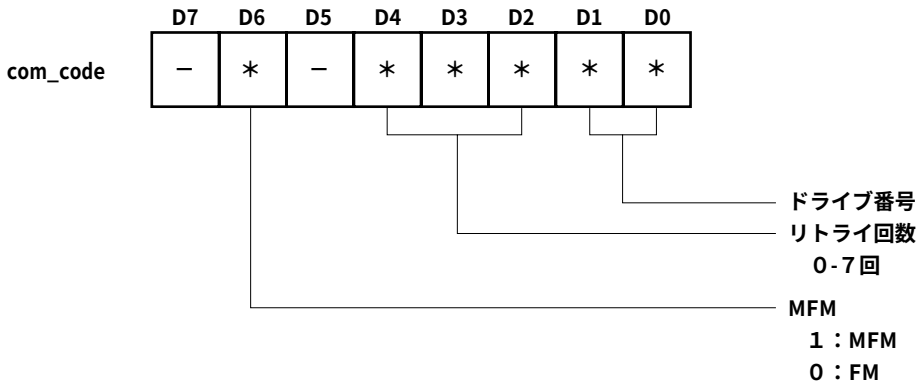
（ｊ）リード・ダイアグノスティック処理

モジュール名	RDDAG
言　　語	C言語
入　　力	com_code, com_prm
出　　力	ret_st, ret_prm

［機　　能］

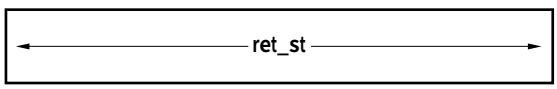
（ｉ）com_prmの内容に従ってDMAコントローラを設定します。

（ii）com_code, com_prmの内容に従ってREAD A TRACKコマンドを発行し，１トラック分のデータをリードします。



－：Don't care

（iii）コマンドの実行結果をret_st, ret_prmにセットします。



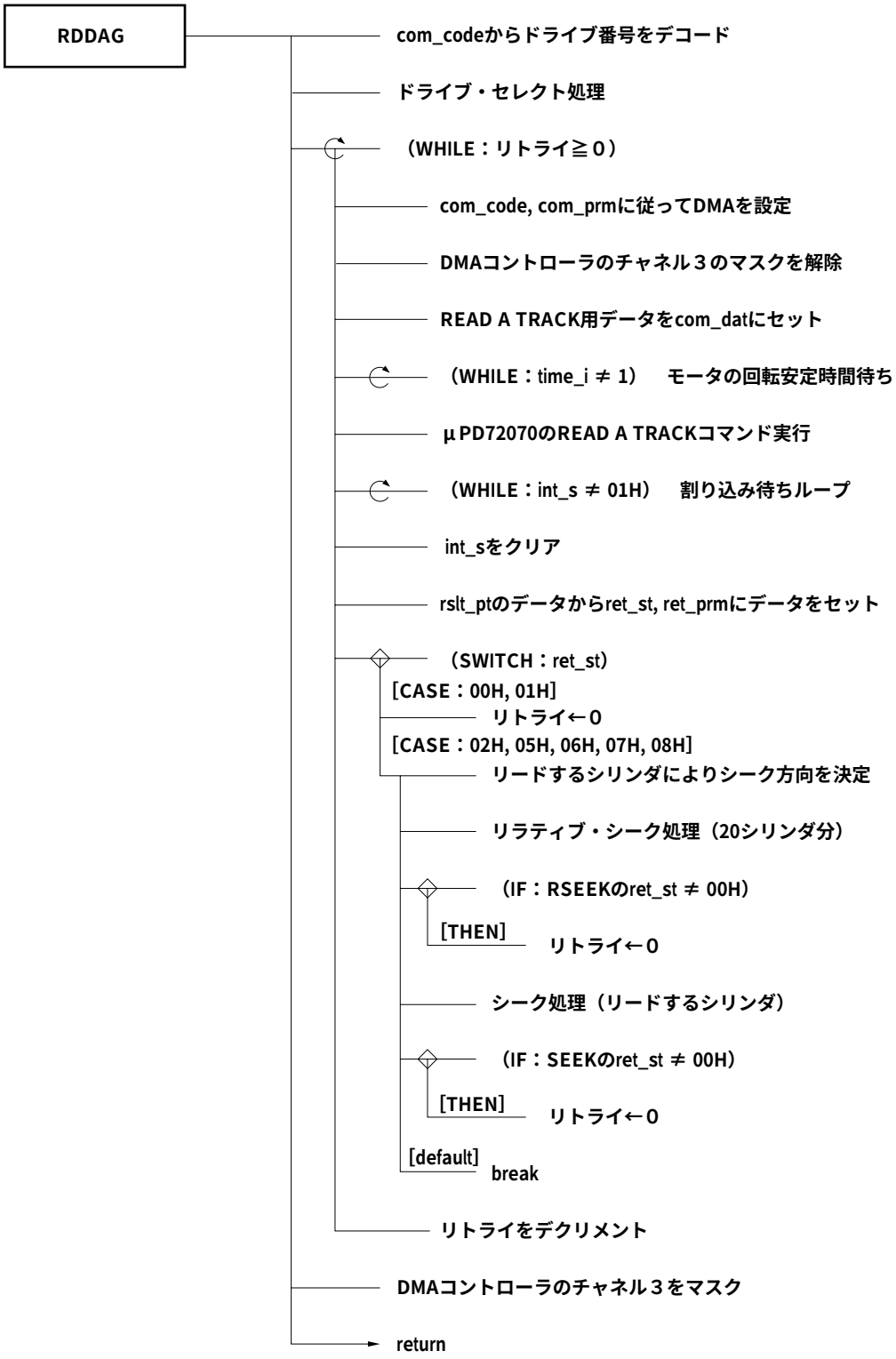
00H：正常終了
01H：Not Ready
02H：IDAM非検出
08H：DAM非検出
0AH：オーバラン・エラー
0BH：最終セクタを越えて
　　アクセスしようとした
0EH：エラーがあるデータを
　　リードした

（iv）ret_stが02H, 08H, 0AH, 0BH, 0EHで，リトライ回数が１以上のときは，リードを指定回数だけリトライします。

リトライ後，最終結果をret_st, ret_prmにセットし終了します。

保守／廃止

SPD-12


RDDAG
com_codeからドライブ番号をデコード
ドライブ・セレクト処理
（WHILE：リトライ≧０）
com_code, com_prmに従ってDMAを設定
DMAコントローラのチャネル３のマスクを解除
READ A TRACK用データをcom_datにセット
（WHILE：time_i ≠ 1）　モータの回転安定時間待ち
μPD72070のREAD A TRACKコマンド実行
（WHILE：int_s ≠ 01H）　割り込み待ちループ
int_sをクリア
rslt_ptのデータからret_st, ret_prmにデータをセット
（SWITCH：ret_st）
[CASE：00H, 01H]
リトライ←０
[CASE：02H, 05H, 06H, 07H, 08H]
リードするシリンダによりシーク方向を決定
リラティブ・シーク処理（20シリンダ分）
（IF：RSEEKのret_st ≠ 00H）
[THEN]
リトライ←０
シーク処理（リードするシリンダ）
（IF：SEEKのret_st ≠ 00H）
[THEN]
リトライ←０
[default]
break
リトライをデクリメント
DMAコントローラのチャネル３をマスク
return

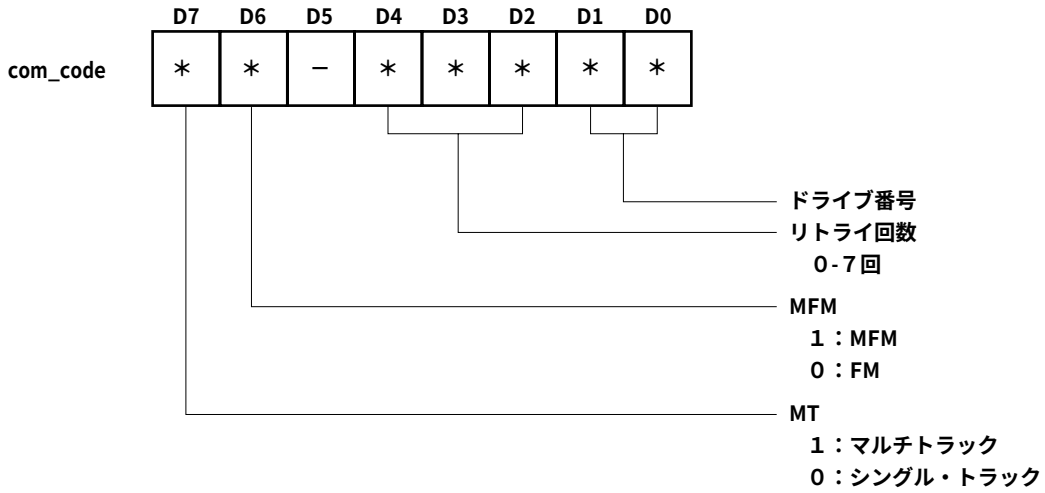
（k）ライト・データ処理

モジュール名	WRDAT
言　　語	C言語
入　　力	com_code, com_prm
出　　力	ret_st, ret_prm

［機　　能］

（ｉ）com_prmの内容に従って，DMAコントローラを設定します。

（ii）com_code, com_prmの内容に従ってWRITE DATAコマンドを発行しDAM付きデータをライトします。



－：Don't care

（iii）コマンドの実行結果をret_st, ret_prmにセットします。

（iv） ret_stが02H, 05H, 06H, 0AH, 0BHで，リトライ回数が１以上のときは，ライトを指定回数だけリトライします。

リトライ後，最終結果をret_st, ret_prmにセットし終了します。

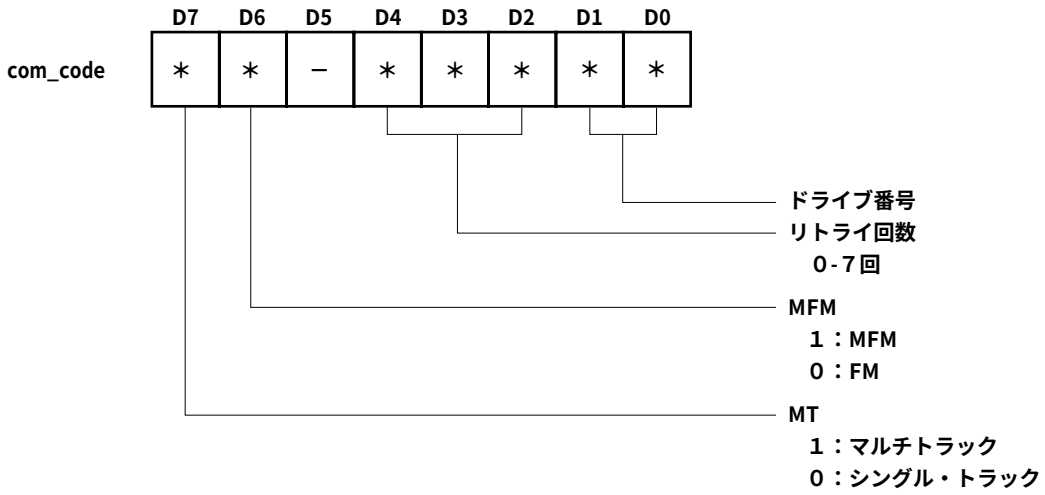
（ｌ）ライト・デリーテッド・データ処理

モジュール名	WRDDAT
言　　語	C言語
入　　力	com_code，com_prm
出　　力	ret_st，ret_prm

［機　　能］

（ｉ）com_prmの内容に従って，DMAコントローラを設定します。

（ii）com_code，com_prmの内容に従ってWRITE DELETED DATAコマンドを発行し，DDAM付きデータをライトします。



－：Don't care

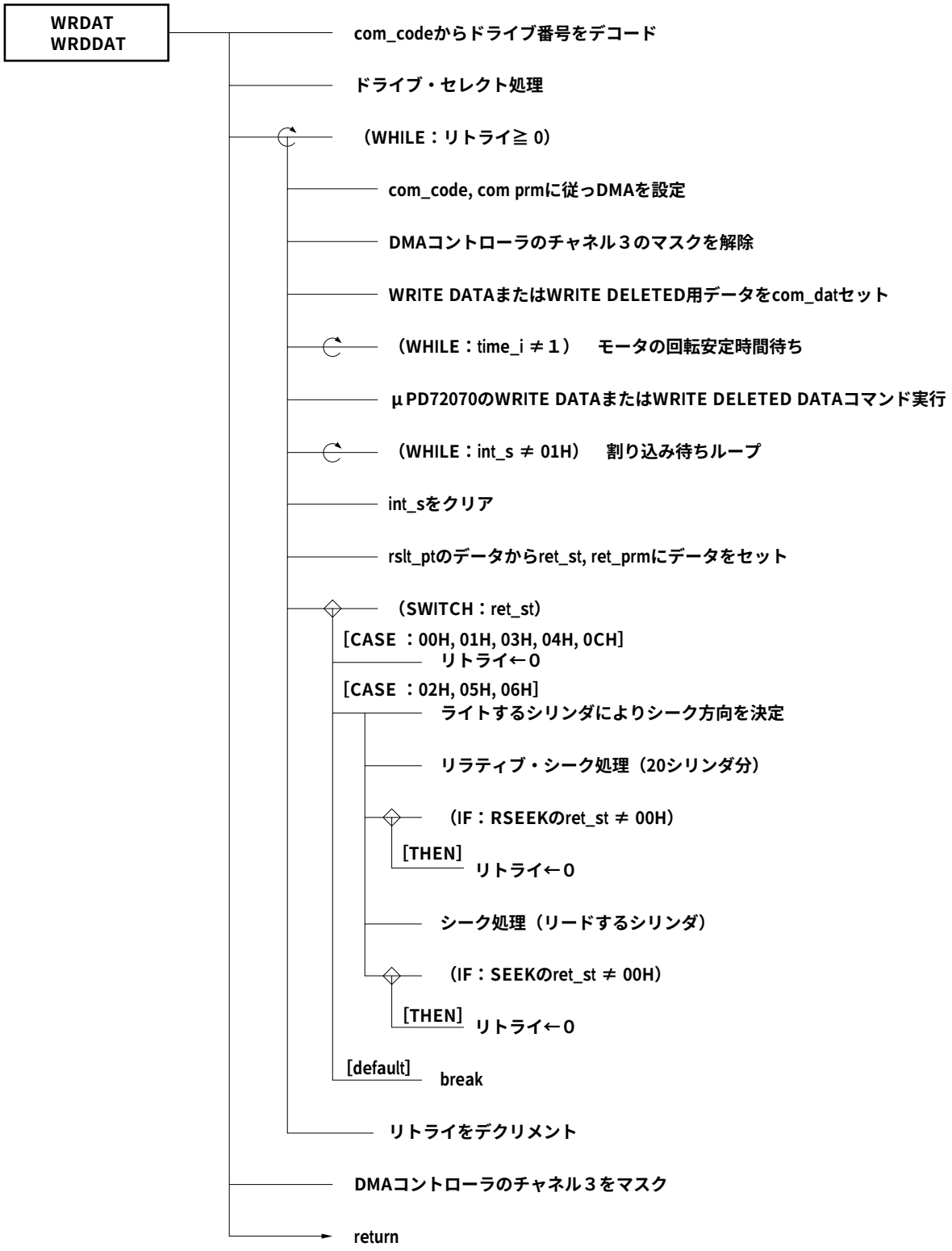
（iii）コマンドの実行結果をret_st，ret_prmにセットします。

（iv）ret_stが02H，05H，06H，0AH，0BHで，リトライ回数が１以上のときは，ライトを指定回数だけリトライします。

リトライ後，最終結果をret_st，ret_prmにセットし終了します。

保守／廃止

SPD-13


WRDAT
WRDDAT
com_codeからドライブ番号をデコード
ドライブ・セレクト処理
（WHILE：リトライ≧ 0）
com_code, com prmに従っDMAを設定
DMAコントローラのチャネル３のマスクを解除
WRITE DATAまたはWRITE DELETED用データをcom_datセット
（WHILE：time_i ≠１）　モータの回転安定時間待ち
μPD72070のWRITE DATAまたはWRITE DELETED DATAコマンド実行
（WHILE：int_s ≠ 01H）　割り込み待ちループ
int_sをクリア
rslt_ptのデータからret_st, ret_prmにデータをセット
（SWITCH：ret_st）
[CASE ：00H, 01H, 03H, 04H, 0CH]
リトライ←0
[CASE ：02H, 05H, 06H]
ライトするシリンダによりシーク方向を決定
リラティブ・シーク処理（20シリンダ分）
（IF：RSEEKのret_st ≠ 00H）
[THEN]
リトライ←0
シーク処理（リードするシリンダ）
（IF：SEEKのret_st ≠ 00H）
[THEN]
リトライ←0
[default]
break
リトライをデクリメント
DMAコントローラのチャネル３をマスク
return

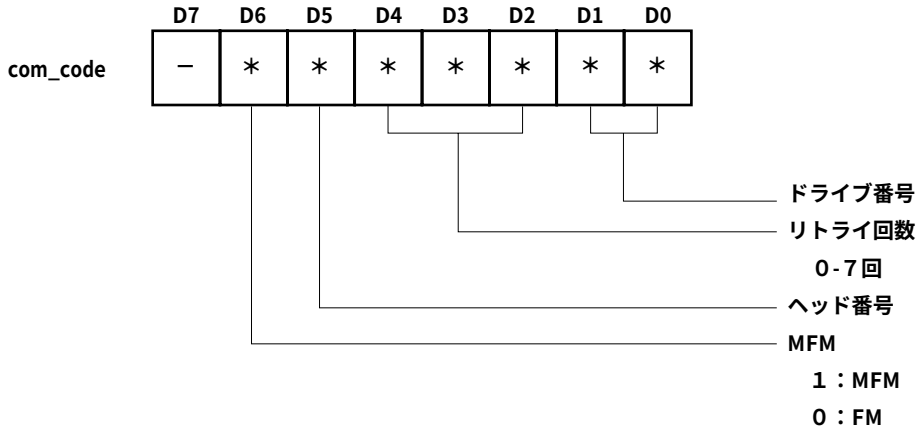
（m）トラック・フォーマット処理

モジュール名	TRFORMAT
言　　語	C言語
入　　力	com_code, com_prm
出　　力	ret_st, ret_prm

［機　　能］

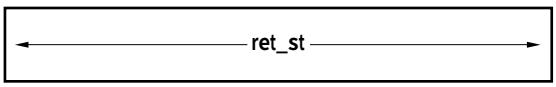
（ｉ）com_prmの内容に従ってDMAコントローラを設定します。

（ii）com_code, com_prmの内容に従ってFORMAT A TRACKコマンドを発行し，１トラック分フォーマットします。



－：Don't care

（iii）コマンドの実行結果をret_st, ret_prmにセットします。

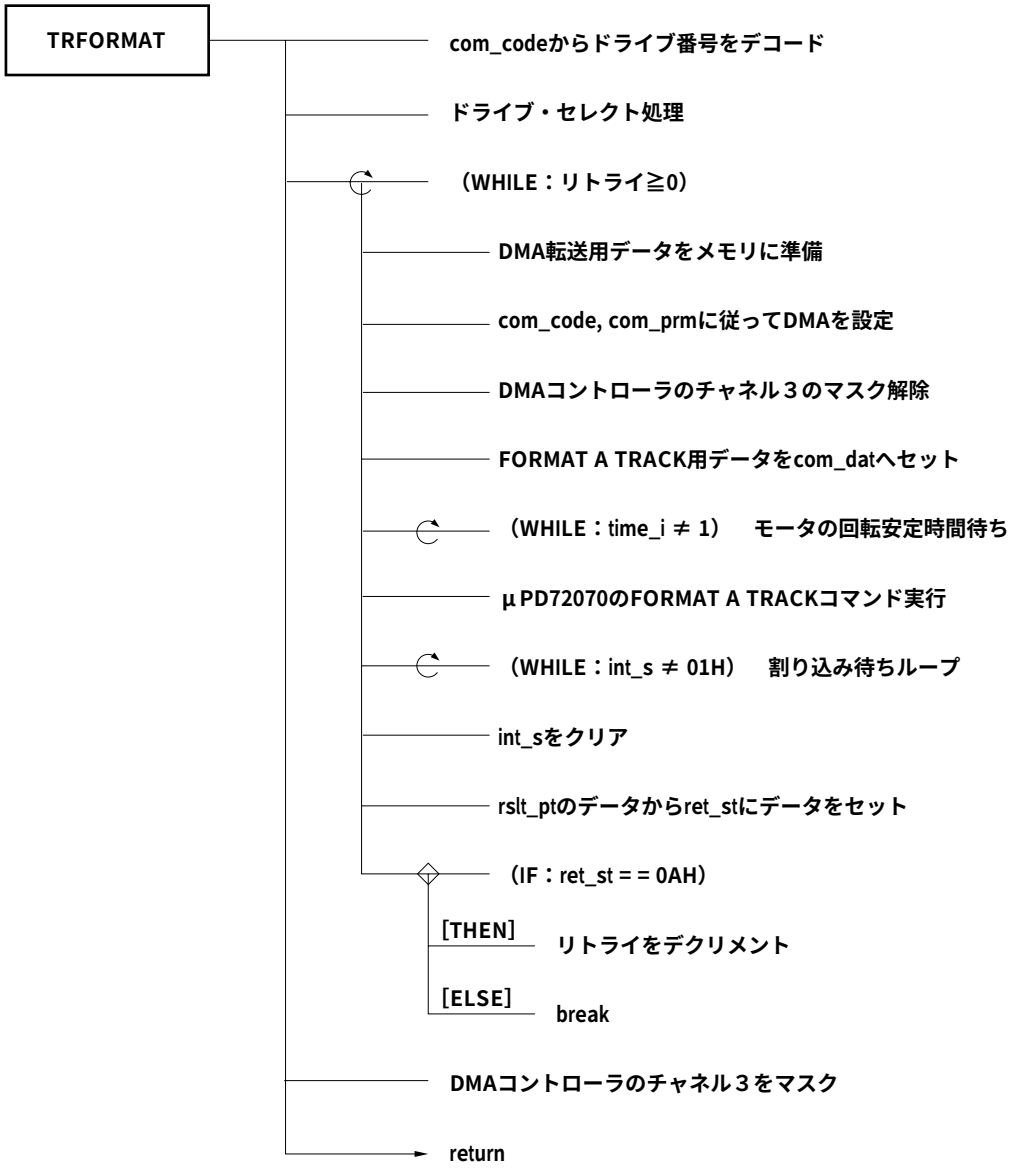


00H：正常終了
01H：Not Ready
0AH：オーバラン・エラー
0CH：ライト・プロテクト

（iv）ret_stが0AHで，リトライ回数が１以上のときは，フォーマットを指定回数だけリトライします。
リトライ後，最終結果をret_st, ret_prmにセットし終了します。

保守／廃止

SPD-14

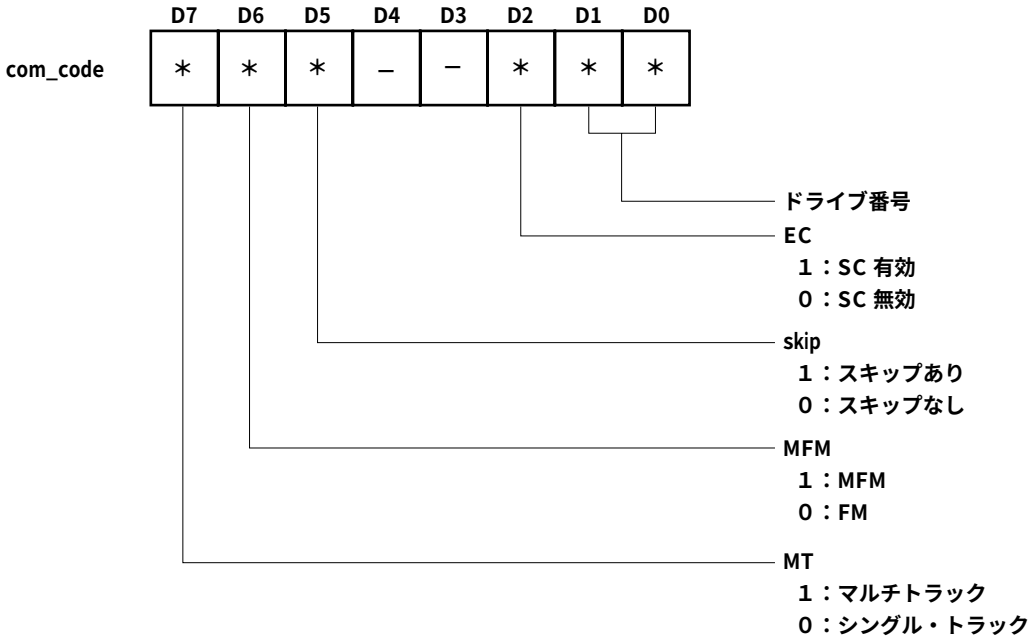

TRFORMAT
com_codeからドライブ番号をデコード
ドライブ・セレクト処理
（WHILE：リトライ≧0）
DMA転送用データをメモリに準備
com_code, com_prmに従ってDMAを設定
DMAコントローラのチャネル３のマスク解除
FORMAT A TRACK用データをcom_datへセット
（WHILE：time_i ≠ 1）　モータの回転安定時間待ち
μPD72070のFORMAT A TRACKコマンド実行
（WHILE：int_s ≠ 01H）　割り込み待ちループ
int_sをクリア
rslt_ptのデータからret_stにデータをセット
（IF：ret_st = = 0AH）
[THEN]
リトライをデクリメント
[ELSE]
break
DMAコントローラのチャネル３をマスク
return

（ｎ）ベリファイ処理

モジュール名	VERIFY
言　　語	C言語
入　　力	com_code, com_prm
出　　力	ret_st, ret_prm

［機　　能］

（ｉ）com_code, com_prmの内容に従ってVERIFYコマンドを発行し，ベリファイを実行します。



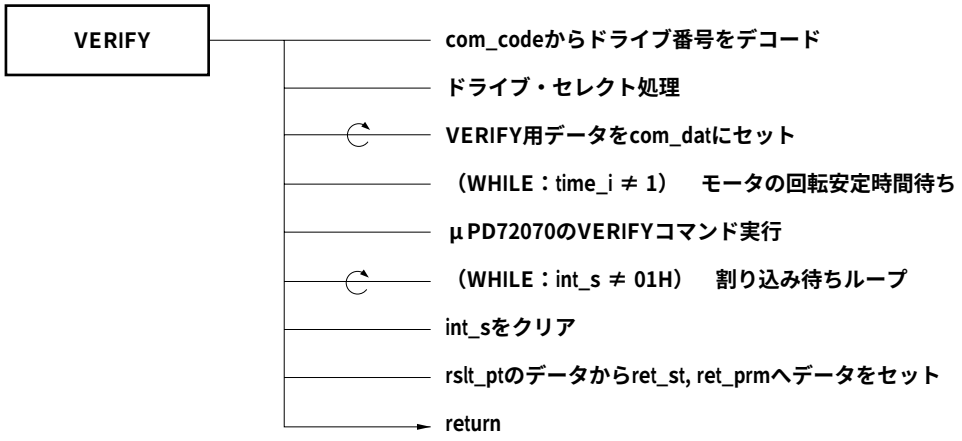
－：Don't care

なお，ECが０のときはDTL /SCパラメータはFFHとします。

（ⅱ）コマンドの実行結果をret_st, ret_prmにセットし終了します。

保守／廃止

SPD-15


VERIFY
com_codeからドライブ番号をデコード
ドライブ・セレクト処理
VERIFY用データをcom_datにセット
（WHILE：time_i ≠ 1）　モータの回転安定時間待ち
μPD72070のVERIFYコマンド実行
（WHILE：int_s ≠ 01H）　割り込み待ちループ
int_sをクリア
rslt_ptのデータからret_st, ret_prmへデータをセット
return

（o）ドライブ・セレクト処理

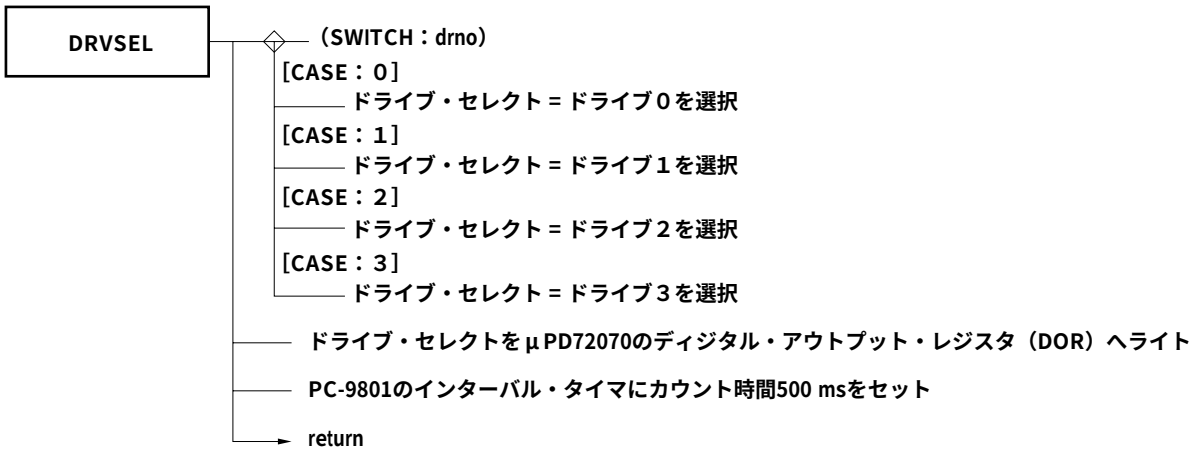
モジュール名	DRVSEL
言　　語	C言語
入　　力	drno
出　　力	な　し

［機　　能］

drnoで示されるドライブ番号に対応した値をμPD72070のディジタル・アウトプット・レジスタにライトし，μPD72070のDS信号とME信号をアクティブにします。

また，FDDのモータの回転安定時間待ちに使用するPC-9801のインターバル・タイマをセットします。

SPD-16



（p）タイマ割り込み処理

モジュール名	TIME
言　　語	C言語
入　　力	なし
出　　力	time_i

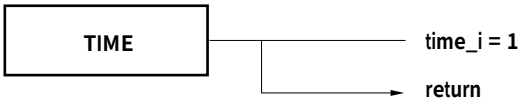
［機　　能］

DRVSELモジュールでセットされたPC-9801のインターバル・タイマの割り込み処理を行います。

インターバル・タイマの割り込みがあったことを知らせる変数time_iに１をセットします。

注意　LIOモジュールのリード／ライト系モジュールは，time_iに１がセットされた（モータONから500 msが経過した）ことを確認してから，コマンドを実行します。time_iはモータOFF されたときにクリア（０をセット）してください。

SPD-17



（２）LIOの変数

ここでは，LIOで使用する変数について説明します。

表３－６に変数と機能を示します。

表３－６　LIOの変数

変 数 名	機　　能
com_code コマンド・コード （１バイト）	LIOの各モジュールへ実行するドライブ番号や，その他のパラメータを与える変数です。 内容については各モジュールの説明に示します。
com_prm コマンド・パラメータ （構造体）	実行セクタや，メディア情報のパラメータを各LIOモジュールへ引き渡す変数です。 構造体の内容については**表３－７　構造体com_prmの内容**を参照してください。
ret_st リターン・ステータス （１バイト）	コマンドの実行結果を示します。 値が示す内容については**表３－８　リターン・ステータス・コード**を参照してください。
ret_prm リターン・パラメータ （構造体）	コマンド実行終了セクタまたは，実行終了の次のセクタを示す変数です。 シリンダ番号（C），ヘッド番号（H），セクタ番号（R），データ長（N）（各１バイト）が構造体メンバです。
sense_r センス・リザルト （１バイト）	ドライブ・センス処理の結果がセットされる変数です。 内容については**３.２.２　（１）（ｃ）ドライブ・センス処理**の説明を参照してください。
drno ドライブ・ナンバ （１バイト）	コマンドを実行するドライブ番号を示す変数です。
time_i タイマ・インタラプト （１バイト）	PC-9801のインターバル・タイマの割り込みがあったことを知らせる変数です。

保守／廃止

表３－７　構造体com_prmの内容

構造体メンバ名	機　　能
dtsz データ・サイズ （２バイト）	転送バイト数を示す変数です。 int_henとの共用体として宣言されます。
int_hen データ・サイズ変換 （構造体）	転送バイト数を示す変数です。 dtszとの共用体として宣言され，dtszの値の下位，上位それぞれ８ビットずつにわける構造体メンバ（kai, jyoui）で構成されています。
dt_off データ・オフセット （２バイト）	転送先または，転送元のオフセット・アドレスを示す変数です。
dt_seg データ・セグメント （２バイト）	転送先または，転送元のセグメント・アドレスを示す変数です。
c シリンダ・ナンバ （１バイト）	コマンドを実行するシリンダ番号を示す変数です。 シーク処理ではシーク先のシリンダ番号（NCN）がセットされ，リラティブ・シーク処理ではシークするシリンダ数（RCN）がセットされます。
h ヘッド・ナンバ （１バイト）	コマンドを実行するヘッド番号（サイド）を示す変数です。
r セクタ・ナンバ （１バイト）	コマンドを実行するセクタ番号を示す変数です。
n フォーマット・バイト （１バイト）	コマンドを実行する１セクタあたりのデータ長を示す変数です。
dtl データ・レングス （１バイト）	１セクタあたりの処理データ長を示す変数です。n ≠ 00Hのときは意味をもちません。ただしトラック・フォーマット処理，ベリファイ処理では，トラックあたりのセクタ数（SC）を示します。
gsl ギャップ・スキップ・レングス （１バイト）	GAP3の読み飛ばしバイト数を示す変数です。ただしトラック・フォーマット処理では，GAP3の書き込みバイト数（GPL）を示します。
eot エンド・オブ・トラック （１バイト）	トラック上でアクセスする最終セクタ番号を示す変数です。
d データ （１バイト）	トラック・フォーマット処理でデータ領域にライトするデータ・パターンを示す変数です。

保守／廃止

表３－８　リターン・ステータス・コード

リターン・ステータス・コード	内　　容
00H	正常終了
01H	Not Ready
02H	IDAM　非検出
03H	指定したシリンダがない　　ND = 1, BC = 1, NC = 0
04H	指定したシリンダがない　　ND = 1, BC = 0, NC = 1
05H	指定したセクタがない（H, R, Nの不一致）　　ND = 1, BC = 0, NC = 0
06H	ID部のCRCエラー
07H	DATA部のCRCエラー
08H	DAM, DDAM　非検出
09H	DDAM, またはDAM付きデータをリードした （リード・データ処理，リード・ダイアグノスティック処理はDDAM，リード・デリーテッド・データ処理ではDAM）
0AH	オーバラン・エラー
0BH	最終セクタを越えてアクセスしようとした
0CH	ライト・プロテクト
0DH	リキャリブレイト・エラー
0EH	エラーがあるデータをリードした

保守／廃止

３.２.３　デバイス・ドライバ

この章では，１M/2 MバイトFDDをアクセスするためのデバイス・ドライバについて説明します。

（１）デバイス・ヘッダ

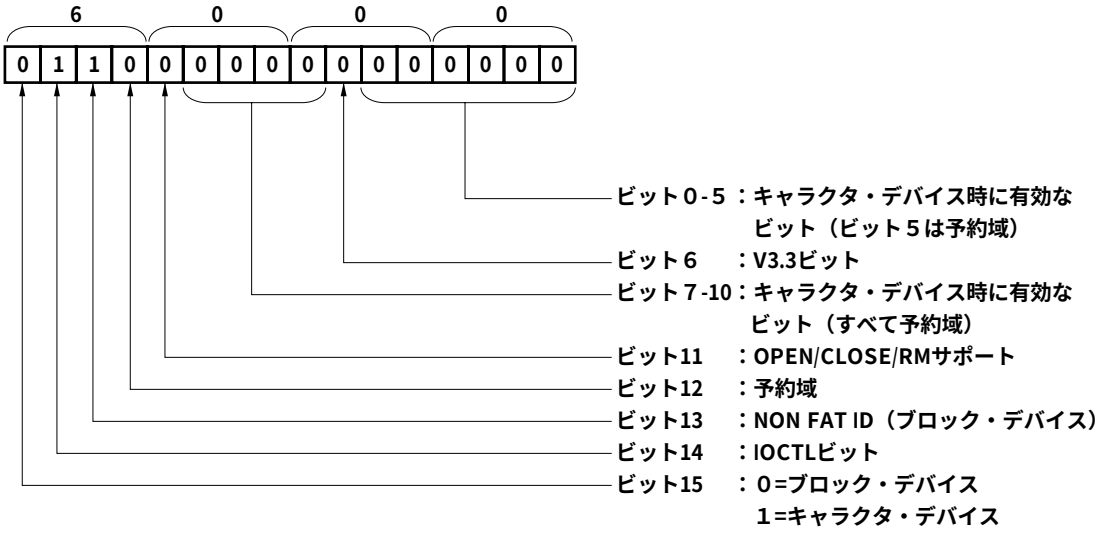
ここではデバイス・ドライバのデバイス・ヘッダについて説明します。

デバイス・ヘッダの構成を表３－９に，デバイス属性の説明を図３－３にそれぞれ示します。

表３－９　デバイス・ヘッダ

値	意　　味
－１	リンク情報
6000H	デバイス属性
STRATEGY	ストラテジ・エントリ・ポイント
INTERRUPT	割り込みエントリ・ポイント
4	ユニット数

図３－３　デバイス属性について



備考　予約域には０をセットします。

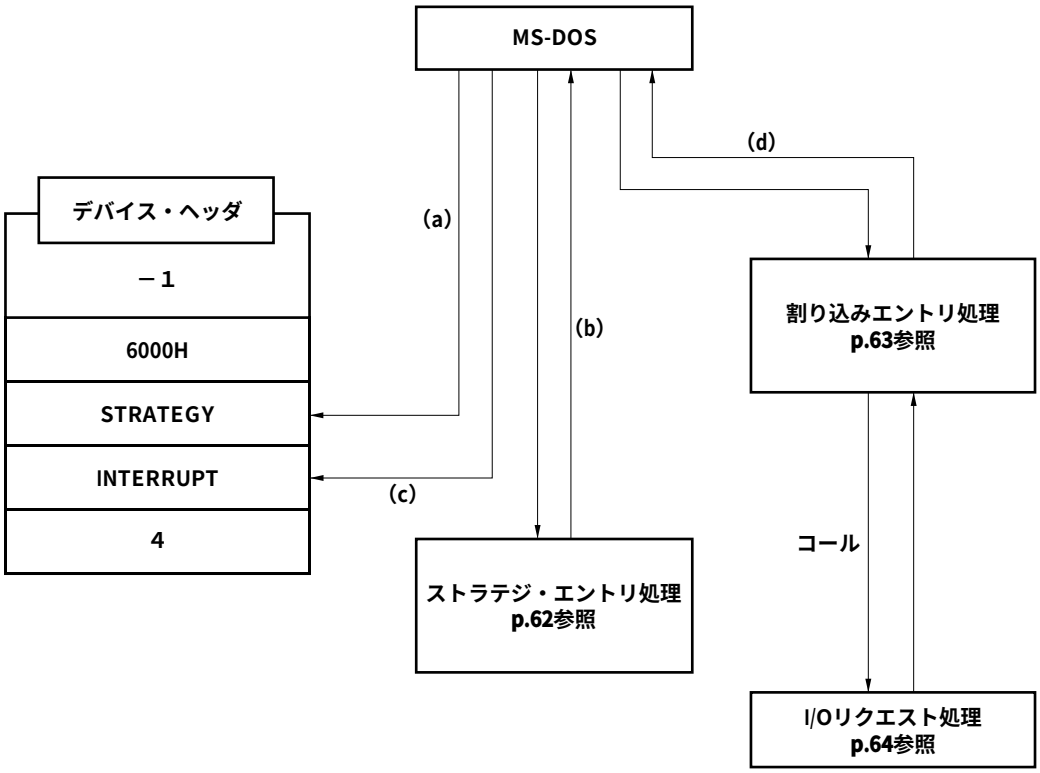
保守／廃止

（２）MS-DOSとのインタフェース

ここでは，MS-DOSとのインタフェース部を説明します。

図３－４にMS-DOSとμPD72070デバイス・ドライバとのインタフェースを示し，MS-DOSがμPD72070デバイス・ドライバにアクセスする動作を説明します。

図３－４　MS-DOSとのインタフェース



（ａ）MS-DOSは，デバイス・ヘッダを参照し，ストラテジ・エントリ処理（STRATEGY）のポインタを確認します。

（ｂ）MS-DOSは，ストラテジ・エントリ処理（STRATEGY）をコールします。

（ｃ）MS-DOSは，デバイス・ヘッダを参照し，割り込みエントリ処理（INTERRUPT）のポインタを確認します。

（ｄ）MS-DOSは，割り込みエントリ処理（INTERRUPT）をコールします。

保守／廃止

（３）I/Oリクエスト処理

ここでは，I/Oリクエスト処理ブロックの説明をします。

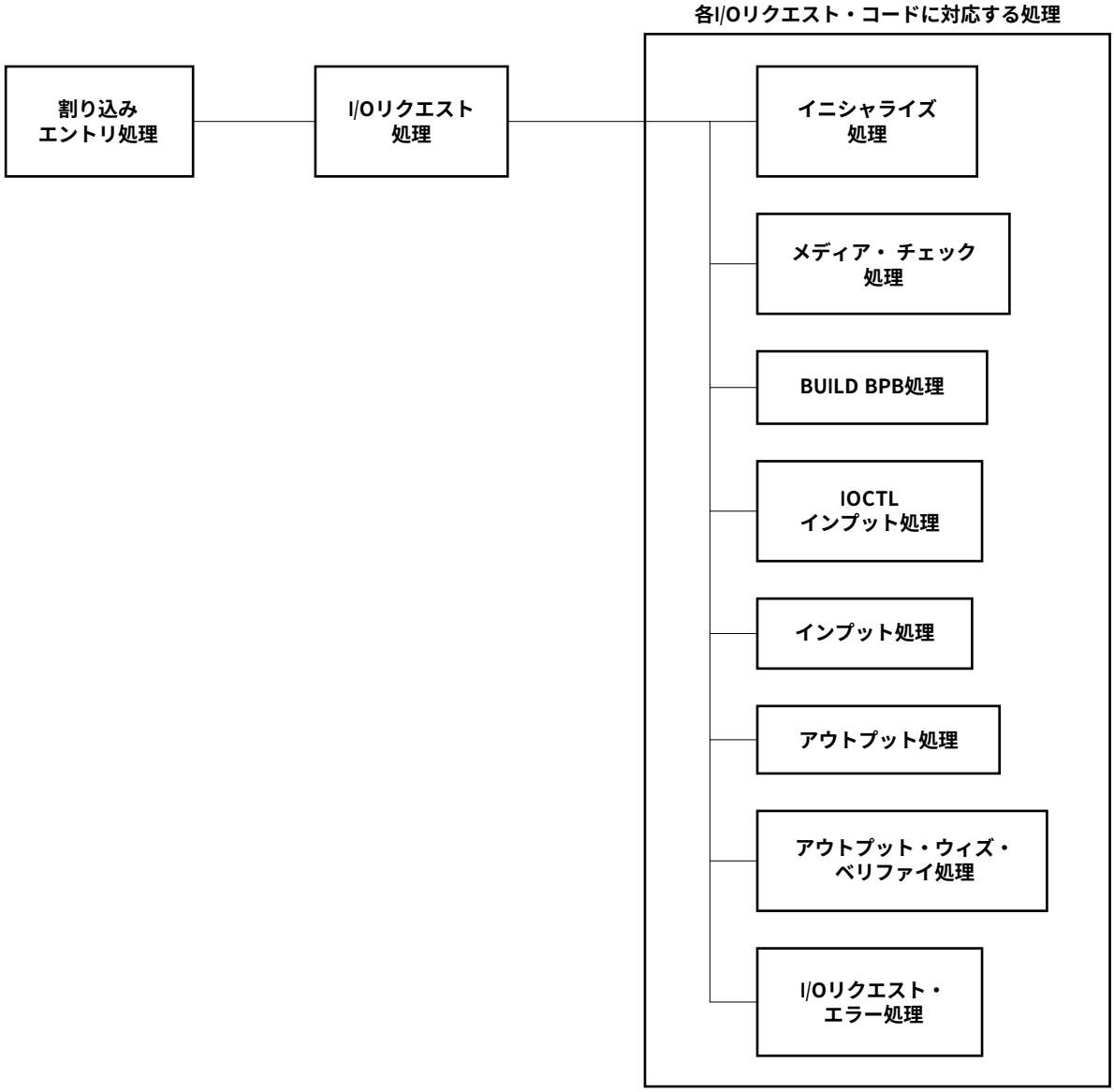
I/OリクエストMS-DOSからの要求（I/Oリクエスト・コード）に従ってデータのリード／ライトなどを実行します。

図３－５にI/Oリクエスト処理のプログラム構成を示します。

I/Oリクエスト処理へのアクセスは割り込みエントリ処理をコールすることで実行します。I/Oリクエスト処理は，I/Oリクエスト・コードによってコールする処理を選択します。

また，表３－10にI/Oリクエスト・コードとI/Oリクエスト処理のモジュールを示します。

図３－５　I/Oリクエスト処理ブロックの構成



保守／廃止

表３－10　I/Oリクエスト・コードとI/Oリクエスト処理

I/Oリクエスト・コード	I/Oリクエスト処理名	参照ページ
00H	イニシャライズ処理	p.65
01H	メディア・チェック処理	p.66
02H	BUILD BPB処理	p.68
03H	IOCTLインプット処理	p.70
04H	インプット処理	p.70
08H	アウトプット処理	p.72
09H	アウトプット・ウィズ・ベリファイ処理	p.74
その他	I/Oリクエスト・エラー処理	p.76

（4）デバイス・ドライバ・モジュールの説明

ここではデバイス・ドライバのモジュールの説明をします。表３－11に示す内容に従って各モジュールを説明し，SPDチャートを示します。

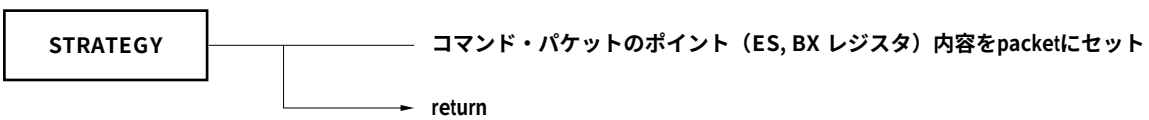
表３－11　各モジュール説明内容

モジュール名	モジュール名を示します。
言　　語	関数がどの言語で作成されているかを示します。
入　　力	入力時の変数および，レジスタを示します。
出　　力	出力時の変数および，レジスタを示します。
［機　　能］ 関数の処理内容を示します。	

（a）ストラテジ・エントリ処理

モジュール名	STRATEGY
言　　語	アセンブリ言語
入　　力	BXレジスタ：コマンド・パケットのオフセット・アドレス ESレジスタ：コマンド・パケットのセグメント・アドレス
出　　力	packet
［機　　能］ MS-DOSからBX，ESレジスタを使用して渡してくるコマンド・パケットのポインタをpacketに退避します。	

SPD-18



（b）割り込みエントリ処理

モジュール名	INTERRUPT
言　　語	アセンブリ言語
入　　力	なし
出　　力	pro_seg, pro_off

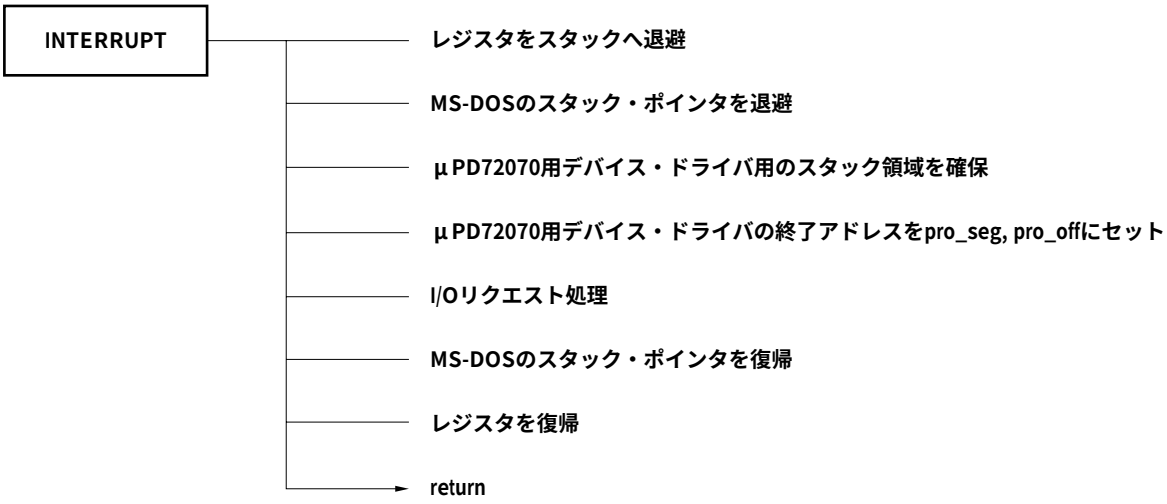
［機　　能］

（ｉ）デバイス・ドライバのスタック領域を確保します。

（ii）デバイス・ドライバの終了アドレスをpro_seg, pro_offにセットします。

（iii）I/Oリクエスト処理をコールします。

SPD-19

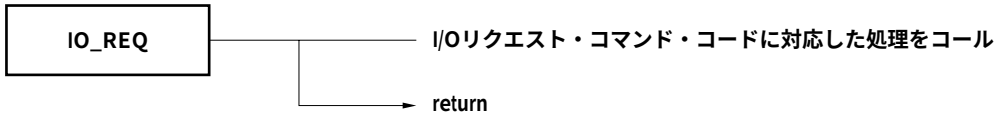


保守／廃止

（ｃ）I/Oリクエスト処理

モジュール名	IO_REQ
言　　語	C言語
入　　力	packet
出　　力	なし
［機　　能］ I/Oリクエスト・コマンド・コードに対応した処理をコールします。	

SPD-20



（d）イニシャライズ処理

モジュール名	INITIAL
言　　語	C言語
入　　力	pro_seg, pro_off
出　　力	packet, unit_ex

［機　　能］

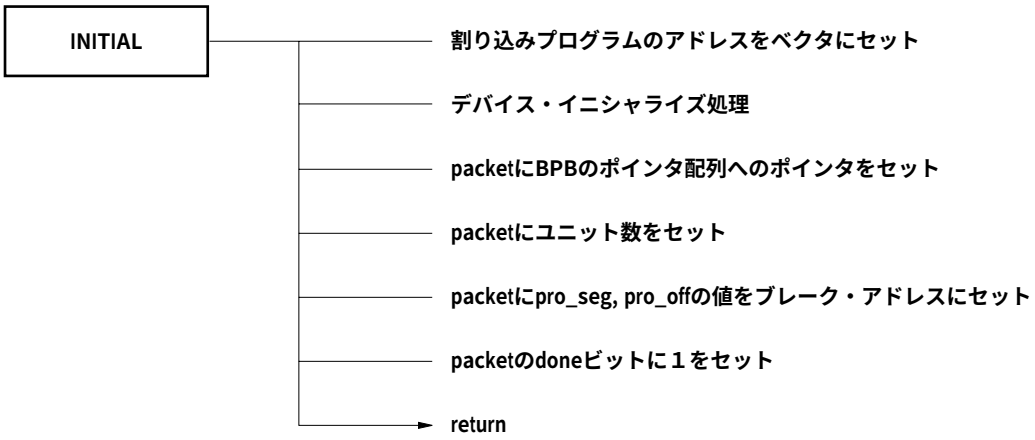
（ｉ）デバイス・イニシャライズ処理をコールして，μPD72070を初期化します。

（ⅱ）コマンド・パケットにBPBのポインタ配列へのポインタをセットします。

（ⅲ）コマンド・パケットにユニット数をセットします。

（ⅳ）コマンド・パケットにpro_seg, pro_offの値をブレーク・アドレスにセットします。

SPD-21



（ｅ）メディア・チェック処理

モジュール名	MD_CHEK
言　　語	C言語
入　　力	packet
出　　力	packet

［機　　能］

（ｉ）リクエスト・ヘッダで示されるドライブのディスクが１度抜かれたかどうか，チェックします。
（ディスク・チェンジ信号のチェック）

（ⅱ）ディスクが抜かれていればコマンド・パケットの返す値に－１（ディスクが交換された）をセットします。

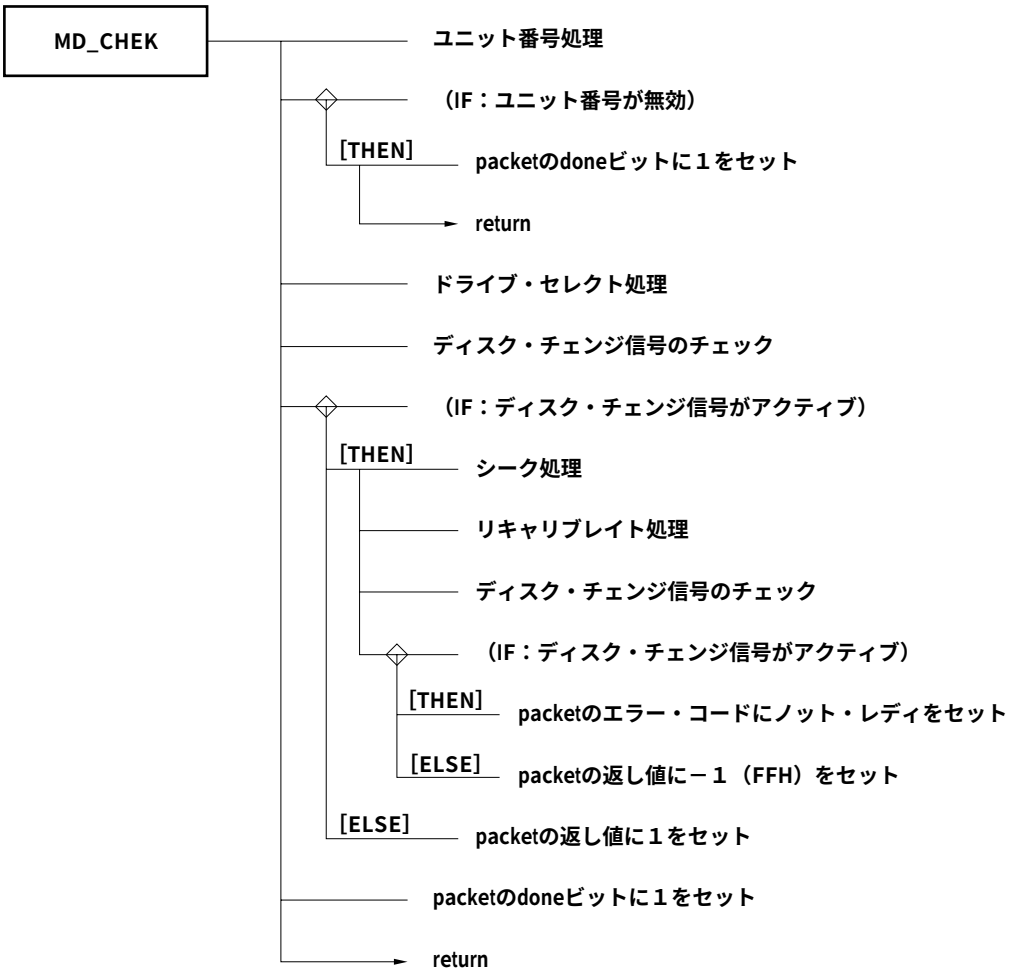
（ⅲ）さらに現在ドライブにディスクがセットされているかチェックし，セットされていなければコマンド・パケットのエラー・コードにノット・レディをセットします。

（ⅳ）ディスクが抜かれていなければコマンド・パケットの返す値に１（ディスクは交換されていない）をセットします。

（ⅴ）コマンド・パケットにdoneビットをセットします。

保守／廃止

SPD-22


MD_CHEK
ユニット番号処理
（IF：ユニット番号が無効）
[THEN]
packetのdoneビットに１をセット
return
ドライブ・セレクト処理
ディスク・チェンジ信号のチェック
（IF：ディスク・チェンジ信号がアクティブ）
[THEN]
シーク処理
リキャリブレイト処理
ディスク・チェンジ信号のチェック
（IF：ディスク・チェンジ信号がアクティブ）
[THEN]
packetのエラー・コードにノット・レディをセット
[ELSE]
packetの返し値に－１（FFH）をセット
[ELSE]
packetの返し値に１をセット
packetのdoneビットに１をセット
return

（ｆ）BUILD BPB処理

モジュール名	BUILD_BPB
言　　語	C言語
入　　力	packet, pro_seg
出　　力	packet

［機　　能］

（ｉ）リクエスト・ヘッダで示されるドライブの予約セクタ（31バイト分）をリードします。

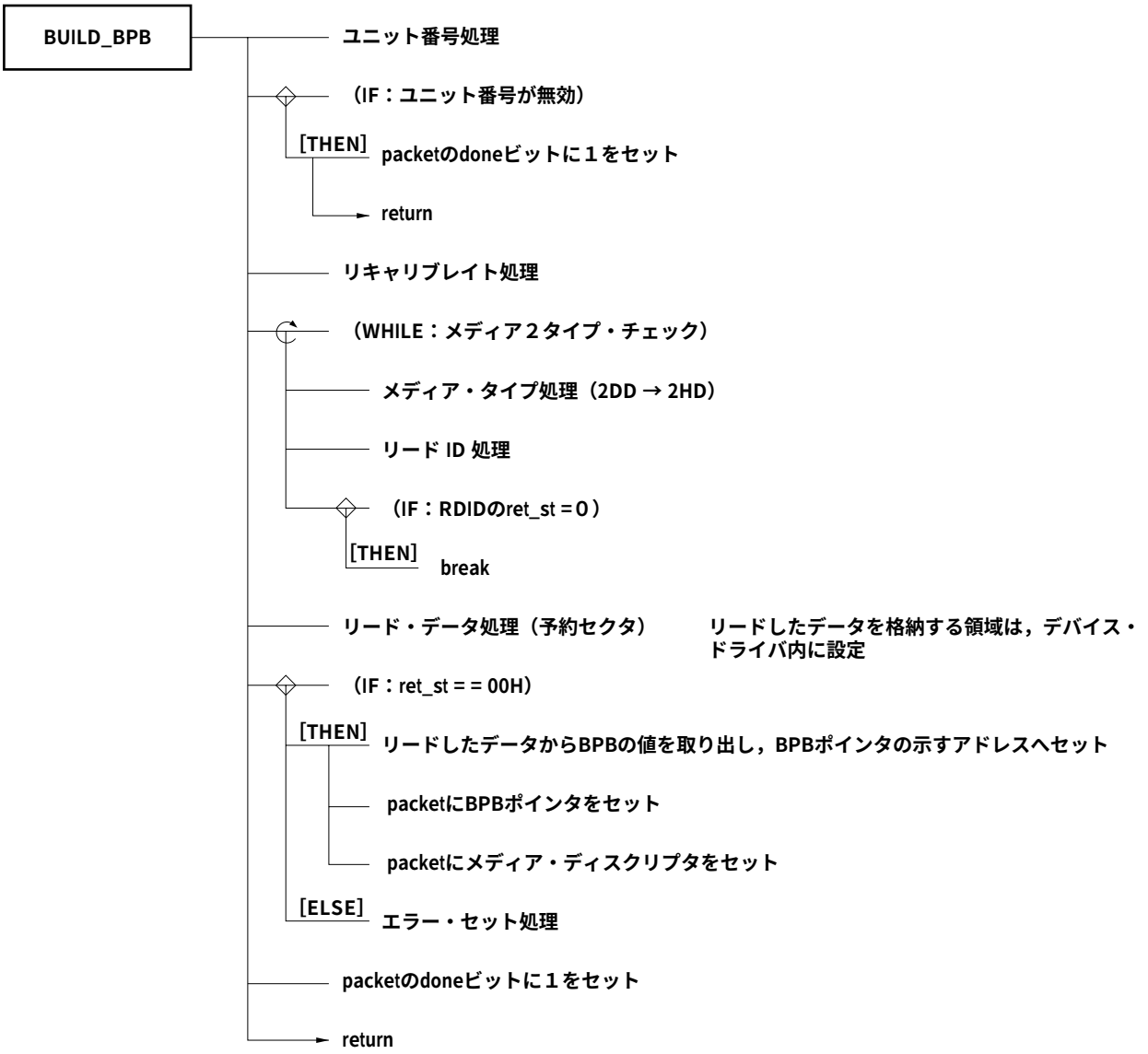
（ⅱ）リクエスト・ヘッダで示されるドライブに対応するBPBポインタの内容に，リードした予約セクタの内容のBPBの値をセットします。

（ⅲ）コマンド・パケットにメディア・ディスクリプタをセットします。

（ⅳ）コマンド・パケットにdoneビットをセットします。

保守／廃止

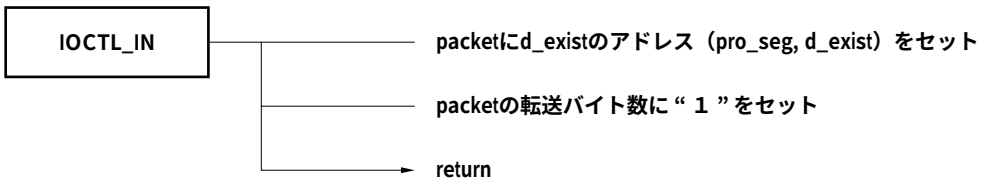
SPD-23


BUILD_BPB
ユニット番号処理
（IF：ユニット番号が無効）
[THEN]
packetのdoneビットに１をセット
return
リキャリブレイト処理
（WHILE：メディア２タイプ・チェック）
メディア・タイプ処理（2DD → 2HD）
リード ID 処理
（IF：RDIDのret_st = 0 ）
[THEN]
break
リード・データ処理（予約セクタ）
リードしたデータを格納する領域は，デバイス・ドライバ内に設定
（IF：ret_st = = 00H）
[THEN]
リードしたデータからBPBの値を取り出し，BPBポインタの示すアドレスへセット
packetにBPBポインタをセット
packetにメディア・ディスクリプタをセット
[ELSE]
エラー・セット処理
packetのdoneビットに１をセット
return

（g）コントロール文字列のインプット処理

モジュール名	IOCTL_IN
言　　語	C言語
入　　力	pro_seg, d_exist
出　　力	packet
[機　　能] （ｉ）コマンド・パケットにd_existのアドレスをセットします。 （ii）コマンド・パケットの転送バイト数に１をセットします。	

SPD-24

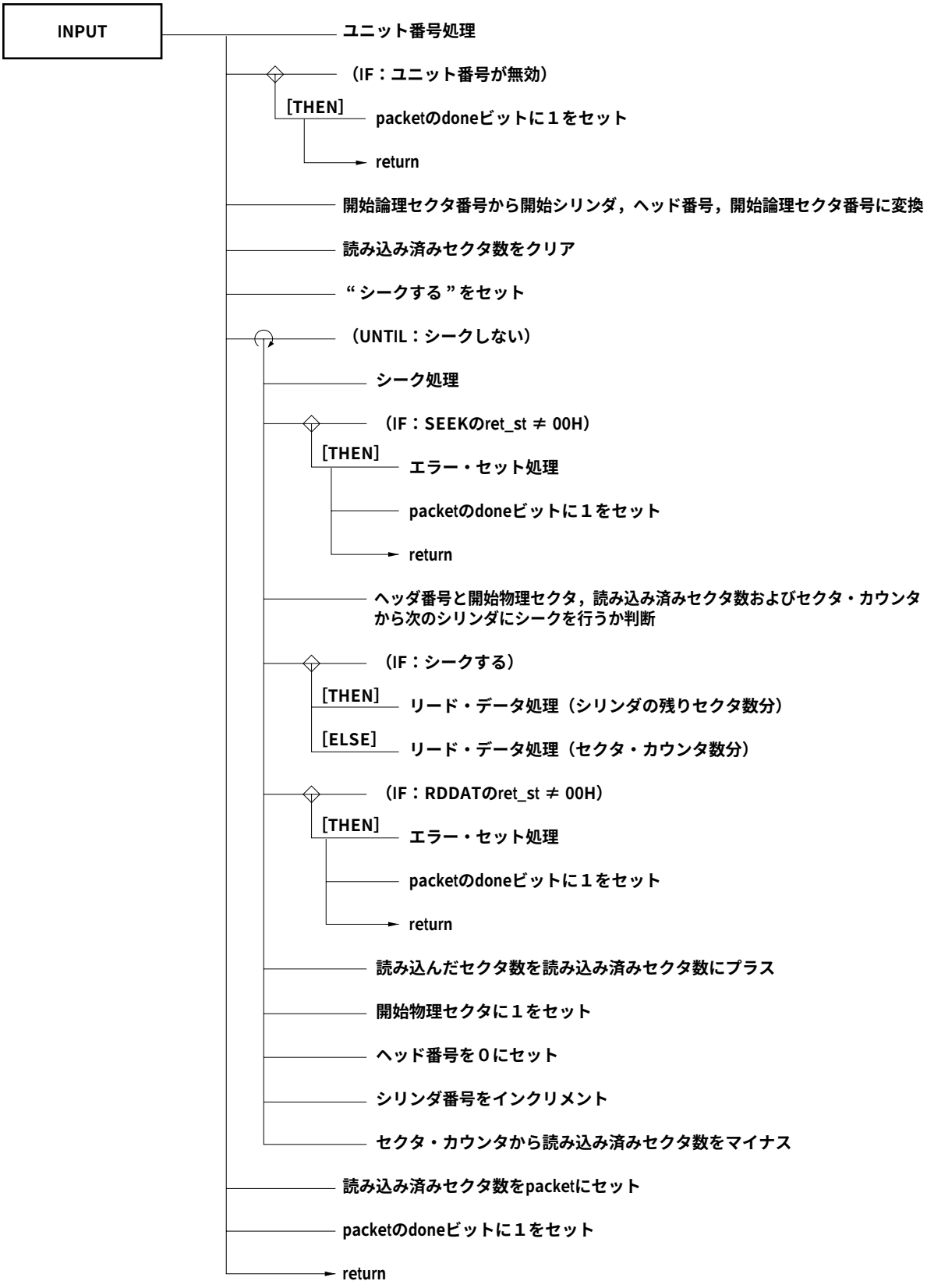


（h）インプット処理

モジュール名	INPUT
言　　語	C言語
入　　力	packet
出　　力	packet
[機　　能] （ｉ）リード・データ処理をコールしてデータをリードします。 （ii）リード・データ処理がエラー終了のときは，エラー・セット処理をコールします。 正常終了のときには，コマンド・パケットにdoneビットをセットします。	

保守／廃止

SPD-25


INPUT
ユニット番号処理
（IF：ユニット番号が無効）
[THEN]
packetのdoneビットに１をセット
return
開始論理セクタ番号から開始シリンダ，ヘッド番号，開始論理セクタ番号に変換
読み込み済みセクタ数をクリア
“ シークする ”をセット
（UNTIL：シークしない）
シーク処理
（IF：SEEKのret_st ≠ 00H）
[THEN]
エラー・セット処理
packetのdoneビットに１をセット
return
ヘッダ番号と開始物理セクタ，読み込み済みセクタ数およびセクタ・カウンタから次のシリンダにシークを行うか判断
（IF：シークする）
[THEN]
リード・データ処理（シリンダの残りセクタ数分）
[ELSE]
リード・データ処理（セクタ・カウンタ数分）
（IF：RDDATのret_st ≠ 00H）
[THEN]
エラー・セット処理
packetのdoneビットに１をセット
return
読み込んだセクタ数を読み込み済みセクタ数にプラス
開始物理セクタに１をセット
ヘッド番号を０にセット
シリンダ番号をインクリメント
セクタ・カウンタから読み込み済みセクタ数をマイナス
読み込み済みセクタ数をpacketにセット
packetのdoneビットに１をセット
return

（ｉ）アウトプット処理

モジュール名	OUTPUT
言　　語	C言語
入　　力	packet
出　　力	packet

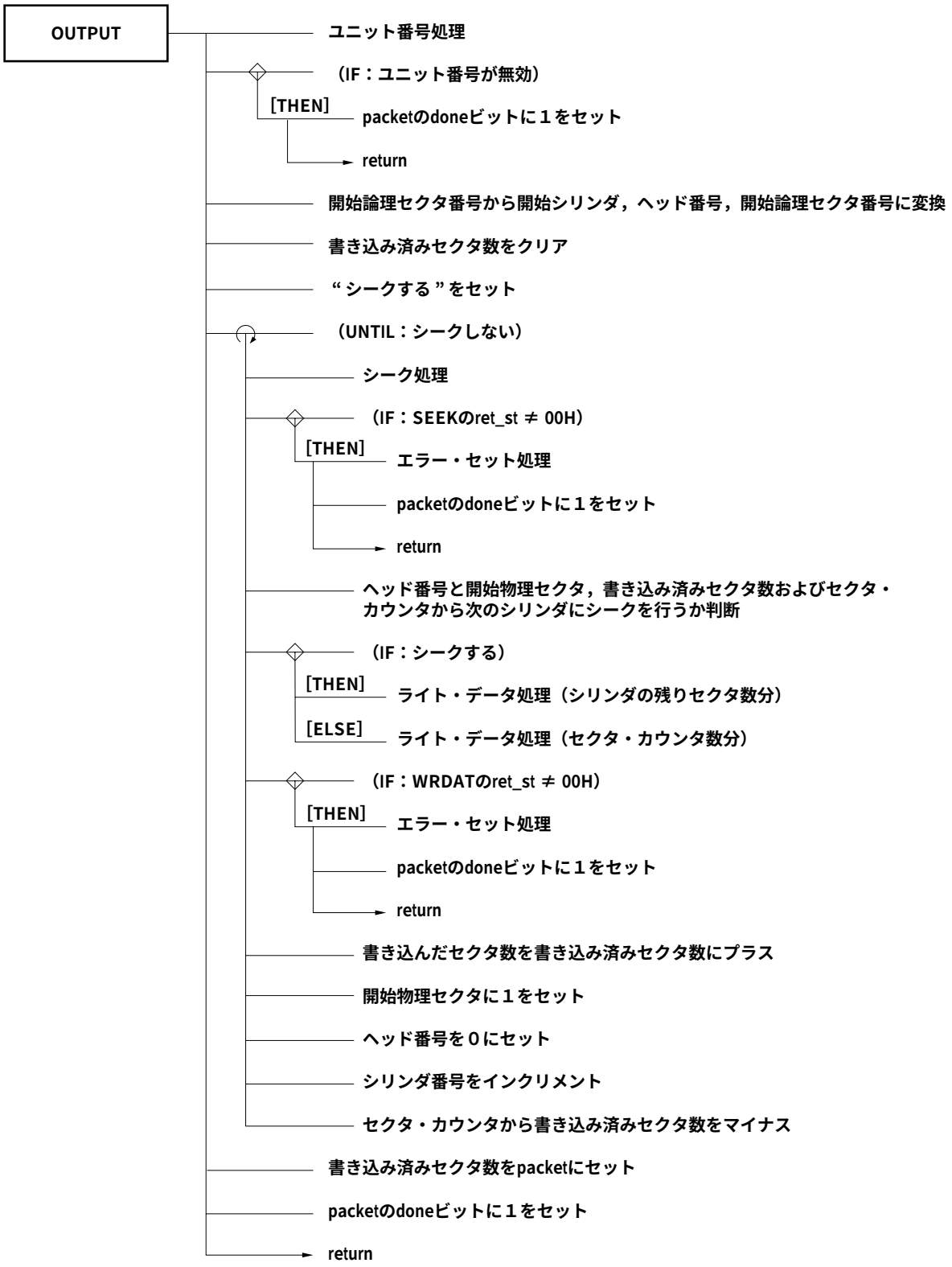
［機　　能］

（ｉ）ライト・データ処理をコールしてデータをライトします。

（ⅱ）ライト・データ処理がエラー終了のときは，エラー・セット処理をコールします。
　　正常終了のときには，コマンド・パケットにdoneビットをセットします。

保守／廃止

SPD-26


OUTPUT
ユニット番号処理
（IF：ユニット番号が無効）
[THEN]
packetのdoneビットに１をセット
return
開始論理セクタ番号から開始シリンダ，ヘッド番号，開始論理セクタ番号に変換
書き込み済みセクタ数をクリア
“ シークする ” をセット
（UNTIL：シークしない）
シーク処理
（IF：SEEKのret_st ≠ 00H）
[THEN]
エラー・セット処理
packetのdoneビットに１をセット
return
ヘッド番号と開始物理セクタ，書き込み済みセクタ数およびセクタ・
カウンタから次のシリンダにシークを行うか判断
（IF：シークする）
[THEN]
ライト・データ処理（シリンダの残りセクタ数分）
[ELSE]
ライト・データ処理（セクタ・カウンタ数分）
（IF：WRDATのret_st ≠ 00H）
[THEN]
エラー・セット処理
packetのdoneビットに１をセット
return
書き込んだセクタ数を書き込み済みセクタ数にプラス
開始物理セクタに１をセット
ヘッド番号を０にセット
シリンダ番号をインクリメント
セクタ・カウンタから書き込み済みセクタ数をマイナス
書き込み済みセクタ数をpacketにセット
packetのdoneビットに１をセット
return

（ｊ）アウトプット・ウィズ・ベリファイ処理

モジュール名	OUTPUT_V
言　　語	C言語
入　　力	packet
出　　力	packet

［機　　能］

（ｉ）ライト・データ処理をコールしてデータをライトします。

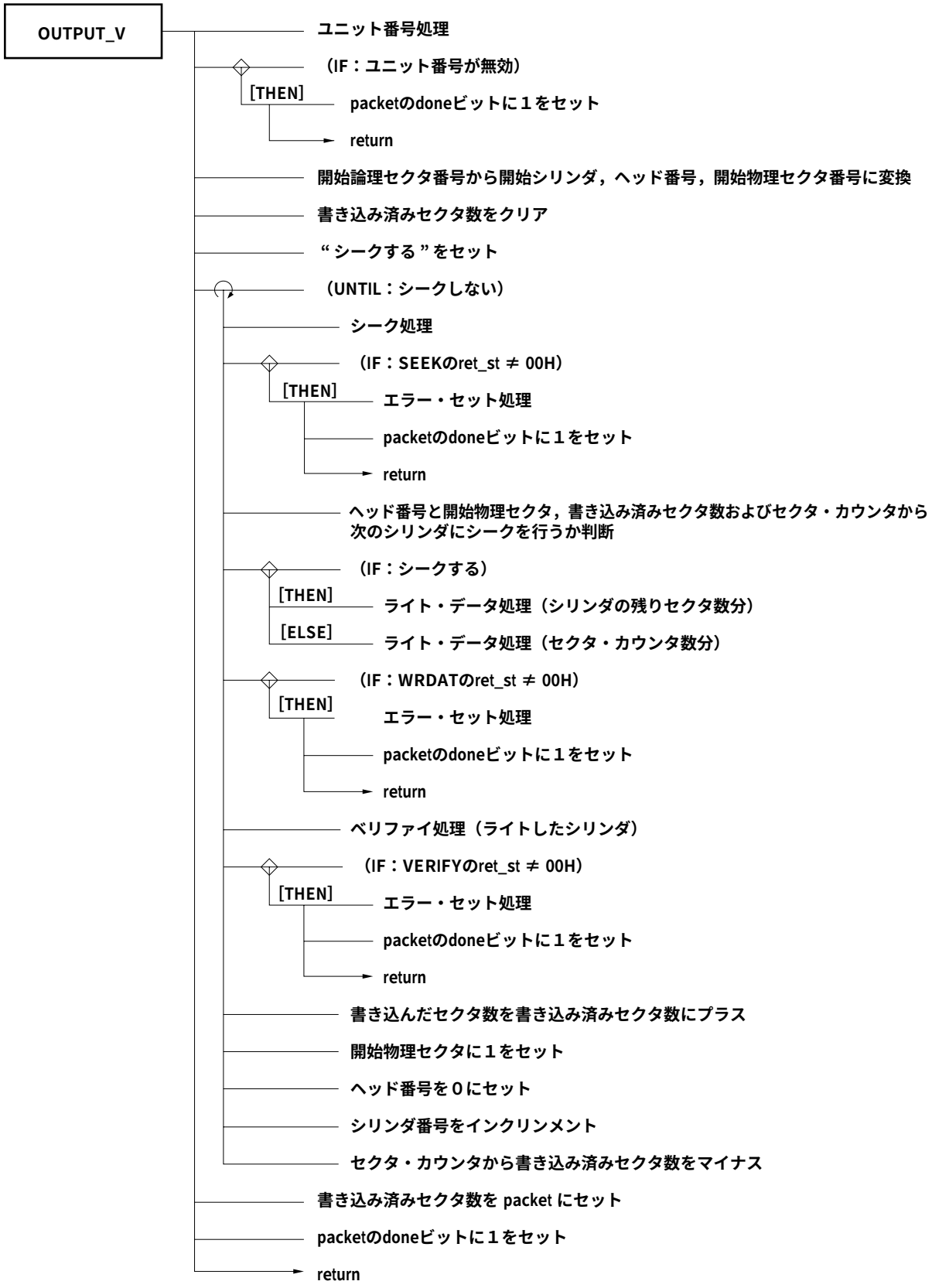
（ii）ベリファイ処理をコールしてライトしたデータをベリファイします。

（iii）ライト・データ処理またはベリファイ処理がエラー終了のときは，エラー・セット処理をコールします。

正常終了のときには，コマンド・パケットにdoneビットをセットします。

保守／廃止

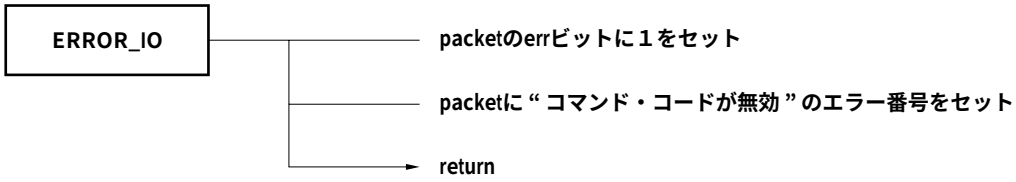
SPD-27


OUTPUT_V
ユニット番号処理
（IF：ユニット番号が無効）
[THEN]
packetのdoneビットに１をセット
return
開始論理セクタ番号から開始シリンダ，ヘッド番号，開始物理セクタ番号に変換
書き込み済みセクタ数をクリア
“ シークする ” をセット
（UNTIL：シークしない）
シーク処理
（IF：SEEKのret_st ≠ 00H）
[THEN]
エラー・セット処理
packetのdoneビットに１をセット
return
ヘッド番号と開始物理セクタ，書き込み済みセクタ数およびセクタ・カウンタから
次のシリンダにシークを行うか判断
（IF：シークする）
[THEN]
ライト・データ処理（シリンダの残りセクタ数分）
[ELSE]
ライト・データ処理（セクタ・カウンタ数分）
（IF：WRDATのret_st ≠ 00H）
[THEN]
エラー・セット処理
packetのdoneビットに１をセット
return
ベリファイ処理（ライトしたシリンダ）
（IF：VERIFYのret_st ≠ 00H）
[THEN]
エラー・セット処理
packetのdoneビットに１をセット
return
書き込んだセクタ数を書き込み済みセクタ数にプラス
開始物理セクタに１をセット
ヘッド番号を０にセット
シリンダ番号をインクリンメント
セクタ・カウンタから書き込み済みセクタ数をマイナス
書き込み済みセクタ数を packet にセット
packetのdoneビットに１をセット
return

（ｋ）I/Oリクエスト・エラー処理

モジュール名	ERROR_IO
言　　語	C言語
入　　力	なし
出　　力	packet
[機　　能] コマンド・パケットにI/Oリクエスト・コマンド・コードが無効であることをセットします。	

SPD-28

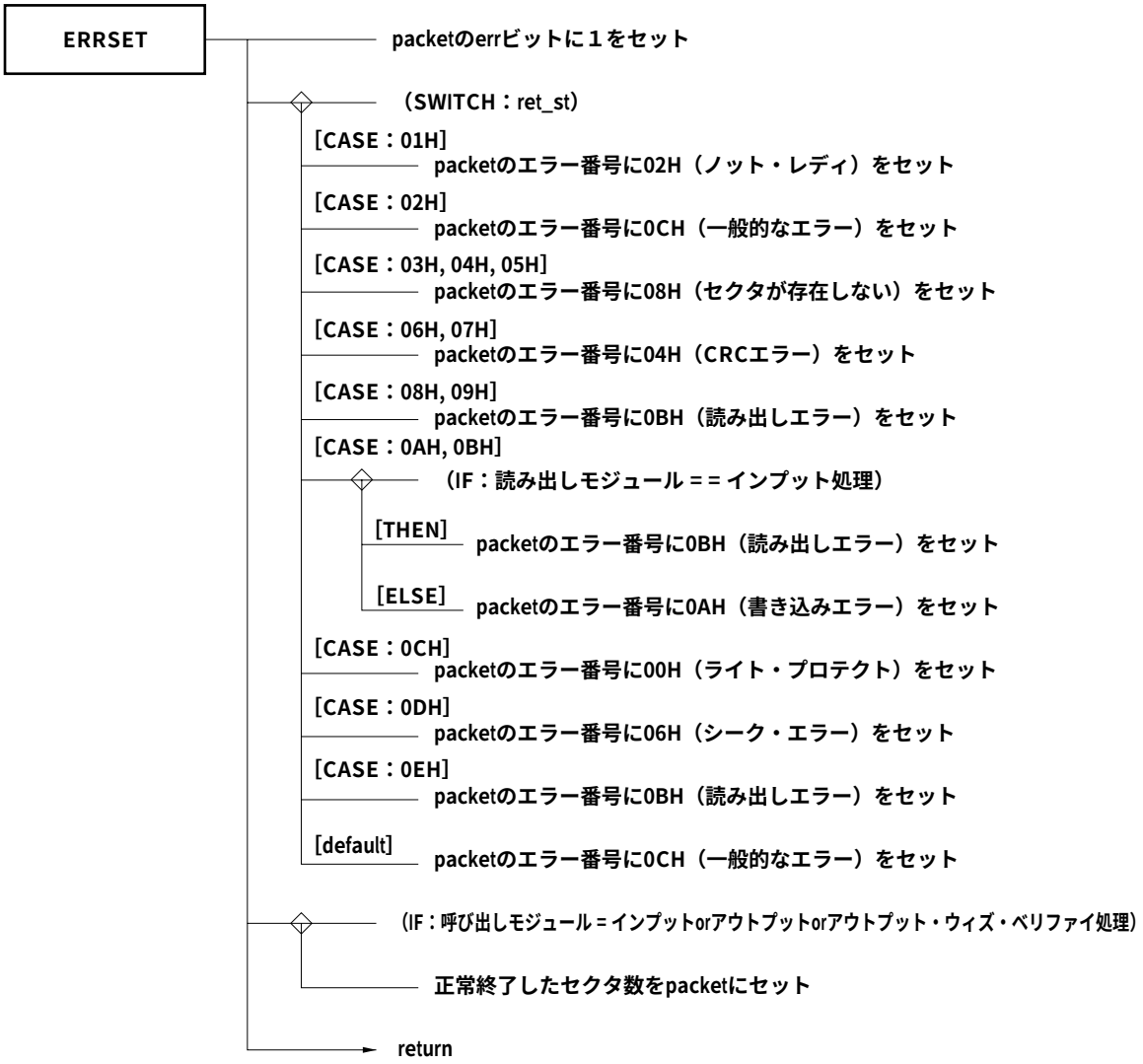


（ｌ）エラー・セット処理

モジュール名	ERRSET
言　　語	C言語
入　　力	packet, ret_st, ex_sw, tr_sect
出　　力	packet
[機　　能] （ｉ）実行したコマンドのリターン・ステータスにより，対応したエラーをセットします。 （ⅱ）インプット処理，アウトプット処理，アウトプット・ウィズ・ベリファイ処理では，正常に転送できたセクタ数をコマンド・パケットにセットします。	

保守／廃止

SPD-29


ERRSET
packetのerrビットに１をセット
（SWITCH：ret_st）
[CASE：01H]
packetのエラー番号に02H（ノット・レディ）をセット
[CASE：02H]
packetのエラー番号に0CH（一般的なエラー）をセット
[CASE：03H, 04H, 05H]
packetのエラー番号に08H（セクタが存在しない）をセット
[CASE：06H, 07H]
packetのエラー番号に04H（CRCエラー）をセット
[CASE：08H, 09H]
packetのエラー番号に0BH（読み出しエラー）をセット
[CASE：0AH, 0BH]
（IF：読み出しモジュール = = インプット処理）
[THEN]
packetのエラー番号に0BH（読み出しエラー）をセット
[ELSE]
packetのエラー番号に0AH（書き込みエラー）をセット
[CASE：0CH]
packetのエラー番号に00H（ライト・プロテクト）をセット
[CASE：0DH]
packetのエラー番号に06H（シーク・エラー）をセット
[CASE：0EH]
packetのエラー番号に0BH（読み出しエラー）をセット
[default]
packetのエラー番号に0CH（一般的なエラー）をセット
（IF：呼び出しモジュール = インプットorアウトプットorアウトプット・ウィズ・ベリファイ処理）
正常終了したセクタ数をpacketにセット
return

（m）ユニット番号処理

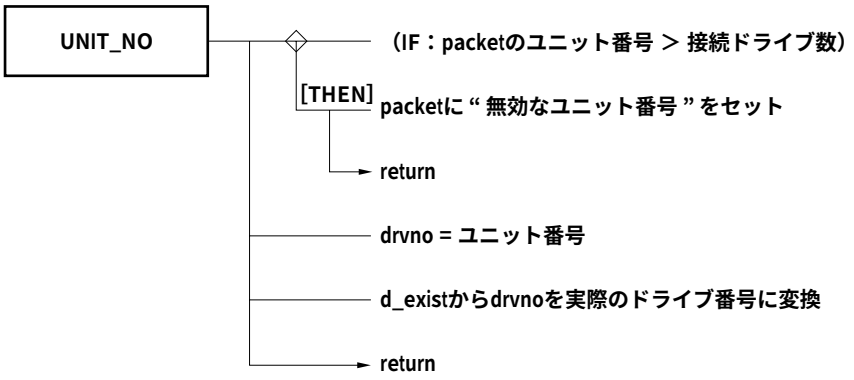
モジュール名	UNIT_NO
言　　語	C言語
入　　力	packet, d_exist, unit_ex
出　　力	packet, drvno

［機　　能］

（ｉ）コマンド・パケットのユニット番号がイニシャライズ・モジュールで得た接続ドライブ数を越えていないか比較します。越えていた場合は“無効なユニット番号”のエラーをコマンド・パケットにセットして処理を終了します。

（ⅱ）コマンド・パケットのユニット番号をμPD72070のドライブ番号に合わせてdrvnoにセットします。

SPD-30



（５）デバイス・ドライバの変数

ここではデバイス・ドライバ・モジュールで使用する変数を説明します。

表３－12に変数と機能を示します。

また，表３－13にエラー・コードとリターン・ステータスの対応表を示します。

表３－12　デバイス・ドライバの変数

変 数 名	意　　味
packet パケット・ポインタ （４バイト）	コマンド・パケットのポインタを保持する変数です。
pro_seg プログラム・エンド （セグメント値） （２バイト）	デバイス・ドライバの終了アドレスのセグメント値を保持する変数です。
pro_off プログラム・エンド （オフセット値） （２バイト）	デバイス・ドライバの終了アドレスのオフセット値を保持する変数です。
ret_st リターン・ステータス （１バイト）	LIOモジュールからのリターン・ステータスがセットされる変数です。 セットされる内容については，**表３－８　リターン・ステータス・コード**を参照してください。
ex_sw エグゼ・スイッチ （１バイト）	I/Oリクエストがリード／ライト系コマンドであることを示す変数です。 ex_sw =１のときリード／ライト系コマンドになります。
tr_sect トラック・セクタ （１バイト）	実行するメディアのトラックあたりのセクタ数を示す変数です。
d_exist ドライブ・イグジスト （１バイト）	ドライブの接続状況を示す変数です。 示される内容については，**3.2.2　（1）（a）デバイス・イニシャライズ処理**を参照してください。
drvno ドライブ・ナンバ （１バイト）	μPD72070から見たドライブ番号を示す変数です。
unit_ex ユニット・イグジスト （１バイト）	μPD72070に接続されているドライブ数を示す変数です。
time_i タイマ・インタラプト （１バイト）	PC-9801のインターバル・タイマの割り込みがあったことを知らせる変数です。FDDのモータOFF時にクリア（０をセット）します。

表３－13　エラー・コード，リターン・ステータス対応表

エラー・コード	内　容	リターン・ステータス
00H	ライト・プロテクト	0CH
01H	無効なユニット	14H
02H	ノット・レディ	01H
03H	無効なコマンド・コード	なし
04H	CRCエラー	06H，07H
06H	シーク・エラー	0DH
08H	セクタが存在しない	03H，04H，05H
0AH	書き込みエラー	0AH，0BH（ライト実行時）
0BH	読み出しエラー	08H，09H，0BH，0EH（リード実行時）
0CH	一般的なエラー	02H

３.２.４　フォーマット・コマンド

ここでは，μPD72070用デバイス・ドライバを用いたMS-DOS用フォーマット・コマンドを説明します。

（１）フォーマット仕様

このフォーマット・コマンドは，次のフロッピィ・ディスクをサポートします。

●2DD（９セクタ／トラック）720 Kバイト
●2HD（18セクタ／トラック）1.44 Mバイト

各メディアのフォーマット仕様を表３－14に示します。

保守／廃止

表３－14　フォーマット仕様

項　　目	2DD（９セクタ）	2HD
総トラック数	160	160
セクタ数／トラック	9	18
バイト数／セクタ	512	512
セクタ数／クラスタ	2	1
予約セクタ数	1	1
FAT数	2	2
ルート・ディレクトリのエントリ数	112	224
総論理セクタ数	1440	2880
メディア・ディスクプリタ FAT　ID	F9H	F0H
セクタ数/FAT	3	9
ルート・ディレクトリ・セクタ数	7	14

なおFATは12ビットFAT（標準フォーマット）でフォーマットを行います。

次にフォーマット後の各メディアの状態を説明します。

（a）2DD（９セクタ／トラック）

2DDのフォーマット後のセクタの割り当て状態を図３－６に示し，予約セクタ，FAT 領域，ルート・ディレクトリ領域，およびデータ領域の内容は図３－７，図３－８，図３－９，図３－10に示します。

図３－６　2DDのセクタ割り当て

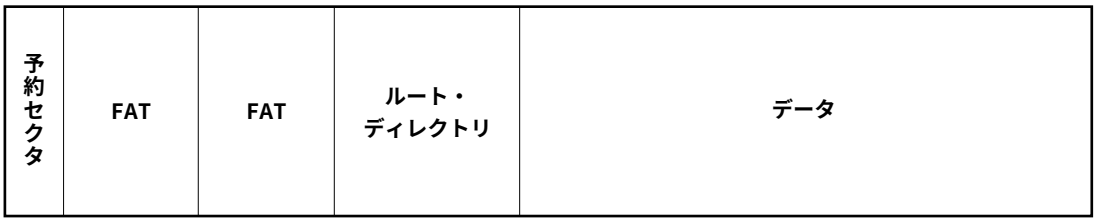


図３－７　2DDの予約セクタの内容

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
EB	1C	90	U_{55}	P_{50}	D_{44}	7_{37}	2_{32}	0_{30}	7_{37}	0_{30}	00	02	02	01	00
02	70	00	A0	05	F9	03	00	09	00	02	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	----------														00
⋮							⋮								⋮
00	----------														00

備考　書き込まれる値については，**表３－15　予約セクタの内容説明**を参照してください。

図３－８　2DDのFAT領域の内容

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
F9	FF	FF	00	00	00	00	00	00	00	00	00	00	00	00	00
00	----------														00
⋮							⋮								⋮
00	----------														00

図３－９　ルート・ディレクトリ領域の内容

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5
E5	E5	E5	----------												E5
00	E5	----------													E5
E5	----------														E5
⋮							⋮								⋮
00	E5	----------													E5
E5	----------														E5

図３－10　データ領域の内容

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5
E5	----------														E5
⋮							⋮								⋮
E5	----------														E5

（b）2HD（18セクタ／トラック）

2HDのフォーマット後のセクタの割り当て状態を図３－11に示し，予約セクタ，FAT領域の内容を図３－12，図３－13に示します。

ルート・ディレクトリ領域とデータ領域の内容は，2DDの各領域と同じ内容になります。

図３－11　2HDのセクタ割り当て

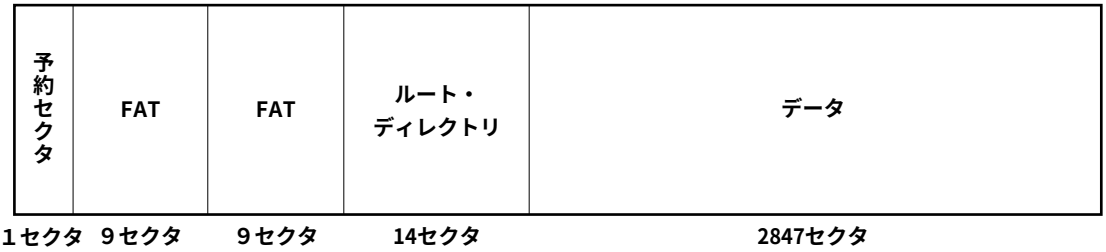


図３－12　2HDの予約セクタの内容

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
EB	1C	90	U_{55}	P_{50}	D_{44}	7_{37}	2_{32}	0_{30}	7_{37}	0_{30}	00	02	01	01	00
02	E0	00	40	0B	F0	09	00	12	00	02	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	----------														00
⋮							⋮								⋮
00	----------														00

備考　書き込まれる値については，**表３－15　予約セクタの内容説明**を参照してください。

図３－13　2HDのFAT領域の内容

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
F0	FF	FF	00	00	00	00	00	00	00	00	00	00	00	00	00
00	----------														00
⋮							⋮								⋮
00	----------														00

表３－15　予約セクタの内容説明

オフセット	長　さ	内　　容	備　　考
00H	３バイト	SHORTブランチ	オフセット値，1EHへのブランチ命令
03H	８バイト	OEMメーカ名	今回はASCⅡコードで“μPD72070”
0BH	１ワード	バイト数／セクタ	表３－14　フォーマット仕様参照
0DH	１バイト	セクタ数／クラスタ	
0EH	１ワード	予約セクタ数	
10H	１バイト	FAT数	
11H	１ワード	ルート・ディレクトリのエントリ数	
13H	１ワード	総論理セクタ数	
15H	１バイト	メディア・ディスクリプタ	
16H	１ワード	セクタ数/FAT	
18H	１ワード	セクタ数／トラック	
1AH	１ワード	ヘッド数	フロッピィ・ディスクなので02H
1CH	１ワード	隠れたセクタ数	隠れたセクタはないので00H
1EH	－	00H	本来はIPL領域なので今回はすべて00H

（2）フォーマット・コマンドの表示

フォーマット・コマンドを実行中の画面表示を図３－14に示します。

図３－14　フォーマット・コマンド画面表示

```
μPD72070用フォーマット・コマンド

        選択ドライブ　＊：

        選択メディア　＊＊＊



フォーマット中              ＊＊＊トラック
```

フォーマット・コマンドのメッセージ表示は，表３－８で示したリターン・ステータス・コードに対応しています。表３－16にメッセージの対応表を示します。

表３－16　メッセージ表示

リターン・ステータス・コード	メッセージ表示
00H	フォーマット終了。
01H	ドライブの準備ができていません。
02H, 06H, 07H, 08H, 0AH, 0BH	データ・エラーです。
03H, 04H	シーク・エラーです。
05H	セクタが見つかりません。
0CH	書き込み禁止です。
09H, 0DH, 0EH	エラーです。

（3）フォーマット・コマンドのリトライ処理

次にフォーマット・コマンドのリトライ処理を示します。

（a）リターン・ステータス・コードが03H, 04H（指定したシリンダがない）のときは，一度リキャリブレイトしてから再度シークし，リード／ライトをリトライします。それでもエラーがあるときは，エラー表示をします。

（b）ライト後ベリファイを行いエラーが発生したときは，再度ライト，ベリファイを繰り返します。それでもエラーがあるときはエラー表示をします。

（c）リターン・ステータス・コードが06H, 07H（CRCエラー）のときは，任意のシリンダへシークし再度目的のシリンダへシークしてからリード／ライトします。それでもエラーがあるときはエラー表示をします。

（４）フォーマット・コマンドのプログラム構成

図３－15にフォーマット・コマンド・プログラム構成を示します。ERRPRは，フォーマット・コマンド実行中に発生したエラー内容を表示するモジュールです。

表３－17にフォーマット・コマンドで使用するLIOモジュールを示します。

図３－15　フォーマット・コマンド・プログラム構成

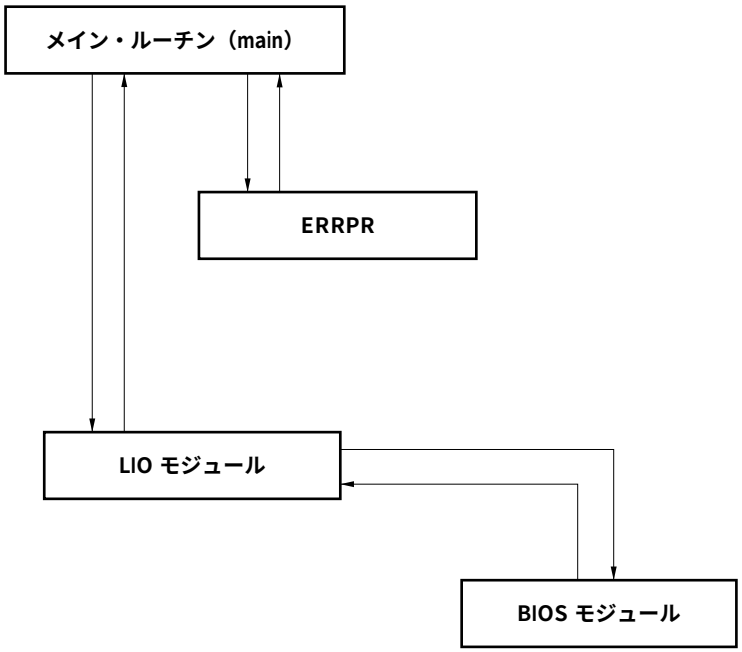


表３－17　使用するLIOモジュール

モジュール名	機　　能
MEDIA	選択されたメディア用に設定します。
RECAL	ドライブのヘッドをリキャリブレイトします。
SEEK	ヘッドをシークします。
RSEEK	リトライ時にヘッドをシークします。
TRFORMAT	１トラック分フォーマットします。
WRDAT	データをライトします。
VERIFY	ライトしたデータのベリファイを行います。

備考　LIOモジュールについては**3.2.2　LIO部**を参照してください。

（5）フォーマット・コマンド・モジュール説明

ここではフォーマット・コマンド・モジュールの説明をします。表３－18に示す内容に従って各モジュールを説明し，SPDチャートを示します。

表３－18　各モジュール説明内容

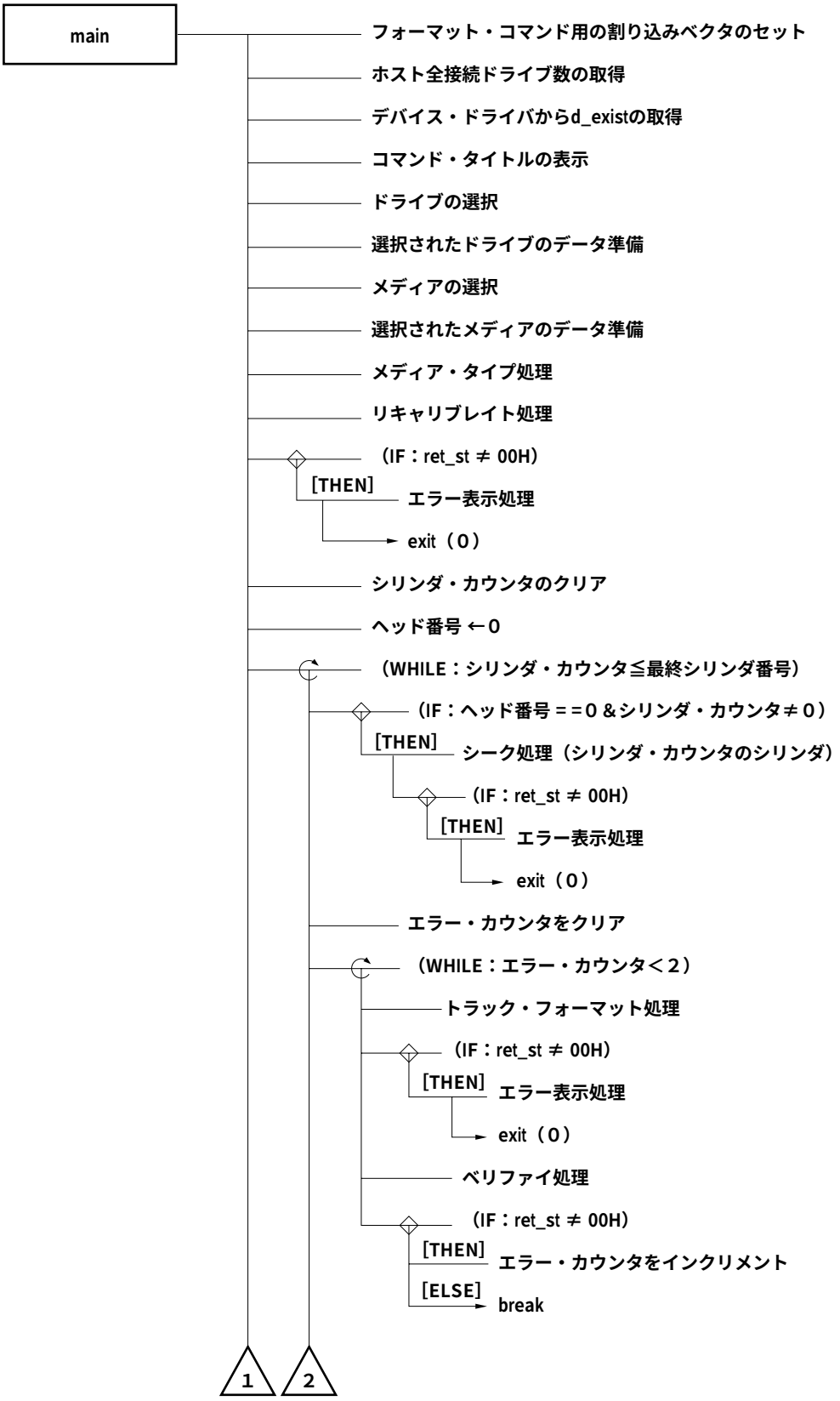
モジュール名	モジュール名を示します。
言　　語	関数がどの言語で作成されているかを示します。
入　　力	入力時の変数および，レジスタを示します。
出　　力	出力時の変数および，レジスタを示します。
［機　　能］ 関数の処理内容を示します。	

（a）メイン・プログラム

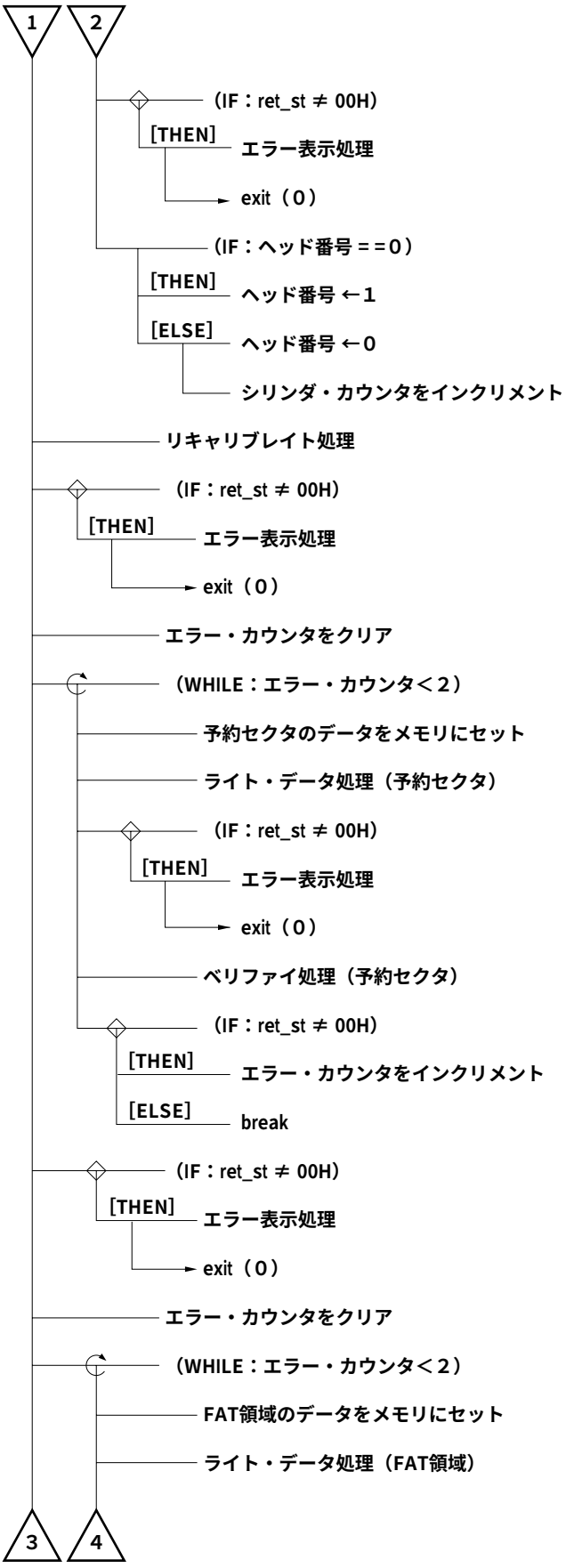
モジュール名	main
言　　語	C言語
入　　力	d_exist
出　　力	なし
［機　　能］ MS-DOS用フォーマットを実行するモジュールです。 注意　メイン・プログラム中で，デバイス・ドライバの割り込みベクタを書き換えており，コマンド終了後にベクタを戻します。フォーマット・コマンド実行中は，STOPキーなどで中断しないでください。	

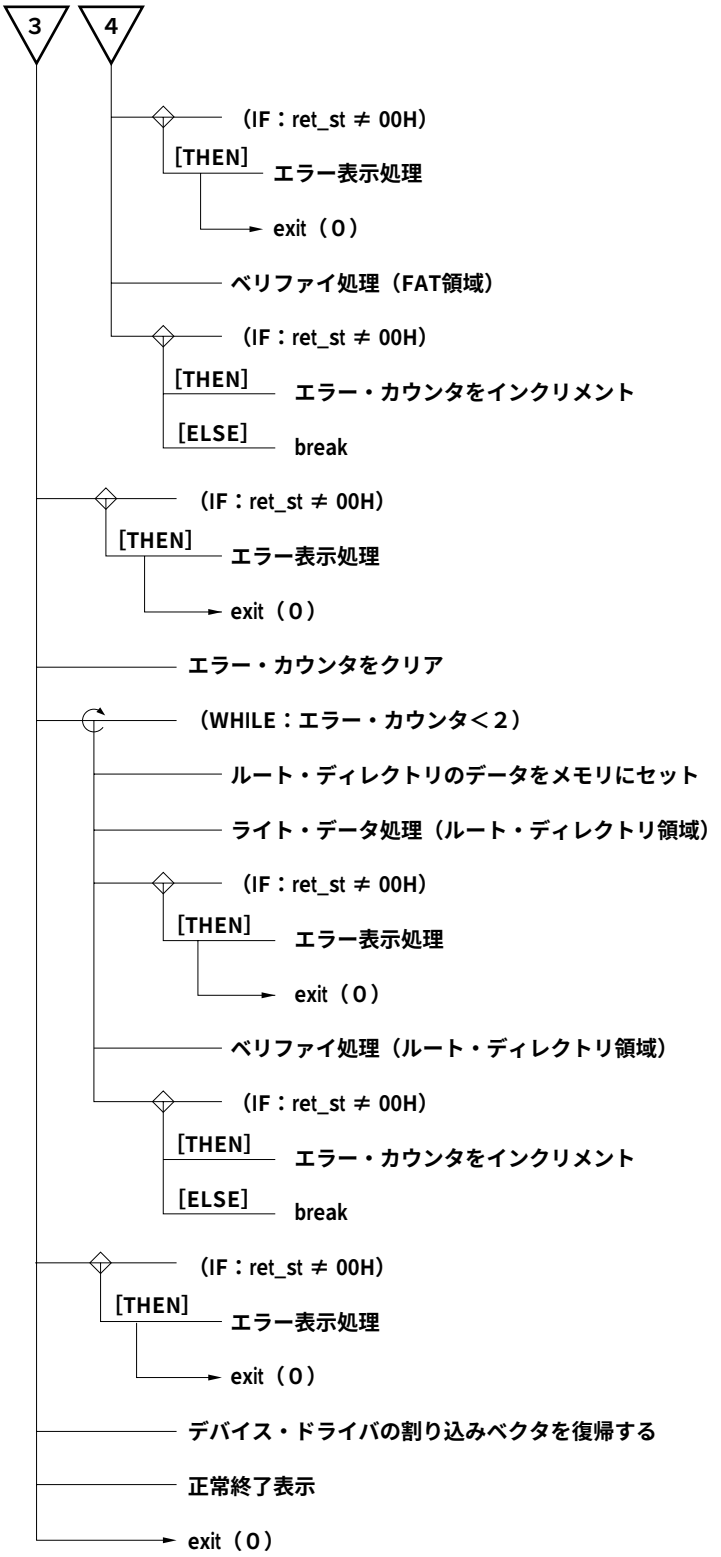
保守／廃止

SPD-31


main
フォーマット・コマンド用の割り込みベクタのセット
ホスト全接続ドライブ数の取得
デバイス・ドライバからd_existの取得
コマンド・タイトルの表示
ドライブの選択
選択されたドライブのデータ準備
メディアの選択
選択されたメディアのデータ準備
メディア・タイプ処理
リキャリブレイト処理
（IF：ret_st ≠ 00H）
[THEN]
エラー表示処理
exit（０）
シリンダ・カウンタのクリア
ヘッド番号 ←0
（WHILE：シリンダ・カウンタ≦最終シリンダ番号）
（IF：ヘッド番号 = =0 ＆シリンダ・カウンタ≠０）
[THEN]
シーク処理（シリンダ・カウンタのシリンダ）
（IF：ret_st ≠ 00H）
[THEN]
エラー表示処理
exit（０）
エラー・カウンタをクリア
（WHILE：エラー・カウンタ＜２）
トラック・フォーマット処理
（IF：ret_st ≠ 00H）
[THEN]
エラー表示処理
exit（０）
ベリファイ処理
（IF：ret_st ≠ 00H）
[THEN]
エラー・カウンタをインクリメント
[ELSE]
break
1
2

保守／廃止

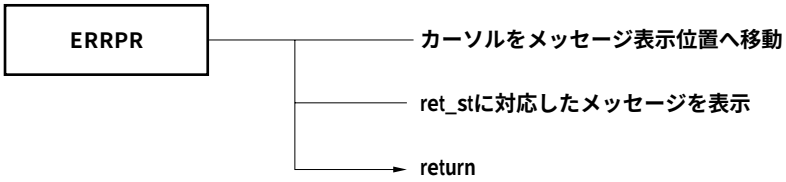

1
2
（IF：ret_st ≠ 00H）
[THEN]
エラー表示処理
exit（０）
（IF：ヘッド番号＝＝０）
[THEN]
ヘッド番号 ←１
[ELSE]
ヘッド番号 ←０
シリンダ・カウンタをインクリメント
リキャリブレイト処理
（IF：ret_st ≠ 00H）
[THEN]
エラー表示処理
exit（０）
エラー・カウンタをクリア
（WHILE：エラー・カウンタ＜２）
予約セクタのデータをメモリにセット
ライト・データ処理（予約セクタ）
（IF：ret_st ≠ 00H）
[THEN]
エラー表示処理
exit（０）
ベリファイ処理（予約セクタ）
（IF：ret_st ≠ 00H）
[THEN]
エラー・カウンタをインクリメント
[ELSE]
break
（IF：ret_st ≠ 00H）
[THEN]
エラー表示処理
exit（０）
エラー・カウンタをクリア
（WHILE：エラー・カウンタ＜２）
FAT領域のデータをメモリにセット
ライト・データ処理（FAT領域）
3
4


3
4
（IF：ret_st ≠ 00H）
[THEN]
エラー表示処理
exit（0）
ベリファイ処理（FAT領域）
（IF：ret_st ≠ 00H）
[THEN]
エラー・カウンタをインクリメント
[ELSE]
break
（IF：ret_st ≠ 00H）
[THEN]
エラー表示処理
exit（0）
エラー・カウンタをクリア
（WHILE：エラー・カウンタ＜２）
ルート・ディレクトリのデータをメモリにセット
ライト・データ処理（ルート・ディレクトリ領域）
（IF：ret_st ≠ 00H）
[THEN]
エラー表示処理
exit（0）
ベリファイ処理（ルート・ディレクトリ領域）
（IF：ret_st ≠ 00H）
[THEN]
エラー・カウンタをインクリメント
[ELSE]
break
（IF：ret_st ≠ 00H）
[THEN]
エラー表示処理
exit（0）
デバイス・ドライバの割り込みベクタを復帰する
正常終了表示
exit（0）

（b）エラー表示処理

モジュール名	ERRPR
言　　語	C言語
入　　力	ret_st
出　　力	なし
［機　　能］ エラー・メッセージをフォーマット・コマンド実行画面のメッセージ表示位置に表示します。	

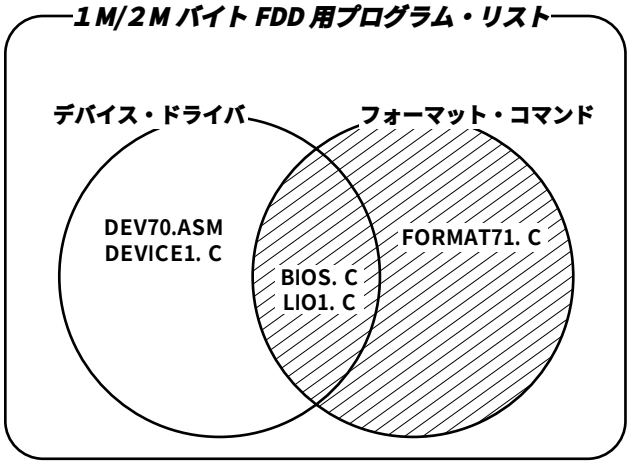
SPD-32



3.2.5　リ ス ト

次のプログラムのリストを示します。

●DEV70.ASM
●BIOS.C
●LIO1.C
●DEVICE1.C
●FORMAT71.C



●DEV70.ASM

(DEV70.ASM) p.1

```
_TEXT SEGMENT BYTE PUBLIC 'CODE'
_TEXT ENDS

_DATA SEGMENT WORD PUBLIC 'DATA'
_DATA ENDS

_BSS SEGMENT WORD PUBLIC 'BSS'
_BSS ENDS

_PEND SEGMENT WORD PUBLIC 'PEND'
_PEND ENDS

DGROUP GROUP _TEXT, _DATA, _BSS, _PEND
      ASSUME CS:_TEXT, DS:DGROUP, SS:DGROUP, ES:DGROUP

_DATA SEGMENT WORD PUBLIC 'DATA'
;
;
_packet       DD      ?                    ;コマンド・パケットのポインタ・セーブ領域
STACKSAVE     DD      ?                    ;スタック・ポインタのセーブ領域
DGROUP@       DW      ?
_pro_off      DW      ?                    ;プログラム終了アドレスのセーブ領域
_pro_seg      DW      ?
              DB      1024*2 DUP(?)        ;
STACK_TOP     LABEL WORD                   ;スタックの先頭アドレス
;
_DATA ENDS
;
;
_PEND SEGMENT WORD PUBLIC 'PEND'
PROGEND       LABEL WORD                   ;プログラムの終了アドレス
_PEND ENDS
;
;
;
PUBLIC        _packet
PUBLIC        _pro_off
PUBLIC        _pro_seg
PUBLIC        DGROUP@
;
_TEXT SEGMENT BYTE PUBLIC 'CODE'
;
EXTRN _IO_REQ:NEAR                         ;Ｉ／Ｏリクエスト処理
;
ORG   0
;
;*********************************************************
;*                                                       *
;*              デバイス・ヘッダの構成                   *
;*                                                       *
;*********************************************************
;
header        LABEL   WORD
      DD      -1                           ;リンク情報
      DW      6000H                        ;デバイス属性
      DW      STRATEGY                     ;ストラテジ・エントリ・ポイント
      DW      INTERRUPT                    ;割り込みエントリ・ポイント
      DB      4                            ;ユニット数
;
;************************************************************************
;*                                                                      *
;*    ストラテジ・エントリ処理        ＩＮ    : BXﾚｼﾞｽﾀ, ESﾚｼﾞｽﾀ       *
;*                                    ＯＵＴ  : packet                 *
;*                                                                      *
;************************************************************************
;
STRATEGY      PROC    FAR
;
      MOV     WORD PTR CS:_packet, BX      ;コマンド・パケットのポインタを待避する
      MOV     WORD PTR CS:_packet+2, ES
```

(DEV70.ASM) p.2

```
      RET
;
STRATEGY      ENDP
;
;
;*************************************************************************
;*                                                                       *
;*    割り込み処理エントリ処理          ＩＮ    : なし                   *
;*                                      ＯＵＴ  : pro_seg, pro_off       *
;*                                                                       *
;*************************************************************************
;
INTERRUPT     PROC    FAR
;
      PUSH    AX                              ;全レジスタをスタックへ待避させる
      PUSH    BX
      PUSH    CX
      PUSH    DX
      PUSH    SI
      PUSH    DI
      PUSH    BP
      PUSH    DS
      PUSH    ES
;
      MOV     AX, CS                          ;ＤＳ＝ＣＳ
      MOV     DS, AX
      MOV     CS:DGROUP@, AX
      MOV     WORD PTR STACKSAVE, SP          ;ＭＳ－ＤＯＳのスタック・ポインタの待避
      MOV     WORD PTR STACKSAVE+2, SS
;
      CLI
      MOV     SS, AX                          ;デバイス・ドライバのスタック領域のセット
      MOV     SP, OFFSET DGROUP:STACK_TOP
      STI
;
      MOV     _pro_off, OFFSET DGROUP:PROGEND ;デバイス・ドライバの終了アドレスをpro_off, pro_segにセットする
      MOV     _pro_seg, CS
      CALL    _IO_REQ                         ;Ｉ／Ｏリクエスト処理を呼び出す
;
      CLI
      MOV     SP, WORD PTR STACKSAVE          ;ＭＳ－ＤＯＳのスタックを復帰させる
      MOV     SS, WORD PTR STACKSAVE+2
      STI
;
      POP     ES                              ;レジスタを復帰させる
      POP     DS
      POP     BP
      POP     DI
      POP     SI
      POP     DX
      POP     CX
      POP     BX
      POP     AX
      RET
;
INTERRUPT     ENDP
;
_TEXT ENDS
;
END
```

●BIOS.C

(BIOS.C) p.1

```
#include<dos.h>

#define             OCW2            0x0000                  /*  割り込みコントローラのOCW2のIOアドレス  */
#define             STR70           0x00d8                  /*  μPD72070のｽﾃｰﾀｽ･ﾚｼﾞｽﾀのIOアドレス  */
#define             DTR70           0x00da                  /*  μPD72070のﾃﾞｰﾀ･ﾚｼﾞｽﾀのIOアドレス   */



struct retpra
{                                                           /*  ﾘﾀｰﾝ･ｽﾃｰﾀｽ･ﾊﾟﾗﾒｰﾀの構造体   */
      unsigned char c;                                      /*  実行終了後のシリンダ番号    */
      unsigned char h;                                      /*  実行終了後のヘッド番号      */
      unsigned char r;                                      /*  実行終了後のセクタ番号      */
      unsigned char n;                                      /*  １セクタのデータ長  */
};
      struct sr                                             /*  シーク・リザルトの構造体    */
{
      unsigned char dr_st;                                  /*  シーク・コマンド実行後のﾘｻﾞﾙﾄ･ｽﾃｰﾀｽ */
      unsigned char dr_pcn;                                 /*  シーク・コマンド終了後のヘッドの位置  */
};


void exit(int);
void COMMAND(unsigned char *, char);
void interrupt INTRPT(void);
char RESULT(void);


            /*****************************************************************/
            /*                                                               */
            /*  コマンド・ライト処理        ＩＮ    : com_dat[], com_num     */
            /*                              ＯＵＴ  : なし                   */
            /*                                                               */
            /*****************************************************************/

struct sr seek_r[4];
volatile unsigned char int_s;
volatile unsigned char ready;
unsigned char rslt_pt[7];

void COMMAND(unsigned char *com_pt, char com_num)
{
      unsigned char fdcst, dat;
      char count1;

      count1 = 0;                                           /*  カウンタのクリア    */
      while(count1 < com_num)                               /*  パラメータの転送ループ      */
      {
            do
            {
                  fdcst = inportb(STR70);                   /*  μPD72020のｽﾃｰﾀｽ･ﾚｼﾞｽﾀをリードする  */
                  fdcst = fdcst & 0xc0;                     /*  ｽﾃｰﾀｽ･ﾚｼﾞｽﾀのRQMビットとDIOビットのチェック */
            }
            while(fdcst != 0x80);                           /*  RQMビット＝1,DIOビット＝0になるまでくり返し */

            dat = *(com_pt + count1);
            outportb(DTR70, dat);                           /*  com_dat[count1]の内容をμPD72070の  */
                                                            /*  ﾃﾞｰﾀ･ﾚｼﾞｽﾀにライトする      */
            ++count1;                                       /*  カウンタをインクリメント    */
      }
}


            /*****************************************************/
            /*                                                   */
            /*  割り込み処理        ＩＮ    : なし               */
            /*                      ＯＵＴ  : int_s, seek_r[], ready  */
            /*                                                   */
            /*****************************************************/
```

(BIOS.C) p.2

```
void interrupt INTRPT(void)
{
      unsigned char fdcst, rsltst0, hedpos;

      int_s = 0x01;                                          /*  割り込み発生のフラグを立てる          */
      do
      {
            fdcst = inportb(STR70);                          /*  μPD72070のｽﾃｰﾀｽ･ﾚｼﾞｽﾀのリード        */
      }
      while(fdcst < 0x80);                                   /*  ｽﾃｰﾀｽ･ﾚｼﾞｽﾀのRQMビット=1になるまで繰り返す  */

      if(fdcst < 0xc0)                                       /*  データ転送方向の判断          */
      {                                                      /*  転送方向：ﾎｽﾄ→FDC  */
            outportb(DTR70, 0x08);                           /*  SENSE INTERRUPT STATUSコマンドの発行  */
            do
            {
                  fdcst = inportb(STR70);                    /*  ｽﾃｰﾀｽ･ﾚｼﾞｽﾀのリード */
            }
            while(fdcst < 0xc0);                             /*  データの転送が許可されたか  */
            rsltst0 = inportb(DTR70);                        /*  ﾃﾞｰﾀ･ﾚｼﾞｽﾀをリードしてﾘｻﾞﾙﾄ･ｽﾃｰﾀｽ0にセット  */
            if((rsltst0 & 0xc0) != 0x80)                     /*  無効コマンドの判定  */
            {
                  do                                         /*  無効コマンドではないときの処理        */
                  {
                        fdcst = inportb(STR70);              /*  ｽﾃｰﾀｽ･ﾚｼﾞｽﾀのリード */
                  }
                  while(fdcst < 0xc0);                       /*  データの転送が許可されたか  */
                  hedpos = inportb(DTR70);                   /*  ﾃﾞｰﾀ･ﾚｼﾞｽﾀをリードしてヘッドの位置を得る      */
                  if((rsltst0 & 0xc0) == 0xc0)               /*  割り込み要因の判断  */
                  {                                          /*  割り込み要因がドライブの状態遷移によるとき  */
                        int_s = 0x00;
                        if((rsltst0 & 0x08) == 0x00)         /*  ノット・レディ→レディ        */
                        {
                              switch(rsltst0 & 0x03)
                              {
                                    case 0x00:               /*  ドライブ０  */
                                          ready = ready | 0x01;
                                          break;
                                    case 0x01:               /*  ドライブ１  */
                                          ready = ready | 0x02;
                                          break;
                                    case 0x02:               /*  ドライブ２  */
                                          ready = ready | 0x04;
                                          break;
                                    case 0x03:               /*  ドライブ３  */
                                          ready = ready | 0x08;
                                          break;
                              }
                        }
                  }
                  else                                       /*  割り込み要因がｼｰｸ系のEﾌｪｰｽﾞ終了によるとき  */
                  {                                          /*  ｼｰｸ･ﾘｻﾞﾙﾄに結果をセット      */
                        seek_r[(rsltst0 & 0x03)].dr_st = rsltst0;
                        seek_r[(rsltst0 & 0x03)].dr_pcn = hedpos;
                  }
            }
      }
      else                                                   /*  転送方向：FDC→ホストのとき  */
      {
            RESULT();                                        /*  ﾘｻﾞﾙﾄ･ﾓｼﾞｭｰﾙ呼びだし          */
      }
      outportb(OCW2, 0x63);                                  /*  ＦＩコマンド発行     */
}
```

(BIOS.C) p.3

```
        /*****************************************************************/
        /*                                                               */
        /*  リザルト・リード処理        ＩＮ    : なし                   */
        /*                              ＯＵＴ  : rslt_pt[], rslt_num    */
        /*                                                               */
        /*****************************************************************/


char RESULT(void)
{
    unsigned char fdcst;
    char rslt_num;

    rslt_num = 0;                                   /*  ﾘｻﾞﾙﾄ･ﾅﾝﾊﾞのクリア  */

    while(1)                                        /*  ループ      */
    {
        do
        {
            fdcst = inportb(STR70);                 /* fdcstにμPD72070のｽﾃｰﾀｽ･ﾚｼﾞｽﾀの内容を読み込む*/
        }
        while(fdcst < 0x80);                        /*  ｽﾃｰﾀｽ･ﾚｼﾞｽﾀのRQMﾋﾞｯﾄ=1になるまでﾘｰﾄﾞする    */

        if((fdcst & 0x40) == 0x00)                  /*  ｽﾃｰﾀｽ･ﾚｼﾞｽﾀのDIOビット=0のとき      */
        {
            break;                                  /*  ループをぬける      */
        }
        rslt_pt[rslt_num] = inportb(DTR70);         /*  μPD72070のﾃﾞｰﾀ･ﾚｼﾞｽﾀをリードして   */
                                                    /*  rslt_pt+rslt_num(ｱﾄﾞﾚｽ)にセット          */
        ++rslt_num;                                 /*  ﾘｻﾞﾙﾄ･ﾅﾝﾊﾞをインクリメント  */
    }
    return(rslt_num);                               /*  リターン値：ﾘｻﾞﾙﾄ･ﾅﾝﾊﾞ      */
}
```

保守／廃止

●LIO1.C

(LIO1.C) p.1

```
#include<dos.h>


#define         OCW2        0x0000      /*  割り込みコントローラのOCW2のIOアドレス      */
#define         STR70       0x00d8      /*  μPD72070のｽﾃｰﾀｽ･ﾚｼﾞｽﾀのIOアドレス  */
#define         DTR70       0x00da      /*  μPD72070のﾃﾞｰﾀ･ﾚｼﾞｽﾀのIOアドレス   */
#define         DOR70       0x00d4      /*  μPD72070のﾃﾞｨｼﾞﾀﾙ･ｱｳﾄﾌﾟｯﾄ･ﾚｼﾞｽﾀのIOｱﾄﾞﾚｽ  */
#define         DRR70       0x00d8      /*  μPD72070のﾃﾞｰﾀ･ﾚｰﾄ･ﾚｼﾞｽﾀのIOアドレス      */
#define         CCR70       0x00de      /*  μPD72070のｺﾝﾌｨｷﾞｭﾚｰｼｮﾝ･ｺﾝﾄﾛｰﾙ･ﾚｼﾞｽﾀのIOｱﾄﾞﾚｽ*/
#define         BSWRP1      0x02d0      /*  ﾊﾞｳﾝﾀﾞﾘ･ｽｷｬﾝ回路のﾗｲﾄ･ﾎﾟｰﾄ1のIOアドレス      */
#define         IRMR        0x0002      /*  割り込みコントローラのﾏｽｸ･ﾚｼﾞｽﾀのIOアドレス */
#define         DADDR       0x000d      /*  DMAコントローラのｱﾄﾞﾚｽ･ﾚｼﾞｽﾀのIOアドレス    */
#define         DCOUNT      0x000f      /*  DMAコントローラのｶｳﾝﾄ･ﾚｼﾞｽﾀのIOｱﾄﾞﾚｽ      */
#define         DWRSMR      0x0015      /*  DMAｺﾝﾄﾛｰﾗのﾗｲﾄ･ｼﾝｸﾞﾙ･ﾏｽｸ･ﾚｼﾞｽﾀのIOｱﾄﾞﾚｽ  */
#define         DMODER      0x0017      /*  DMAコントローラのﾓｰﾄﾞ･ﾚｼﾞｽﾀのIOアドレス      */
#define         DCLBFF      0x0019      /*  DMAｺﾝﾄﾛｰﾗのｸﾘｱ･ﾊﾞｲﾄ･ﾎﾟｲﾝﾀ･FFのIOアドレス  */
#define         DBNKR       0x0025      /*  DMAコントローラのﾊﾞﾝｸ･ﾚｼﾞｽﾀのIOアドレス      */
#define         DBAAIM      0x0029      /*  DMAコントローラのﾊﾞﾝｸ･ｱﾄﾞﾚｽ･ｵｰﾄ･ｲﾝｸﾘﾒﾝﾄ･    */
                                        /*  ﾓｰﾄ･ﾚｼﾞｽﾀのIOアドレス       */
#define         INTVEC      0x0b        /*  割り込みベクタ番号  */


struct compra
{                                       /*  コマンド・パラメータの構造体        */
    union                               /*  転送バイト数の共用体        */
    {
        unsigned int dtsz;              /*  転送データのバイト数(int型) */
        struct                          /*  int型をchar型にわける構造体 */
        {
            unsigned char kai;          /*  下位８ビット分      */
            unsigned char jyoui;        /*  上位８ビット分      */
        } int_hen;

    } trsize;
    unsigned int dt_off;                /*  転送ﾃﾞｰﾀのｵﾌｾｯﾄ･ｱﾄﾞﾚｽ      */
    unsigned int dt_seg;                /*  転送ﾃﾞｰﾀのｾｸﾞﾒﾝﾄ･ｱﾄﾞﾚｽ      */
    unsigned char c;                    /*  シリンダ番号        */
    unsigned char h;                    /*  ヘッド番号  */
    unsigned char r;                    /*  開始セクタ番号      */
    unsigned char n;                    /*  １セクタのデータ長  */
    unsigned char dtl;                  /*  1ｾｸﾀあたりの処理ﾃﾞｰﾀ長     */
    unsigned char gsl;                  /*  GAP3の読み飛ばしバイト数    */
    unsigned char eot;                  /*  １トラックのセクタ数        */
    unsigned char d;                    /*  ﾌｫｰﾏｯﾄの書きこみﾃﾞｰﾀ･ﾊﾟﾀｰﾝ  */
};

struct retpra
{                                       /*  ﾘﾀｰﾝ･ｽﾃｰﾀｽ･ﾊﾟﾗﾒｰﾀの構造体   */
    unsigned char c;                    /*  実行終了後のシリンダ番号    */
    unsigned char h;                    /*  実行終了後のヘッド番号      */
    unsigned char r;                    /*  実行終了後のセクタ番号      */
    unsigned char n;                    /*  １セクタのデータ長  */
};
    struct sr                           /*  シーク・リザルトの構造体    */
{
    unsigned char dr_st;                /*  シーク・コマンド実行後のﾘｻﾞﾙﾄ･ｽﾃｰﾀｽ */
    unsigned char dr_pcn;               /*  シーク・コマンド終了後のヘッドの位置        */
};

struct form                             /*  フォーマット時のＩＤ部の書き込みデータの    */
                                        /*  構造体      */
{
    unsigned char c;                    /*  シリンダ番号        */
    unsigned char h;                    /*  ヘッド番号  */
    unsigned char r;                    /*  セクタ番号  */
    unsigned char n;                    /*  セクタあたりのフォーマット・バイト数        */
};

struct fpnt                             /*  ｆａｒポインタをセグメントとオフセットに    */
                                        /*  分割する構造体      */
```

保守／廃止

(L101.C) p.2

```
{
      unsigned int adoff;
      unsigned int adseg;
};

union fapnt                                                    /*  ｆａｒポインタの共用体宣言  */
{
      unsigned long pnt;                                       /*  ｆａｒポインタ      */
      struct fpnt pon;                                         /*  オフセットとセグメント      */
};


void exit(int);
void COMMAND(unsigned char *, char);
void interrupt INTRPT(void);
void D_INT(void);
char RESULT(void);
void MEDIAT(unsigned char);
unsigned char RECAL(unsigned char);
unsigned char SEEK(unsigned char, struct compra *);
unsigned char RSEEK(unsigned char, struct compra *);
unsigned char RDDAT(unsigned char, struct compra *, struct retpra *);
unsigned char RDDDAT(unsigned char, struct compra *, struct retpra *);
unsigned char RDID(unsigned char, struct retpra *);
unsigned char RDDAG(unsigned char, struct compra *, struct retpra *);
unsigned char TRFORMAT(unsigned char, struct compra *, struct retpra *);
unsigned char WRDAT(unsigned char, struct compra *, struct retpra *);
unsigned char WRDDAT(unsigned char, struct compra *, struct retpra *);
unsigned char VERIFY(unsigned char, struct compra *, struct retpra *);
void DRVSEL(unsigned char);
unsigned getkey(void);                                         /*  キー・コード取得関数        */
void interrupt TIME(void);



extern unsigned char int_s;
extern struct sr seek_r[];
extern unsigned char rslt_pt[];

unsigned char d_exist;
unsigned char time_i;

            /*************************************************************/
            /*                                                           */
            /*  デバイス・イニシャライズ処理        ＩＮ    :なし        */
            /*                                      ＯＵＴ  :d_exist     */
            /*                                                           */
            /*************************************************************/

void D_INT(void)
{
      unsigned char imrs, com_code, ret_st, drchk, bswpl, irr, pcn_b, st_r;
      char dcoun;
      union fapnt vector;

      d_exist = 0x00;                                          /*  d_existの初期化     */
      imrs = inportb(IRMR);                                    /*  割り込みｺﾝﾄﾛｰﾗのﾏｽｸ･ﾚｼﾞｽﾀをリードする      */
      imrs = imrs | 0x08;
      outportb(IRMR, imrs);                                    /*  ＩＲ３をマスクする  */

      vector.pnt = (unsigned long)INTRPT;                      /*  割り込みﾓｼﾞｭｰﾙのアドレスを共用体へｾｯﾄ      */

      poke(0x0000, (unsigned int)(INTVEC * 4), vector.pon.adoff);
                                                               /*  割り込みﾓｼﾞｭｰﾙのｵﾌｾｯﾄ･ｱﾄﾞﾚｽをベクタにｾｯﾄ    */

      poke(0x0000, (unsigned int)(INTVEC * 4 + 2), vector.pon.adseg);
                                                               /*  割り込みﾓｼﾞｭｰﾙのｾｸﾞﾒﾝﾄ･ｱﾄﾞﾚｽをベクタにｾｯﾄ*/
      outportb(DWRSMR, 0x07);                                  /*  DMAｺﾝﾄﾛｰﾗのチャネル３をマスクする   */
      outportb(DMODER, 0x63);                                  /*  DMAｺﾝﾄﾛｰﾗのチャネル３の転送方向を初期化     */
      outportb(DBAAIM, 0x0f);                                  /*  DMAｺﾝﾄﾛｰﾗのﾁｬﾈﾙ3のﾊﾞﾝｸ･ﾓｰﾄﾞを16MBに設定  */
```

保守／廃止

(L101.C) p.3

```
	bswpl = inportb(BSWRP1);                        /*  ﾗｲﾄ･ﾎﾟｰﾄ１をリードする       */
	bswpl = bswpl & 0xdf;                           /*  INT出力をマスクする */
	outportb(BSWRP1, bswpl);
	outportb(DOR70, 0x1c);                          /*  μPD72070の初期化   */
	outportb(DRR70, 0x80);                          /*  μPD72070のソフトウエア・リセット   */
	outportb(DRR70, 0x1c);                          /*  μPD72070の転送レートの初期化       */
	outportb(CCR70, 0x00);                          /*  μPD72070の転送レートの初期化       */
	bswpl = inportb(BSWRP1);                        /*  ﾗｲﾄ･ﾎﾟｰﾄ１をリードする       */
	bswpl = bswpl | 0x20;                           /*  INT出力をマスクを解除する   */
	outportb(BSWRP1, bswpl);
	do
	{
		outportb(OCW2, 0x0a);                   /*  割り込みコントローラのIRRのリード   */
		irr = inportb(OCW2);
	}while((irr & 0x08) == 0x00);                   /*  IR3がｱｸﾃｨﾌﾞになるまで繰り返し       */
	pcn_b = 0;
	while(pcn_b != 0x80)                            /*  発行したSENSE INTERRUPT STATUSが    */
                                                        /*  INVALID COMMANDになるまで繰り返す   */
	{
		do
		{
			st_r = inportb(STR70);          /*  μPD72070のｽﾃｰﾀｽ･ﾚｼﾞｽﾀのリード      */
		}
		while(st_r != 0x80);
		outportb(DTR70, 0x08);                  /*  SENSE INTERRUPT STATUSコマンドの発行        */
		do
		{
			st_r = inportb(STR70);          /*  μPD72070のｽﾃｰﾀｽ･ﾚｼﾞｽﾀのリード      */
		}
		while((st_r & 0xf0) != 0xd0);
		while((st_r & 0xc0) != 0x80)            /*  μPD72070のｽﾃｰﾀｽ･ﾚｼﾞｽﾀが80Hになるまで繰り返す*/
		{
			pcn_b = inportb(DTR70);         /*  μPD72070のﾃﾞｰﾀ･ﾚｼﾞｽﾀのリード       */
			do
			{
				st_r = inportb(STR70);  /*  μPD72070のｽﾃｰﾀｽ･ﾚｼﾞｽﾀのリード      */
			}
			while((st_r & 0x80) != 0x80);
		}
	}
	imrs = inportb(IRMR);                           /*  割り込みｺﾝﾄﾛｰﾗのﾏｽｸ･ﾚｼﾞｽﾀをリードする       */
	imrs = imrs & 0xf7;
	outportb(IRMR, imrs);                           /*  ＩＲ３のマスク解除  */

	com_code = 0x01;                                /*  メディアを２ＨＤにセット    */
	MEDIAT(com_code);                               /*  ﾒﾃﾞｨｱ･ﾀｲﾌﾟ･ﾓｼﾞｭｰﾙ呼びだし(2HDﾃﾞｨｽｸ) */

	dcoun = 0;
	bswpl = inportb(BSWRP1);                        /*  ﾗｲﾄ･ﾎﾟｰﾄ１をリードする       */
	bswpl = bswpl | 0x60;
	outportb(BSWRP1, bswpl);                        /*  アクティブ・レディをＯＮにする       */

	while(dcoun <= 3)                               /*  ドライブの接続チェック       */
	{
		com_code = dcoun;                       /*  dcounドライブにリキャリブレイト・コマンドを */
                                                        /*  発行する    */
		ret_st = RECAL(com_code);
		drchk = 0x01 << dcoun;
		if(ret_st == 0x00)                      /*  正常終了    */
		{
			d_exist = d_exist | drchk;      /*  ドライブ有りをd_existにセット       */
		}
		else                                    /*  異常終了    */
		{
			drchk = ~drchk;
			d_exist = d_exist & drchk;      /*  ドライブ未接続をd_existにセット     */
		}
		++dcoun;                                /*  ドライブ番号をインクリメント        */
	}
```

保守／廃止

(LIO1.C) p.4

```
	bswpl = inportb(BSWRP1);                                /*  ﾗｲﾄ･ﾎﾟｰﾄ1をリードする       */
	bswpl = (bswpl & 0xbf) | 0x20;
	outportb(BSWRP1, bswpl);                                /*  アクティブ・レディをＯＦＦにする    */

}


	/************************************************************/
	/*                                                          */
	/*  メディア・タイプ・セレクト処理      ＩＮ    : com_code  */
	/*                                      ＯＵＴ  : なし      */
	/*                                                          */
	/************************************************************/


void MEDIAT(unsigned char com_code)
{
	unsigned char deccom, com_dat[4];
	char com_num;

	deccom = 0x03 & com_code;                               /*  メディア・タイプをcom_codeより取りだす      */
	switch(deccom)                                          /*  メディア・タイプによって転送ﾚｰﾄをｾｯﾄする    */
	{
		case 0x00:
			outportb(CCR70, 0x02);                  /*  ２ＤＤディスクのとき(250kbps)       */
			break;
		case 0x01:
			outportb(CCR70, 0x00);                  /*  ２ＨＤディスクのとき(500kbps)       */
			break;
		default:
			outportb(CCR70, 0x03);                  /*  ２ＥＤディスクのとき(1.0Mbps)       */
			break;
	}

		/*************************************/
		/*  SPECIFYコマンドのデータ・セット    */
		/*************************************/

	com_dat[0] = 0x03;
	switch(deccom)
	{

		case 0x00:
			com_dat[1] = 0xd1;                      /*  ２ＤＤディスクのときのSRT,HUTのセット       */
			com_dat[2] = 0x40;                      /*  ２ＤＤディスクのときのHLTのセット   */
			break;
		case 0x01:
			com_dat[1] = 0xa1;                      /*  ２ＨＤディスクのときのSRT,HUTのセット       */
			com_dat[2] = 0x80;                      /*  ２ＨＤディスクのときのHLTのセット   */
			break;
		default:
			com_dat[1] = 0x01;                      /*  ２ＥＤディスクのときのSRT,HUTのセット       */
			com_dat[2] = 0xfe;                      /*  ２ＥＤディスクのときのHLTのセット   */
			break;
	}
	com_num = 0x03;
	COMMAND(&com_dat[0], com_num);                          /*  コマンド・モジュールの呼びだし      */

		/*****************************************/
		/*  CONFIGUREコマンドのデータ・セット    */
		/*****************************************/

	com_dat[0] = 0x13;
	com_dat[1] = 0x00;
	if( com_code >= 0x04)                                   /*  シーク動作の設定    */
	{
		com_dat[2] = 0x68;                              /*  リード／ライト系コマンドでシークを伴う      */
	}
	else
	{
```

(L101.C) p.5

```
        com_dat[2] = 0x28;                                      /*  リード／ライト系コマンドでシークを伴わない  */
    }
    com_dat[3] = 0xff;                                          /*  書き込み補償開始ﾄﾗｯｸの設定  */
    com_num = 0x04;
    COMMAND(&com_dat[0], com_num);                              /*  コマンド・モジュール呼びだし        */

        /***********************************************/
        /*  PERPENDICULAR MODEコマンドのデータ・セット  */
        /***********************************************/

    com_dat[0] = 0x12;
    com_dat[1] = 0x01;                                          /*  垂直磁化記録方式以外にセット        */
    com_num = 0x02;
    COMMAND(&com_dat[0], com_num);                              /*  コマンド・モジュール呼びだし        */

        /***********************************************/
        /* SELECT DRIVE TYPEコマンドのデータ・セット */
        /***********************************************/

    com_dat[0] = 0x32;
    com_dat[1] = 0x00;                                          /*  メディアが4MBFD以下のとき    */
    com_num = 0x02;
    COMMAND(&com_dat[0], com_num);                              /*  コマンド・モジュール呼びだし        */
}

        /*****************************************************/
        /*                                                   */
        /*  ドライブ・センス処理        ＩＮ   : com_code    */
        /*                              ＯＵＴ : sense_r     */
        /*                                                   */
        /*****************************************************/


void SENSE(unsigned char com_code)
{
    unsigned char com_dat[2], com_num, sense_r;

    com_dat[0] = 0x04;
    com_dat[1] = com_code;
    com_num = 2;
    COMMAND(&com_dat[0], com_num);                              /*  コマンド・モジュール呼び出し        */
    RESULT();                                                   /*  直接リザルト・モジュールで結果を受け取る    */
    sense_r = rslt_pt[0];                                       /*  リザルト・モジュールで受け取ったデータを    */
                                                                /*  sense_rにセットする */
    return;
}


        /*************************************************************/
        /*                                                           */
        /*  リキャリブレイト処理        ＩＮ    : com_code           */
        /*                              ＯＵＴ  : ret_st, ret_prm    */
        /*                                                           */
        /*************************************************************/


unsigned char RECAL(unsigned char com_code)
{
    signed char retryc, com_num;
    unsigned char com_dat[2], ret_st, coml_code, drn;
    struct compra coml_prm;

    drn = com_code & 0x03;                                      /*  ドライブ番号の抽出  */
    retryc = com_code >> 2;                                     /*  リトライ・カウンタのセット  */
    while(retryc >= 0)
    {
        com_dat[0] = 0x07;
        com_dat[1] = com_code & 0x03;
        com_num = 0x02;
        DRVSEL(com_dat[1]);
```

(L101.C) p.6

```
            COMMAND(&com_dat[0], com_num);
            while(int_s != 0x01);
            int_s = 0x00;
            if(seek_r[drn].dr_st < 0x40)
            {
                  ret_st = 0x00;                                  /*  コマンド正常終了    */
            }
            else
            {
                  if(seek_r[drn].dr_st < 0x70)
                  {
                        ret_st = 0x01;                            /*  ドライブのノット・レディ    */
                  }
                  else
                  {
                        ret_st = 0x0d;                            /*  リキャリブレイト・エラー    */
                  }
            }
            if(ret_st == 0x0d)                                    /*  リキャリブレイト・エラーのときの処理        */
            {
                  com1_code = 0x04 | com_dat[1];                  /*  20ｼﾘﾝﾀﾞ求心方向へシークしてからリトライ     */
                  com1_prm.c = 20;
                  RSEEK(com1_code, &com1_prm);
                  --retryc;
            }
            else                                                  /*  リキャリブレイト・エラー以外のエラー又は    */
                                                                  /*  正常終了のとき      */
            {
                  break;                                          /*  そのまま終了        */
            }
      }
      return ret_st;
}


            /*****************************************************/
            /*                                                   */
            /*  シーク処理          ＩＮ    : com_code, com_prm  */
            /*                      ＯＵＴ  : ret_st, ret_prm    */
            /*                                                   */
            /*****************************************************/


unsigned char SEEK(unsigned char com_code, struct compra *com_prm)
{
      unsigned char com_dat[3], ret_st;
      char com_num;

      DRVSEL(com_code);
      com_dat[0] = 0x0f;                                          /*  ｼｰｸ･ｺﾏﾝﾄﾞ用データのセット   */
      com_dat[1] = com_code;
      com_dat[2] = com_prm->c;
      com_num = 0x03;
      COMMAND(&com_dat[0], com_num);                              /*  ｺﾏﾝﾄﾞ･ﾓｼﾞｭｰﾙ呼び出し        */

      while(int_s != 0x01);                                       /*  コマンド終了割り込み待ち    */
      int_s = 0x00;
      if(seek_r[com_code].dr_st < 0x40)                           /*  コマンドの終了結果の判断    */
      {
            ret_st = 0x00;                                        /*  コマンド正常終了    */
      }
      else
      {
            ret_st = 0x01;                                        /*  ドライブのノット・レディ    */
      }
      return ret_st;
}
```

保守／廃止

(LIOI.C) p.7

```
            /*************************************************************/
            /*                                                           */
            /*  リラティブ・シーク処理      ＩＮ    : com_code, com_prm  */
            /*                              ＯＵＴ  : ret_st, ret_prm    */
            /*                                                           */
            /*************************************************************/


unsigned char RSEEK(unsigned char com_code, struct compra *com_prm)
{
      unsigned char com_dat[3], ret_st, drn;
      char com_num;

      drn = com_code & 0x03;                                    /*  ドライブ番号の抽出  */
      if((com_code & 0x04) == 0x04)                             /*  ﾘﾗﾃｨﾌﾞ･ｼｰｸ用データのセット  */
      {
            com_dat[0] = 0xcf;                                  /*  求心方向へシークするデータ  */
      }
      else
      {
            com_dat[0] = 0x8f;                                  /*  遠心方向へシークするデータ  */
      }
      com_dat[1] = com_code & 0x03;
      com_dat[2] = com_prm->c;
      com_num = 0x03;
      DRVSEL(com_dat[1]);                                       /*  ドライブ・セレクト信号の選択        */
      COMMAND(&com_dat[0], com_num);                            /*  ｺﾏﾝﾄﾞ･ﾓｼﾞｭｰﾙ呼び出し        */

      while(int_s != 0x01);                                     /*  割り込み待ち        */
      int_s = 0x00;
      if(seek_r[drn].dr_st < 0x40)                              /*  コマンドの終了結果の判断    */
      {
            ret_st = 0x00;                                      /*  コマンド正常終了    */
      }
      else
      {
            ret_st = 0x01;                                      /*  ドライブのノット・レディ    */
      }
      return ret_st;
}


            /*************************************************************/
            /*                                                           */
            /*  リード・データ処理  ＩＮ    : com_code, com_prm          */
            /*                      ＯＵＴ  : ret_st, ret_prm            */
            /*                                                           */
            /*************************************************************/


unsigned char RDDAT(unsigned char com_code, struct compra *com_prm, struct retpra *ret_prm)
{
      unsigned char com_dat[9], adrss[3], com1_code, ret_st, ret1_st, dvno;
      signed char com_num, retryc, i;
      unsigned long madrss;
      struct compra com1_prm;

      dvno = com_code & 0x03;                                   /*  ドライブ番号のデコード      */
      DRVSEL(dvno);                                             /*  ドライブ・セレクト信号の選択  */
      retryc = (com_code & 0x1c) >> 2;                          /*  ﾘﾄﾗｲ･ｶｳﾝﾀのセット    */
      com_dat[0] = 0x06 | (0xe0 & com_code);                    /*  ﾘｰﾄﾞ･ﾃﾞｰﾀ用データのセット    */
      if(com_prm->h == 0x00)                                    /*  ドライブとヘッド番号の設定  */
      {
            com_dat[1] = com_code & 0x03;                       /*  ヘッド番号＝０のときの設定  */
      }
      else
      {
            com_dat[1] = (com_code | 0x04) & 0x07;              /*  ヘッド番号＝１のときの設定  */
      }
```

保守／廃止

(L101.C) p.8

```
--com_prm->trsize.dtsz;                                    /*  転送データ・サイズをデクリメント    */

while(retryc >= 0)
{
        madrss = com_prm->dt_seg;                          /*  転送アドレスの設定  */
        madrss = (madrss << 4) + com_prm->dt_off;
        i = 0;
        while(i <= 2)
        {
                adrss[i] = (unsigned char)(madrss >> (8 * i));
                                                           /*  転送アドレスを8ﾋﾞｯﾄごとに分解       */
                ++i;
        }

        outportb(DMODER, 0x47);                            /*  DMAｺﾝﾄﾛｰﾗのﾓｰﾄﾞ･ﾚｼﾞｽﾀに転送方向をセット      */
        outportb(DCLBFF, 0x00);                            /*  DMAｺﾝﾄﾛｰﾗのｸﾘｱ･ﾊﾞｲﾄ･ﾎﾟｲﾝﾀFFをリセット        */
        outportb(DADDR,adrss[0]);                          /*  DMAｺﾝﾄﾛｰﾗのｱﾄﾞﾚｽ･ﾚｼﾞｽﾀにｱﾄﾞﾚｽ(0-7ﾋﾞｯﾄ)をｾｯﾄ */
        outportb(DADDR,adrss[1]);                          /*  DMAｺﾝﾄﾛｰﾗのｱﾄﾞﾚｽ･ﾚｼﾞｽﾀにｱﾄﾞﾚｽ(8-15ﾋﾞｯﾄ)をｾｯﾄ*/
        outportb(DBNKR,adrss[2]);                          /*  DMAｺﾝﾄﾛｰﾗのﾁｬﾈﾙ3ﾊﾞﾝｸ･ﾚｼﾞｽﾀにｱﾄﾞﾚｽ(16-23ﾋﾞｯﾄ)*/
                                                           /*  をセット     */
        outportb(DCOUNT,com_prm->trsize.int_hen.kai);
                                                           /*  DMAｺﾝﾄﾛｰﾗのｶｳﾝﾄ･ﾚｼﾞｽﾀにﾃﾞｰﾀ･ｻｲｽﾞをセット     */
        outportb(DCOUNT,com_prm->trsize.int_hen.jyoui);
                                                           /*  DMAｺﾝﾄﾛｰﾗのｶｳﾝﾄ･ﾚｼﾞｽﾀにﾃﾞｰﾀ･ｻｲｽﾞをセット     */
        outportb(DWRSMR,0x03);                             /*  DMAｺﾝﾄﾛｰﾗのﾁｬﾈﾙ3のﾏｽｸを解除する      */
        com_dat[2] = com_prm->c;                           /*  シリンダ番号の設定  */
        com_dat[3] = com_prm->h;                           /*  ヘッド番号の設定     */
        com_dat[4] = com_prm->r;                           /*  開始セクタ番号の設定         */
        com_dat[5] = com_prm->n;                           /*  セクタあたりのバイト数の設定         */
        com_dat[6] = com_prm->eot;                         /*  １トラックのセクタ数の設定  */
        com_dat[7] = com_prm->gsl;                         /*  GAP3の読み飛ばしバイト数の設定       */
        com_dat[8] = com_prm->dtl;                         /*  com_prm.n≠0なので値に意味はない     */
        com_num = 0x09;
        while(time_i != 1);                                /*  モータの回転安定時間待ち     */
        COMMAND(&com_dat[0], com_num);                     /*  ｺﾏﾝﾄﾞ･ﾓｼﾞｭｰﾙ呼びだし         */

        while(int_s != 0x01);                              /*  コマンド終了割り込み待ち     */
        int_s = 0x00;

        if(rslt_pt[0] >= 0x40)                             /*  コマンドの終了結果の判断     */
        {
                if((rslt_pt[0] & 0x08) == 0x08)
                {
                        ret_st = 0x01;                     /*  ドライブのノット・レディ     */
                }
                else
                {
                        switch(rslt_pt[1])
                        {
                                case 0x01:
                                        if((rslt_pt[2] & 0x01) == 0x01)
                                        {
                                                ret_st = 0x08;          /*  ＤＡＭ非検出         */
                                        }
                                        else
                                        {
                                                ret_st = 0x02;          /*  ＩＤＡＭ非検出       */
                                        }
                                        break;
                                case 0x04:
                                        switch(rslt_pt[2])
                                        {
                                                case 0x02:
                                                        ret_st = 0x03;  /*  指定したシリンダがない       */
                                                        break;
                                                case 0x10:
                                                        ret_st = 0x04;  /*  指定したシリンダがない       */
                                                        break;
                                                case 0x00:
                                                        ret_st = 0x05;  /*  指定したセクタがない         */
```

(L101.C) p.9

```
                                        break;
                                default:
                                        break;
                        }
                        break;
                case 0x20:
                        if((rslt_pt[2] & 0x20) == 0x20)
                        {
                                ret_st = 0x07;  /*  データ部のＣＲＣエラー       */
                        }
                        else
                        {
                                ret_st = 0x06;  /*  ＩＤ部のＣＲＣエラー         */
                        }
                        break;
                case 0x10:
                        ret_st = 0x0a;          /*  オーバラン・エラー  */
                        break;
                case 0x80:
                        ret_st = 0x0b;          /*  最終セクタを越えてアクセスしようとした      */
                        break;
                default:
                        break;
                }
        }
}
else
{
        if(rslt_pt[2] < 0x40)
        {
                ret_st = 0x00;                  /*  コマンド正常終了    */
        }
        else
        {
                ret_st = 0x09;                  /*  ＤＤＡＭ付きデータをリードした      */
        }
}

ret_prm->c = rslt_pt[3];                        /*  コマンド終了時のﾘｻﾞﾙﾄ･ﾊﾟﾗﾒｰﾀのセット        */
ret_prm->h = rslt_pt[4];
ret_prm->r = rslt_pt[5];
ret_prm->n = rslt_pt[6];

switch(ret_st)                                  /*  リトライ実行の判定  */
{
        case 0x00:                              /*  ﾘﾀｰﾝ･ｽﾃｰﾀｽが00H,01Hのとき   */
        case 0x01:
                retryc = 0;                     /*  リトライなし        */
                break;
        case 0x02:                              /*  ﾘﾀｰﾝ･ｽﾃｰﾀｽが02H,05H-08Hのとき       */
        case 0x05:
        case 0x06:
        case 0x07:
        case 0x08:
                if(com_prm->c <= 50)            /*  リードするシリンダの位置の判断(シークする   */
                                                /*  方向の決定) */
                {
                        com1_code = com_dat[1] | 0x04;  /*  シリンダの位置が50シリンダ以下ならば        */
                }                               /*  求心方向へシークする        */
                else
                {
                        com1_code = com_dat[1] & 0x03;  /*  シリンダの位置が51シリンダ以上ならば        */
                }                               /*  遠心方向にシークする        */

                com1_prm.c = 20;                /*  シークするシリンダ数の設定  */

                ret1_st = RSEEK(com1_code, &com1_prm);
                                                /*  ﾘﾗﾃｨﾌﾞ･ｼｰｸの実行    */

                if(ret1_st != 0x00)             /*  ﾘﾗﾃｨﾌﾞ･ｼｰｸの実行結果がエラーか？    */
```

(L101.C) p.10

```
                        {
                                retryc = 0;
                                break;                          /*  リトライせずに終了する      */
                        }
                        coml_code = com_code & 0x03;            /*  リードするシリンダへ再シークする    */
                        coml_prm.c = com_prm->c;
                        retl_st = SEEK(coml_code, &coml_prm);
                                                                /*  シーク実行  */

                        if(retl_st != 0x00)                     /*  シークの実行結果がエラーか？        */
                        {
                                retryc = 0;                     /*  リトライせずに終了  */
                        }
                        break;
                default:
                        break;
        }
        --retryc;                                               /*  ﾘﾄﾗｲ･ｶｳﾝﾀをﾃﾞｸﾘﾒﾝﾄしてリトライする  */
    }

    outportb(DWRSMR, 0x07);                                     /*  DMAｺﾝﾄﾛｰﾗのﾁｬﾈﾙ3をマスクする        */
    return ret_st;
}


        /*************************************************************************/
        /*                                                                       */
        /*  リード・デリーテッド・データ処理     ＩＮ    : com_code, com_prm      */
        /*                                       ＯＵＴ  : ret_st, ret_prm        */
        /*                                                                       */
        /*************************************************************************/


unsigned char RDDDAT(unsigned char com_code, struct compra *com_prm, struct retpra *ret_prm)
{
    unsigned char com_dat[9], coml_code, adrss[3], ret_st, retl_st, dvno;
    signed char com_num, retryc, i;
    unsigned long madrss;
    struct compra coml_prm;

    dvno = com_code & 0x03;                                     /*  ドライブ番号のデコード      */
    DRVSEL(dvno);                                               /*  ドライブ・セレクト信号の選択        */
    retryc = (com_code & 0x1c) >> 2;                            /*  ﾘﾄﾗｲ･ｶｳﾝﾀのセット   */

    com_dat[0] = 0x0c | (0xe0 & com_code);                      /*  ﾘｰﾄﾞ･ﾃﾞﾘｰﾃｯﾄﾞ･ﾃﾞｰﾀ用データのセット  */

    if(com_prm->h == 0x00)                                      /*  ドライブとヘッド番号の設定  */
    {
        com_dat[1] = com_code & 0x03;                           /*  ヘッド番号＝０のときの設定  */
    }
    else
    {
        com_dat[1] = (com_code | 0x04) & 0x07;                  /*  ヘッド番号＝１のときの設定  */
    }

    --com_prm->trsize.dtsz;                                     /*  転送データ・サイズをデクリメント    */

    while(retryc >= 0)
    {
        madrss = com_prm->dt_seg;                               /*  転送アドレスの設定  */
        madrss = (madrss << 4) + com_prm->dt_off;
        i = 0;
        while(i <= 2)
        {
            adrss[i] = (unsigned int)(madrss >> (8 * i));
                                                                /*  転送アドレスを8ﾋﾞｯﾄごとに分解       */
            ++i;
        }

        outportb(DMODER, 0x47);                                 /*  DMAｺﾝﾄﾛｰﾗのﾓｰﾄﾞ･ﾚｼﾞｽﾀに転送方向をセット     */
```

(L101.C) p.11

```
outportb(DCLBFF, 0x00);                                 /*  DMAｺﾝﾄﾛｰﾗのｸﾘｱ･ﾊﾞｲﾄ･ﾎﾟｲﾝﾀFFをリセット          */
outportb(DADDR,adrss[0]);                               /*  DMAｺﾝﾄﾛｰﾗのｱﾄﾞﾚｽ･ﾚｼﾞｽﾀにｱﾄﾞﾚｽ(0-7ﾋﾞｯﾄ)をｾｯﾄ */
outportb(DADDR,adrss[1]);                               /*  DMAｺﾝﾄﾛｰﾗのｱﾄﾞﾚｽ･ﾚｼﾞｽﾀにｱﾄﾞﾚｽ(8-15ﾋﾞｯﾄ)をｾｯﾄ*/
outportb(DBNKR,adrss[2]);                               /*  DMAｺﾝﾄﾛｰﾗのﾁｬﾈﾙ3ﾊﾞﾝｸ･ﾚｼﾞｽﾀにｱﾄﾞﾚｽ(16-23ﾋﾞｯﾄ)*/
                                                        /*  をセット     */
outportb(DCOUNT,com_prm->trsize.int_hen.kai);
                                                        /*  DMAｺﾝﾄﾛｰﾗのｶｳﾝﾄ･ﾚｼﾞｽﾀにﾃﾞｰﾀ･ｻｲｽﾞをセット     */
outportb(DCOUNT,com_prm->trsize.int_hen.jyoui);
                                                        /*  DMAｺﾝﾄﾛｰﾗのｶｳﾝﾄ･ﾚｼﾞｽﾀにﾃﾞｰﾀ･ｻｲｽﾞをセット     */
outportb(DWRSMR,0x03);                                  /*  DMAｺﾝﾄﾛｰﾗのﾁｬﾈﾙ3のﾏｽｸを解除する      */
com_dat[2] = com_prm->c;                                /*  シリンダ番号の設定  */
com_dat[3] = com_prm->h;                                /*  ヘッド番号の設定     */
com_dat[4] = com_prm->r;                                /*  開始セクタ番号の設定         */
com_dat[5] = com_prm->n;                                /*  セクタあたりのバイト数の設定         */
com_dat[6] = com_prm->eot;                              /*  １トラックのセクタ数の設定  */
com_dat[7] = com_prm->gsl;                              /*  GAP3の読み飛ばしバイト数の設定       */
com_dat[8] = com_prm->dtl;                              /*  com_prm.n≠0なので値に意味はない     */
com_num = 0x09;
while(time_i != 1);                                     /*  モータの回転安定時間待ち     */
COMMAND(&com_dat[0], com_num);                          /*  ｺﾏﾝﾄﾞ･ﾓｼﾞｭｰﾙ呼びだし         */

while(int_s != 0x01);                                   /*  コマンド終了割り込み待ち     */
int_s = 0x00;
if(rslt_pt[0] >= 0x40)                                  /*  コマンドの終了結果の判断     */
{
        if((rslt_pt[0] & 0x08) == 0x08)
        {
                ret_st = 0x01;                          /*  ドライブのノット・レディ     */
        }
        else
        {
                switch(rslt_pt[1])
                {
                        case 0x01:
                                if((rslt_pt[2] & 0x01) == 0x01)
                                {
                                        ret_st = 0x08;                  /*  ＤＡＭ非検出         */
                                }
                                else
                                {
                                        ret_st = 0x02;                  /*  ＩＤＡＭ非検出       */
                                }
                                break;
                        case 0x04:
                                switch(rslt_pt[2])
                                {
                                        case 0x02:
                                                ret_st = 0x03;          /*  指定したシリンダがない       */
                                                break;
                                        case 0x10:
                                                ret_st = 0x04;          /*  指定したシリンダがない       */
                                                break;
                                        case 0x00:
                                                ret_st = 0x05;          /*  指定したセクタがない         */
                                                break;
                                        default:
                                                break;
                                }
                                break;
                        case 0x20:
                                if((rslt_pt[2] & 0x20) == 0x20)
                                {
                                        ret_st = 0x07;  /*  データ部のＣＲＣエラー       */
                                }
                                else
                                {
                                        ret_st = 0x06;  /*  ＩＤ部のＣＲＣエラー         */
                                }
                                break;
                        case 0x10:
```

保守／廃止

(L101.C) p.12

```
                                ret_st = 0x0a;          /*  オーバラン・エラー  */
                                break;
                        case 0x80:
                                ret_st = 0x0b;          /*  最終セクタを越えてアクセスしようとした      */
                                break;
                        default:
                                break;
                }
        }
}
else
{
        if(rslt_pt[2] < 0x40)
        {
                ret_st = 0x00;                          /*  コマンド正常終了    */
        }
        else
        {
                ret_st = 0x09;                          /*  ＤＤＡＭ付きデータをリードした      */
        }
}
ret_prm->c = rslt_pt[3];                                /*  コマンド終了時のﾘｻﾞﾙﾄ･ﾊﾟﾗﾒｰﾀのセット        */
ret_prm->h = rslt_pt[4];
ret_prm->r = rslt_pt[5];
ret_prm->n = rslt_pt[6];

switch(ret_st)                                          /*  リトライ実行の判定  */
{
        case 0x00:                                      /*  ﾘﾀｰﾝ･ｽﾃｰﾀｽが00H,01Hのとき   */
        case 0x01:
                retryc = 0;                             /*  リトライなし        */
                break;
        case 0x02:                                      /*  ﾘﾀｰﾝ･ｽﾃｰﾀｽが02H,05H-08Hのとき       */
        case 0x05:
        case 0x06:
        case 0x07:
        case 0x08:
                if(com_prm->c <= 50)                    /*  リードするシリンダの位置の判断(シークする   */
                                                        /*  方向の決定) */
                {
                        coml_code = com_dat[1] | 0x04;  /*  シリンダの位置が50シリンダ以下ならば        */
                }                                       /*  求心方向へシークする        */
                else
                {
                        coml_code = com_dat[1] | 0x03;  /*  シリンダの位置が51シリンダ以上ならば        */
                }                                       /*  遠心方向にシークする        */

                coml_prm.c = 20;                        /*  シークするシリンダ数の設定  */

                retl_st = RSEEK(coml_code, &coml_prm);
                                                        /*  ﾘﾗﾃｨﾌﾞ･ｼｰｸの実行    */

                if(retl_st != 0x00)                     /*  ﾘﾗﾃｨﾌﾞ･ｼｰｸの実行結果がエラーか？    */
                {
                        retryc = 0;
                        break;                          /*  リトライせずに終了する      */
                }
                coml_code = com_code & 0x03;            /*  リードするシリンダへ再シークする    */
                coml_prm.c = com_prm->c;
                retl_st = SEEK(coml_code, &coml_prm);
                                                        /*  シーク実行  */

                if(retl_st != 0x00)                     /*  シークの実行結果がエラーか？        */
                {
                        retryc = 0;                     /*  リトライせずに終了  */
                }
                break;
        default:
                break;
}
```

保守／廃止

(L101.C) p.13

```
            --retryc;                                           /*  ﾘﾄﾗｲ･ｶｳﾝﾀをﾃﾞｸﾘﾒﾝﾄしてリトライする  */
      }
      outportb(DWRSMR, 0x07);                                   /*  DMAｺﾝﾄﾛｰﾗのﾁｬﾈﾙ3をマスクする        */
      return ret_st;
}



            /*****************************************************/
            /*                                                   */
            /*  リードＩＤ処理      ＩＮ    : com_code           */
            /*                      ＯＵＴ  : ret_st, ret_prm    */
            /*                                                   */
            /*****************************************************/


unsigned char RDID(unsigned char com_code, struct retpra *ret_prm)
{
      unsigned char com_dat[2], ret_st, dvno;
      signed char com_num, retryc;

      dvno = com_code & 0x03;                                   /*  ドライブ番号のデコード      */
      DRVSEL(dvno);                                             /*  ドライブ・セレクト信号の選択        */
      retryc = (com_code &0x1c) >> 2;                           /*  ﾘﾄﾗｲ･ｶｳﾝﾀのセット   */
      com_dat[0] = 0x0a | (0x40 & com_code);                    /*  ﾘｰﾄﾞ･ID用データのセット     */
      while(retryc >= 0)
      {
            if((com_code & 0x20) == 0x20)                       /*  ドライブとヘッド番号の設定  */
            {
                  com_dat[1] = 0x04 | (0x03 & com_code);        /*  ヘッド番号＝１のときの設定  */
            }
            else
            {
                  com_dat[1] = 0x03 & com_code;                 /*  ヘッド番号＝０のときの設定  */
            }
            com_num = 0x02;
            while(time_i != 1);                                 /*  モータの回転安定時間待ち    */
            COMMAND(&com_dat[0], com_num);                      /*  ｺﾏﾝﾄﾞ･ﾓｼﾞｭｰﾙ呼びだし        */

            while(int_s != 0x01);                               /*  コマンド終了割り込み待ち    */
            int_s = 0x00;
            if(rslt_pt[0] < 0x40)                               /*  コマンドの終了結果の判断    */
            {
                  ret_st = 0x00;                                /*  コマンド正常終了    */
            }
            else
            {
                  if((rslt_pt[0] & 0x08) == 0x08)
                  {
                        ret_st = 0x01;                          /*  ドライブのノット・レディ    */
                  }
                  if((rslt_pt[1] & 0x01) == 0x01)
                  {
                        ret_st = 0x02;                          /*  ＩＤＡＭ非検出      */
                  }
                  if((rslt_pt[1] & 0x04) == 0x04)
                  {
                        ret_st = 0x05;                          /*  指定したセクタがない        */
                  }
            }
            ret_prm->c = rslt_pt[3];                            /*  コマンド終了時のﾘｻﾞﾙﾄ･ﾊﾟﾗﾒｰﾀのセット        */
            ret_prm->h = rslt_pt[4];
            ret_prm->r = rslt_pt[5];
            ret_prm->n = rslt_pt[6];
            if((ret_st == 0x00) || (ret_st == 0x01))            /*  リトライ実行の判断  */
            {                                                   /*  ﾘﾀｰﾝ･ｽﾃｰﾀｽが00H,01Hのとき   */
                  break;                                        /*  リトライなし        */
            }
            --retryc;                                           /*  ﾘﾀｰﾝ･ｽﾃｰﾀｽがそれ以外のときはﾘﾄﾗｲ･ｶｳﾝﾀを     */
                                                                /*  ﾃﾞｸﾘﾒﾝﾄしてリトライする     */
```

(L101.C) p.14

```
        }
        return ret_st;
}


                /*************************************************************************/
                /*                                                                       */
                /*  リード・ダイアグノスティック処理      ＩＮ    : com_code, com_prm      */
                /*                                        ＯＵＴ  : ret_st, ret_prm        */
                /*                                                                       */
                /*************************************************************************/

unsigned char RDDAG(unsigned char com_code, struct compra *com_prm, struct retpra *ret_prm)
{
        unsigned char com_dat[9], adrss[3], coml_code, ret_st, retl_st, dvno;
        signed char com_num, retryc, i;
        unsigned long madrss;
        struct compra coml_prm;

        dvno = com_code & 0x03;                                 /*  ドライブ番号のデコード      */
        DRVSEL(dvno);                                           /*  ドライブ・セレクト信号の選択        */
        retryc = (com_code & 0x1c) >> 2;                        /*  ﾘﾄﾗｲ･ｶｳﾝﾀのセット   */
        com_dat[0] = 0x02 | (0x40 & com_code);                  /*  ﾘｰﾄﾞ･ｱ･ﾄﾗｯｸ用データのセット */
        if(com_prm->h == 0x00)                                  /*  ドライブとヘッド番号の設定  */
        {
                com_dat[1] = com_code & 0x03;                   /*  ヘッド番号＝０のときの設定  */
        }
        else
        {
                com_dat[1] = (com_code | 0x04) & 0x07;          /*  ヘッド番号＝１のときの設定  */
        }
        --com_prm->trsize.dtsz;                                 /*  転送データ・サイズをデクリメント    */
        while(retryc >= 0)
        {
                madrss = com_prm->dt_seg;                       /*  転送アドレスの設定  */
                madrss = (madrss << 4) + com_prm->dt_off;
                i = 0;
                while(i <= 2)
                {
                        adrss[i] = (unsigned char)(madrss >> (8 * i));
                                                                /* 転送アドレスを8ﾋﾞｯﾄごとに分解        */
                        ++i;
                }

                outportb(DMODER, 0x47);                         /*  DMAｺﾝﾄﾛｰﾗのﾓｰﾄﾞ･ﾚｼﾞｽﾀに転送方向をセット     */
                outportb(DCLBFF, 0x00);                         /*  DMAｺﾝﾄﾛｰﾗのｸﾘｱ･ﾊﾞｲﾄ･ﾎﾟｲﾝﾀFFをリセット       */
                outportb(DADDR, adrss[0]);                      /*  DMAｺﾝﾄﾛｰﾗのｱﾄﾞﾚｽ･ﾚｼﾞｽﾀにｱﾄﾞﾚｽ(0-7ﾋﾞｯﾄ)をｾｯﾄ */
                outportb(DADDR, adrss[1]);                      /*  DMAｺﾝﾄﾛｰﾗのｱﾄﾞﾚｽ･ﾚｼﾞｽﾀにｱﾄﾞﾚｽ(8-15ﾋﾞｯﾄ)をｾｯﾄ*/
                outportb(DBNKR, adrss[2]);                      /*  DMAｺﾝﾄﾛｰﾗのﾁｬﾈﾙ3ﾊﾞﾝｸ･ﾚｼﾞｽﾀにｱﾄﾞﾚｽ(16-23ﾋﾞｯﾄ)*/
                                                                /*  をセット     */
                outportb(DCOUNT, com_prm->trsize.int_hen.kai);
                                                                /*  DMAｺﾝﾄﾛｰﾗのｶｳﾝﾄ･ﾚｼﾞｽﾀにﾃﾞｰﾀ･ｻｲｽﾞをセット     */
                outportb(DCOUNT, com_prm->trsize.int_hen.jyoui);
                                                                /*  DMAｺﾝﾄﾛｰﾗのｶｳﾝﾄ･ﾚｼﾞｽﾀにﾃﾞｰﾀ･ｻｲｽﾞをセット     */
                outportb(DWRSMR, 0x03);                         /*  DMAｺﾝﾄﾛｰﾗのﾁｬﾈﾙ3のﾏｽｸを解除する      */
                com_dat[2] = com_prm->c;                        /*  シリンダ番号の設定  */
                com_dat[3] = com_prm->h;                        /*  ヘッド番号の設定    */
                com_dat[4] = com_prm->r;                        /*  開始セクタ番号の設定        */
                com_dat[5] = com_prm->n;                        /*  セクタあたりのバイト数の設定        */
                com_dat[6] = com_prm->eot;                      /*  １トラックのセクタ数の設定  */
                com_dat[7] = com_prm->gsl;                      /*  GAP3の読み飛ばしバイト数の設定      */
                com_dat[8] = com_prm->dtl;                      /*  com_prm.n≠0なので値に意味はない    */
                com_num = 0x09;
                while(time_i != 1);                             /*  モータの回転安定時間待ち    */
                COMMAND(&com_dat[0], com_num);                  /*  ｺﾏﾝﾄﾞ･ﾓｼﾞｭｰﾙ呼びだし        */

                while(int_s != 0x01);                           /*  コマンド終了割り込み待ち    */
                int_s = 0x00;
                if(rslt_pt[0] >= 0x40)                          /*  コマンドの終了結果の判断    */
                {
```

(L101.C) p.15

```
                switch(rslt_pt[1])
                {
                        case 0x01:
                                if((rslt_pt[2] & 0x01) == 0x01)
                                {
                                        ret_st = 0x08;          /*  ＤＡＭ非検出        */
                                }
                                else
                                {
                                        ret_st = 0x02;          /*  ＩＤＡＭ非検出      */
                                }
                                break;
                        case 0x00:
                                ret_st = 0x01;                  /*  ドライブのノット・レディ    */
                                break;
                        case 0x10:
                                ret_st = 0x0a;                  /*  オーバラン・エラー  */
                                break;
                        case 0x80:
                                ret_st = 0x0b;                  /*  最終セクタを越えてアクセスしようとした      */
                                break;
                        default:
                                break;
                }
        }
        else
        {
                switch(rslt_pt[1])
                {
                        case 0x00:
                                if((rslt_pt[2] & 0x40) == 0x40)
                                {
                                        ret_st = 0x09;          /*  ＤＤＡＭ付きﾃﾞｰﾀをﾘｰﾄﾞした  */
                                }
                                else
                                {
                                        ret_st = 0x00;          /*  コマンド正常終了    */
                                }
                                break;
                        case 0x20:
                                if((rslt_pt[2] & 0x20) == 0x20)
                                {
                                        ret_st = 0x07;          /*  データ部のＣＲＣエラー      */
                                }
                                else
                                {
                                        ret_st = 0x06;          /*  ＩＤ部のＣＲＣエラー        */
                                }
                                break;
                        case 0x04:
                                ret_st = 0x05;                  /*  指定したシリンダがない      */
                                break;
                        default:
                                break;
                }
        }
        ret_prm->c = rslt_pt[3];                                /*  コマンド終了時のﾘｻﾞﾙﾄ･ﾊﾟﾗﾒｰﾀのｾｯﾄ  */
        ret_prm->h = rslt_pt[4];
        ret_prm->r = rslt_pt[5];
        ret_prm->n = rslt_pt[6];

        switch(ret_st)                                          /*  リトライ実行の判定  */
        {
                case 0x00:                                      /*  ﾘﾀｰﾝ･ｽﾃｰﾀｽが00H,01Hのとき   */
                case 0x01:
                        retryc = 0;                             /*  リトライなし        */
                        break;
                case 0x02:                                      /*  ﾘﾀｰﾝ･ｽﾃｰﾀｽが02H,05H-08Hのとき       */
                case 0x05:
                case 0x06:
```

(L101.C) p.16

```
                    case 0x07:
                    case 0x08:
                            if(com_prm->c <= 50)                    /*  リードするシリンダの位置の判断(シークする    */
                                                                    /*  方向の決定) */
                            {
                                    com1_code = com_dat[1] | 0x04;  /*  シリンダの位置が50シリンダ以下ならば          */
                            }                                       /*  求心方向へシークする          */
                            else
                            {
                                    com1_code = com_dat[1] & 0x03;  /*  シリンダの位置が51シリンダ以上ならば          */
                            }                                       /*  遠心方向にシークする          */

                            com1_prm.c = 20;                        /*  シークするシリンダ数の設定  */

                            ret1_st = RSEEK(com1_code, &com1_prm);
                                                                    /*  ﾘﾗﾃｨﾌﾞ･ｼｰｸの実行     */

                            if(ret1_st != 0x00)                     /*  ﾘﾗﾃｨﾌﾞ･ｼｰｸの実行結果がエラーか？     */
                            {
                                    retryc = 0;
                                    break;                          /*  リトライせずに終了する        */
                            }
                            com1_code = com_code & 0x03;            /*  リードするシリンダへ再シークする     */
                            com1_prm.c = com_prm->c;
                            ret1_st = SEEK(com1_code, &com1_prm);
                                                                    /*  シーク実行  */

                            if(ret1_st != 0x00)                     /*  シークの実行結果がエラーか？          */
                            {
                                    retryc = 0;                     /*  リトライせずに終了  */
                            }
                            break;
                    default:
                            break;
            }
            --retryc;                                               /*  ﾘﾄﾗｲ･ｶｳﾝﾀをﾃﾞｸﾘﾒﾝﾄしてリトライする  */
    }
    outportb(DWRSMR, 0x07);                                         /*  DMAｺﾝﾄﾛｰﾗのﾁｬﾈﾙ3をマスクする          */
    return ret_st;
}


            /*********************************************************/
            /*                                                       */
            /*  ライト・データ処理   ＩＮ    : com_code, com_prm      */
            /*                       ＯＵＴ  : ret_st, ret_prm        */
            /*                                                       */
            /*********************************************************/


unsigned char WRDAT(unsigned char com_code, struct compra *com_prm, struct retpra *ret_prm)
{
    unsigned char com_dat[9], adrss[3], com1_code, ret_st, ret1_st, dvno;
    signed char com_num, retryc, i;
    unsigned long madrss;
    struct compra com1_prm;


    dvno = com_code & 0x03;                                         /*  ドライブ番号のデコード       */
    DRVSEL(dvno);                                                   /*  ドライブ・セレクト信号の選択          */
    retryc = (com_code & 0x1c) >> 2;                                /*  ﾘﾄﾗｲ･ｶｳﾝﾀのセット    */
    com_dat[0] = 0x05 | (0xc0 & com_code);                          /*  ﾗｲﾄ･ﾃﾞｰﾀ用データのセット     */
    if(com_prm->h == 0x00)                                          /*  ドライブとヘッド番号の設定  */
    {
            com_dat[1] = com_code & 0x03;                           /*  ヘッド番号＝０のときの設定  */
    }
    else
    {
            com_dat[1] = (com_code | 0x04) & 0x07;                  /*  ヘッド番号＝１のときの設定  */
    }
```

保守／廃止

(L101.C) p.17

```
--com_prm->trsize.dtsz;                                    /*  転送データ・サイズをデクリメント    */
while(retryc >= 0)
{
        madrss = com_prm->dt_seg;                          /*  転送アドレスの設定  */
        madrss = (madrss << 4) + com_prm->dt_off;
        i = 0;
        while(i <= 2)
        {
                adrss[i] = (unsigned char)(madrss >>(8 * i));
                                                           /*  転送アドレスを8ﾋﾞｯﾄごとに分解       */
                ++i;
        }

        outportb(DMODER, 0x4b);                            /*  DMAｺﾝﾄﾛｰﾗのﾓｰﾄﾞ･ﾚｼﾞｽﾀに転送方向をセット     */
        outportb(DCLBFF, 0x00);                            /*  DMAｺﾝﾄﾛｰﾗのｸﾘｱ･ﾊﾞｲﾄ･ﾎﾟｲﾝﾀFFをリセット       */
        outportb(DADDR,adrss[0]);                          /*  DMAｺﾝﾄﾛｰﾗのｱﾄﾞﾚｽ･ﾚｼﾞｽﾀにｱﾄﾞﾚｽ(0-7ﾋﾞｯﾄ)をｾｯﾄ */
        outportb(DADDR,adrss[1]);                          /*  DMAｺﾝﾄﾛｰﾗのｱﾄﾞﾚｽ･ﾚｼﾞｽﾀにｱﾄﾞﾚｽ(8-15ﾋﾞｯﾄ)をｾｯﾄ*/
        outportb(DBNKR,adrss[2]);                          /*  DMAｺﾝﾄﾛｰﾗのﾁｬﾈﾙ3ﾊﾞﾝｸ･ﾚｼﾞｽﾀにｱﾄﾞﾚｽ(16-23ﾋﾞｯﾄ)*/
                                                           /*  をセット    */
        outportb(DCOUNT,com_prm->trsize.int_hen.kai);
                                                           /*  DMAｺﾝﾄﾛｰﾗのｶｳﾝﾄ･ﾚｼﾞｽﾀにﾃﾞｰﾀ･ｻｲｽﾞをセット    */

        outportb(DCOUNT,com_prm->trsize.int_hen.jyoui);
                                                           /*  DMAｺﾝﾄﾛｰﾗのｶｳﾝﾄ･ﾚｼﾞｽﾀにﾃﾞｰﾀ･ｻｲｽﾞをセット    */
        outportb(DWRSMR,0x03);                             /*  DMAｺﾝﾄﾛｰﾗのﾁｬﾈﾙ3のマスクを解除する  */
        com_dat[2] = com_prm->c;                           /*  シリンダ番号の設定  */
        com_dat[3] = com_prm->h;                           /*  ヘッド番号の設定    */
        com_dat[4] = com_prm->r;                           /*  開始セクタ番号の設定        */
        com_dat[5] = com_prm->n;                           /*  セクタあたりのバイト数の設定        */
        com_dat[6] = com_prm->eot;                         /*  １トラックあたりのセクタ数の設定    */
        com_dat[7] = com_prm->gsl;                         /*  GAP3の読み飛ばしバイト数の設定      */
        com_dat[8] = com_prm->dtl;                         /*  com_prm.n≠0なので値に意味はない    */
        com_num = 0x09;
        while(time_i != 1);                                /*  モータの回転安定時間待ち    */
        COMMAND(&com_dat[0], com_num);                     /*  ｺﾏﾝﾄﾞ･ﾓｼﾞｭｰﾙ呼びだし        */
        while(int_s != 0x01);                              /*  コマンド終了割り込み待ち    */
        int_s = 0x00;
        if(rslt_pt[0] >= 0x40)                             /*  コマンドの終了結果の判断    */
        {
                if((rslt_pt[0] & 0x08) == 0x08)
                {
                        ret_st = 0x01;                     /*  ドライブのノット・レディ    */
                }
                else
                {
                        switch(rslt_pt[1])
                        {
                                case 0x01:
                                        ret_st = 0x02;                  /*  ＩＤＡＭ非検出      */
                                        break;
                                case 0x02:
                                        ret_st = 0x0c;                  /*  ライト・プロテクト  */
                                        break;
                                case 0x04:
                                        switch(rslt_pt[2])
                                        {
                                                case 0x02:
                                                        ret_st = 0x03;  /*  指定したシリンダがない      */
                                                        break;
                                                case 0x10:
                                                        ret_st = 0x04;  /*  指定したシリンダがない      */
                                                        break;
                                                case 0x00:
                                                        ret_st = 0x05;  /*  指定したセクタがない        */
                                                        break;
                                                deault:
                                                        break;
                                        }
                                        break;
                                case 0x20:
```

保守／廃止

(L101.C) p.18

```
                                        ret_st = 0x06;          /*  ＩＤ部のＣＲＣエラー        */
                                        break;
                                case 0x10:
                                        ret_st = 0x0a;          /*  オーバラン・エラー  */
                                        break;
                                case 0x80:
                                        ret_st = 0x0b;          /*  最終セクタを越えてアクセスしようとした      */
                                        break;
                                default:
                                        break;
                        }
                }
        }
        else
        {
                ret_st = 0x00;                                  /*  コマンドの正常終了  */
        }
        ret_prm->c = rslt_pt[3];                                /*  コマンド終了時のﾘｻﾞﾙﾄ･ﾊﾟﾗﾒｰﾀのセット        */
        ret_prm->h = rslt_pt[4];
        ret_prm->r = rslt_pt[5];
        ret_prm->n = rslt_pt[6];

        switch(ret_st)                                          /*  リトライ実行の判定  */
        {
                case 0x00:                                      /*  ﾘﾀｰﾝ･ｽﾃｰﾀｽが00H,01H,03H,04H,0CHのとき      */
                case 0x01:
                case 0x03:
                case 0x04:
                case 0x0c:
                        retryc = 0;                             /*  リトライなし        */
                        break;
                case 0x02:                                      /*  ﾘﾀｰﾝ･ｽﾃｰﾀｽが02H,05H,06Hのとき       */
                case 0x05:
                case 0x06:
                        if(com_prm->c <= 50)                    /*  リードするシリンダの位置の判断(シークする  */
                                                                /*  方向の判定) */
                        {
                                com1_code = com_dat[1] | 0x04;  /*  シリンダの位置が50シリンダ以下ならば        */
                        }                                       /*  求心方向へシークする        */
                        else
                        {
                                com1_code = com_dat[1] & 0x03;  /*  シリンダの位置が51シリンダ以上ならば        */
                        }                                       /*  遠心方向にシークする        */

                        com1_prm.c = 20;                        /*  シークするシリンダ数の設定  */

                        ret1_st = RSEEK(com1_code, &com1_prm);
                                                                /*  ﾘﾗﾃｨﾌﾞ･ｼｰｸの実行    */

                        if(ret1_st != 0x00)                     /*  ﾘﾗﾃｨﾌﾞ･ｼｰｸの実行結果がエラーか？    */
                        {
                                retryc = 0;
                                break;                          /*  リトライせずに終了  */
                        }
                        com1_code = com_code & 0x03;            /*  リードするシリンダへ再シークする    */
                        com1_prm.c = com_prm->c;
                        ret1_st = SEEK(com1_code, &com1_prm);
                                                                /*  シーク実行  */

                        if(ret1_st != 0x00)                     /*  シークの実行結果がエラーか？        */
                        {
                                retryc = 0;                     /*  リトライせずに終了  */
                        }
                        break;
                default:
                        break;
        }
        --retryc;                                               /*  ﾘﾄﾗｲ･ｶｳﾝﾀをﾃﾞｸﾘﾒﾝﾄしてリトライする  */
}
outportb(DWRSMR, 0x07);                                         /*  DMAｺﾝﾄﾛｰﾗのﾁｬﾈﾙ3をマスクする        */
```

保守／廃止

(L101.C) p.19

```
    return ret_st;
}



        /*************************************************************************/
        /*                                                                       */
        /*  ライト・デリーテッド・データ処理      ＩＮ    : com_code, com_prm      */
        /*                                        ＯＵＴ  : ret_st, ret_prm        */
        /*                                                                       */
        /*************************************************************************/


unsigned char WRDDAT(unsigned char com_code, struct compra *com_prm, struct retpra *ret_prm)
{
    unsigned char com_dat[9], adrss[3], com1_code, ret_st, ret1_st, dvno;
    signed char com_num, retryc, i;
    unsigned long madrss;
    struct compra com1_prm;

    dvno = com_code & 0x03;                                  /*  ドライブ番号のデコード      */
    DRVSEL(dvno);                                            /*  ドライブ・セレクト信号の選択        */
    retryc = (com_code & 0x1c) >> 2;                         /*  ﾘﾄﾗｲ･ｶｳﾝﾀのセット    */
    com_dat[0] = 0x09 | (0xc0 & com_code);                   /*  ﾗｲﾄ･ﾃﾞｰﾀ用データのセット    */
    if(com_prm->h == 0x00)                                   /*  ドライブとヘッド番号の設定  */
    {
        com_dat[1] = com_code & 0x03;                        /*  ヘッド番号＝０のときの設定  */
    }
    else
    {
        com_dat[1] = (com_code | 0x04) & 0x07;               /*  ヘッド番号＝１のときの設定  */
    }
    --com_prm->trsize.dtsz;                                  /*  転送データ・サイズをデクリメント    */
    while(retryc >= 0)
    {
        madrss = com_prm->dt_seg;                            /*  転送アドレスの設定  */
        madrss = (madrss << 4) + com_prm->dt_off;
        i = 0;
        while(i <= 2)
        {
            adrss[i] = (unsigned char)(madrss >>(8 * i));
                                                             /*  転送アドレスを8ﾋﾞｯﾄごとに分解       */
            ++i;
        }

        outportb(DMODER, 0x4b);                              /*  DMAｺﾝﾄﾛｰﾗのﾓｰﾄﾞ･ﾚｼﾞｽﾀに転送方向をセット      */
        outportb(DCLBFF, 0x00);                              /*  DMAｺﾝﾄﾛｰﾗのｸﾘｱ･ﾊﾞｲﾄ･ﾎﾟｲﾝﾀFFをリセット        */
        outportb(DADDR,adrss[0]);                            /*  DMAｺﾝﾄﾛｰﾗのｱﾄﾞﾚｽ･ﾚｼﾞｽﾀにｱﾄﾞﾚｽ(0-7ﾋﾞｯﾄ)をｾｯﾄ */
        outportb(DADDR,adrss[1]);                            /*  DMAｺﾝﾄﾛｰﾗのｱﾄﾞﾚｽ･ﾚｼﾞｽﾀにｱﾄﾞﾚｽ(8-15ﾋﾞｯﾄ)をｾｯﾄ*/
        outportb(DBNKR,adrss[2]);                            /*  DMAｺﾝﾄﾛｰﾗのﾁｬﾈﾙ3ﾊﾞﾝｸ･ﾚｼﾞｽﾀにｱﾄﾞﾚｽ(16-23ﾋﾞｯﾄ)*/
                                                             /*  をセット    */
        outportb(DCOUNT,com_prm->trsize.int_hen.kai);
                                                             /*  DMAｺﾝﾄﾛｰﾗのｶｳﾝﾄ･ﾚｼﾞｽﾀにﾃﾞｰﾀ･ｻｲｽﾞをセット    */

        outportb(DCOUNT,com_prm->trsize.int_hen.jyoui);
                                                             /*  DMAｺﾝﾄﾛｰﾗのｶｳﾝﾄ･ﾚｼﾞｽﾀにﾃﾞｰﾀ･ｻｲｽﾞをセット    */
        outportb(DWRSMR,0x03);                               /*  DMAｺﾝﾄﾛｰﾗのﾁｬﾈﾙ3のマスクを解除する  */
        com_dat[2] = com_prm->c;                             /*  シリンダ番号の設定  */
        com_dat[3] = com_prm->h;                             /*  ヘッド番号の設定    */
        com_dat[4] = com_prm->r;                             /*  開始セクタ番号の設定        */
        com_dat[5] = com_prm->n;                             /*  セクタあたりのバイト数の設定        */
        com_dat[6] = com_prm->eot;                           /*  １トラックあたりのセクタ数の設定    */
        com_dat[7] = com_prm->gsl;                           /*  GAP3の読み飛ばしバイト数の設定      */
        com_dat[8] = com_prm->dtl;                           /*  com_prm.n≠0なので値に意味はない    */
        com_num = 0x09;
        while(time_i != 1);                                  /*  モータの回転安定時間待ち    */
        COMMAND(&com_dat[0], com_num);                       /*  ｺﾏﾝﾄﾞ･ﾓｼﾞｭｰﾙ呼びだし        */
        while(int_s != 0x01);                                /*  コマンド終了割り込み待ち    */
        int_s = 0x00;
        if(rslt_pt[0] >= 0x40)                               /*  コマンドの終了結果の判断    */
```

(L101.C) p.20

```
{
        if((rslt_pt[0] & 0x08) == 0x08)
        {
                ret_st = 0x01;                          /*  ドライブのノット・レディ    */
        }
        else
        {
                switch(rslt_pt[1])
                {
                        case 0x01:
                                ret_st = 0x02;                          /*  ＩＤＡＭ非検出      */
                                break;
                        case 0x02:
                                ret_st = 0x0c;                          /*  ライト・プロテクト  */
                                break;
                        case 0x04:
                                switch(rslt_pt[2])
                                {
                                        case 0x02:
                                                ret_st = 0x03;          /*  指定したシリンダがない      */
                                                break;
                                        case 0x10:
                                                ret_st = 0x04;          /*  指定したシリンダがない      */
                                                break;
                                        case 0x00:
                                                ret_st = 0x05;          /*  指定したセクタがない        */
                                                break;
                                        deault:
                                                break;
                                }
                                break;
                        case 0x20:
                                ret_st = 0x06;          /*  ＩＤ部のＣＲＣエラー        */
                                break;
                        case 0x10:
                                ret_st = 0x0a;          /*  オーバラン・エラー  */
                                break;
                        case 0x80:
                                ret_st = 0x0b;          /*  最終セクタを越えてアクセスしようとした      */
                                break;
                        default:
                                break;
                }
        }
}
else
{
        ret_st = 0x00;                                  /*  コマンドの正常終了  */
}
ret_prm->c = rslt_pt[3];                                /*  コマンド終了時のﾘｻﾞﾙﾄ･ﾊﾟﾗﾒｰﾀのセット        */
ret_prm->h = rslt_pt[4];
ret_prm->r = rslt_pt[5];
ret_prm->n = rslt_pt[6];

switch(ret_st)                                          /*  リトライ実行の判定  */
{
        case 0x00:                                      /*  ﾘﾀｰﾝ･ｽﾃｰﾀｽが00H,01H,03H,04H,0CHのとき      */
        case 0x01:
        case 0x03:
        case 0x04:
        case 0x0c:
                retryc = 0;                             /*  リトライなし        */
                break;
        case 0x02:                                      /*  ﾘﾀｰﾝ･ｽﾃｰﾀｽが02H,05H,06Hのとき       */
        case 0x05:
        case 0x06:
                if(com_prm->c <= 50)                    /*  リードするシリンダの位置の判断(シークする  */
                                                        /*  方向の判定) */
                {
                        coml_code = com_dat[1] | 0x04;  /*  シリンダの位置が50シリンダ以下ならば        */
```

保守／廃止

(L101.C) p.21

```
                        }                                       /*  求心方向へシークする        */
                        else
                        {
                                com1_code = com_dat[1] & 0x03;  /*  シリンダの位置が51シリンダ以上ならば        */
                        }                                       /*  遠心方向にシークする        */

                        com1_prm.c = 20;                        /*  シークするシリンダ数の設定  */

                        ret1_st = RSEEK(com1_code, &com1_prm);
                                                                /*  ﾘﾗﾃｨﾌﾞ･ｼｰｸの実行    */

                        if(ret1_st != 0x00)                     /*  ﾘﾗﾃｨﾌﾞ･ｼｰｸの実行結果がエラーか？    */
                        {
                                retryc = 0;
                                break;                          /*  リトライせずに終了  */
                        }
                        com1_code = com_code & 0x03;            /*  リードするシリンダへ再シークする    */
                        com1_prm.c = com_prm->c;
                        ret1_st = SEEK(com1_code, &com1_prm);
                                                                /*  シーク実行  */

                        if(ret1_st != 0x00)                     /*  シークの実行結果がエラーか？        */
                        {
                                retryc = 0;                     /*  リトライせずに終了  */
                        }
                        break;
                default:
                        break;
        }
        --retryc;                                               /*  ﾘﾄﾗｲ･ｶｳﾝﾀをﾃﾞｸﾘﾒﾝﾄしてリトライする  */
    }
    outportb(DWRSMR, 0x07);                                     /*  DMAｺﾝﾄﾛｰﾗのﾁｬﾈﾙ3をマスクする        */
    return ret_st;
}



        /*****************************************************************/
        /*                                                               */
        /*  トラック・フォーマット処理   ＩＮ    : com_code, com_prm      */
        /*                               ＯＵＴ  : ret_st, ret_prm        */
        /*                                                               */
        /*****************************************************************/


unsigned char TRFORMAT(unsigned char com_code, struct compra *com_prm, struct retpra *ret_prm)
{
    unsigned char com_dat[9], adrss[3], ret_st, dvno;
    signed char com_num, retryc, i;
    unsigned int dt_seg, dt_off;
    unsigned long madrss;
    struct form fmtdat[36];

    com_dat[0] = 0x0d | (0x40 & com_code);                      /*  ﾄﾗｯｸ･ﾌｫｰﾏｯﾄ用データのセット */

    retryc = (com_code & 0x1c) >> 2;                            /*  ﾘﾄﾗｲ･ｶｳﾝﾀのセット    */

    dvno = com_code & 0x03;                                     /*  ドライブ番号のデコード      */
    DRVSEL(dvno);                                               /*  ドライブ・セレクト信号の選択        */

    if((com_code & 0x20) == 0x00)                               /*  ドライブとヘッド番号の設定  */
    {
        com_dat[1] = com_code & 0x03;                           /*  ヘッド番号＝０のときの設定  */
    }
    else
    {
        com_dat[1] = (com_code | 0x04) & 0x07;                  /*  ヘッド番号＝１のときの設定  */
    }
    while(retryc >= 0)
    {
```

保守／廃止

(L101.C) p.22

```
for(i = 1; i <= com_prm->dtl; i++)                    /*  フォーマットに使用するID部のデータのセット  */
{
        fmtdat[i - 1].c = com_prm->c;
        if((com_code & 0x20) == 0x00)                 /*  ヘッド番号の設定     */
        {
                fmtdat[i - 1].h = 0;                  /*  ヘッド番号＝0のときの設定   */
        }
        else
        {
                fmtdat[i - 1].h = 1;                  /*  ヘッド番号＝１のときの設定  */
        }
        fmtdat[i - 1].r = i;
        fmtdat[i - 1].n = com_prm->n;
}
dt_off = FP_OFF(&fmtdat);                             /*  フォーマット用データのアドレスの取得        */
                                                      /*  (ｵﾌｾｯﾄ･ｱﾄﾞﾚｽ)       */
dt_seg = FP_SEG(&fmtdat);                             /*  フォーマット用データのアドレスの取得        */
                                                      /*  (ｾｸﾞﾒﾝﾄ･ｱﾄﾞﾚｽ)       */
madrss = dt_seg;                                      /*  アドレスを物理アドレスへ変換        */
madrss = (madrss << 4) + dt_off;
i = 0;
while(i <= 2)
{
        adrss[i] = (unsigned char)(madrss >> (8 * i));
                                                      /*  転送アドレスを8ﾋﾞｯﾄごとに分解       */
        ++i;
}

outportb(DMODER, 0x4b);                               /*  DMAｺﾝﾄﾛｰﾗのﾓｰﾄﾞ･ﾚｼﾞｽﾀに転送方向をセット      */
outportb(DCLBFF, 0x00);                               /*  DMAｺﾝﾄﾛｰﾗのｸﾘｱ･ﾊﾞｲﾄ･ﾎﾟｲﾝﾀFFをリセット        */
outportb(DADDR, adrss[0]);                            /*  DMAｺﾝﾄﾛｰﾗのｱﾄﾞﾚｽ･ﾚｼﾞｽﾀにｱﾄﾞﾚｽ(0-7ﾋﾞｯﾄ)をｾｯﾄ*/
outportb(DADDR, adrss[1]);                            /*  DMAｺﾝﾄﾛｰﾗのｱﾄﾞﾚｽ･ﾚｼﾞｽﾀにｱﾄﾞﾚｽ(8-15ﾋﾞｯﾄ)をｾｯﾄ*/
outportb(DBNKR, adrss[2]);                            /*  DMAｺﾝﾄﾛｰﾗのｱﾄﾞﾚｽ･ﾚｼﾞｽﾀにｱﾄﾞﾚｽ(16-23ﾋﾞｯﾄ)
                                                      /*  をセット     */
outportb(DCOUNT, (com_prm->dtl * 4 - 1));             /*  DMAｺﾝﾄﾛｰﾗのｶｳﾝﾄ･ﾚｼﾞｽﾀに転送バイト数をセット */
outportb(DCOUNT, 0);                                  /*  DMAｺﾝﾄﾛｰﾗのｶｳﾝﾄ･ﾚｼﾞｽﾀに転送バイト数をセット */
outportb(DWRSMR, 0x03);                               /*  DMAｺﾝﾄﾛｰﾗのﾁｬﾈﾙ3のﾏｽｸを解除する       */
com_dat[2] = com_prm->n;                              /*  シリンダ番号の設定  */
com_dat[3] = com_prm->dtl;                            /*  １トラックあたりのセクタ数の設定     */
com_dat[4] = com_prm->gsl;                            /*  GAP3の書き込みバイト数       */
com_dat[5] = com_prm->d;                              /*  データ領域に書き込むデータの設定     */
com_num = 0x06;
while(time_i != 1);                                   /*  モータの回転安定時間待ち     */
COMMAND(&com_dat[0], com_num);                        /*  ｺﾏﾝﾄﾞ･ﾓｼﾞｭｰﾙ呼びだし          */
while(int_s != 0x01);                                 /*  コマンド終了割り込み待ち     */
int_s = 0x00;
if(rslt_pt[0] >= 0x40)                                /*  コマンドの終了結果の判断     */
{
        if((rslt_pt[0] & 0x08) == 0x08)
        {
                ret_st = 0x01;                        /*  ドライブのノット・レディ     */
        }
        else
        {
                if((rslt_pt[1] & 0x10) == 0x10)
                {
                        ret_st = 0x0a;                /*  オーバラン・エラー   */
                }
                else
                {
                        ret_st = 0x0c;                /*  ライト・プロテクト   */
                }
        }
}
else
{
        ret_st = 0x00;                                /*  コマンドの正常終了   */
}
ret_prm->c = rslt_pt[3];                              /*  コマンド終了時のﾘｻﾞﾙﾄ･ﾊﾟﾗﾒｰﾀのセット   */
ret_prm->h = rslt_pt[4];
```

保守／廃止

(L101.C) p.23

```
            ret_prm->r = rslt_pt[5];
            ret_prm->n = rslt_pt[6];
            if(ret_st == 0x0a)                                  /*  リターン・ステータスが０ＡＨのとき  */
            {
                  --retryc;                                     /*  ﾘﾄﾗｲ･ｶｳﾝﾀをデクリメントしてリトライ */
            }
            else
            {
                  break;                                        /*  そのほかはリトライせずに終了        */
            }
      }
      outportb(DWRSMR, 0x07);                                   /*  DMAｺﾝﾄﾛｰﾗのﾁｬﾈﾙ3をマスクする        */
      return ret_st;
}


            /*****************************************************/
            /*                                                   */
            /*  ベリファイ処理       ＩＮ    : com_code, com_prm   */
            /*                       ＯＵＴ  : ret_st, ret_prm     */
            /*                                                   */
            /*****************************************************/


unsigned char VERIFY(unsigned char com_code, struct compra *com_prm, struct retpra *ret_prm)
{
      unsigned char com_dat[9], ret_st, dvno;
      char com_num;

      dvno = com_code & 0x03;                                   /*  ドライブ番号のデコード      */
      DRVSEL(dvno);                                             /*  ドライブ・セレクト信号の選択        */

      com_dat[0] = 0x16 | (0xe0 & com_code);                    /*  ﾍﾞﾘﾌｧｲ用データのセット      */

      if(com_prm->h == 0x00)                                    /*  ドライブとヘッド番号の設定  */
      {
            com_dat[1] = com_code & 0x03;                       /*  ヘッド番号＝０のときの設定  */
      }
      else
      {
            com_dat[1] = (com_code | 0x04) & 0x07;              /*  ヘッド番号＝１のときの設定  */
      }
      if((com_code & 0x04) == 0x04)                             /*  ＥＣビットの設定    */
      {
            com_dat[1] = com_dat[1] | 0x80;                     /*  ＥＣビット＝１のときの設定  */
      }
      else
      {
            com_dat[1] = com_dat[1] & 0x7f;                     /*  ＥＣビット＝０のときの設定  */
      }
      com_dat[2] = com_prm->c;                                  /*  シリンダ番号の設定  */
      com_dat[3] = com_prm->h;                                  /*  ヘッド番号の設定    */
      com_dat[4] = com_prm->r;                                  /*  開始セクタ番号の設定        */
      com_dat[5] = com_prm->n;                                  /*  セクタあたりのバイト数の設定        */
      com_dat[6] = com_prm->eot;                                /*  １トラックあたりのセクタ数の設定    */
      com_dat[7] = com_prm->gsl;                                /*  GAP3の読み飛ばしバイト数の設定      */
      com_dat[8] = com_prm->dtl;                                /*  読みだしセクタ数の設定(またはFFH)   */
      com_num = 0x09;
      while(time_i != 1);                                       /*  モータの回転安定時間待ち    */
      COMMAND(&com_dat[0], com_num);                            /*  ｺﾏﾝﾄﾞ･ﾓｼﾞｭｰﾙ呼びだし        */
      while(int_s != 0x01);                                     /*  コマンド終了割り込み待ち    */
      int_s = 0x00;
      if(rslt_pt[0] >= 0x40)                                    /*  コマンドの終了結果の判断    */
      {
            if((rslt_pt[0] & 0x08) == 0x08)
            {
                  ret_st = 0x01;                                /*  ノット・レディ      */
            }
            else
            {
```

(L101.C) p.24

```
                switch(rslt_pt[1])
                {
                        case 0x01:
                                if((rslt_pt[2] & 0x01) == 0x01)
                                {
                                        ret_st = 0x08;                          /*  ＤＡＭ，ＤＤＡＭ非検出      */
                                }
                                else
                                {
                                        ret_st = 0x02;                          /*  ＩＤＡＭ非検出      */
                                }
                                break;
                        case 0x04:
                                switch(rslt_pt[2])
                                {
                                        case 0x02:
                                                ret_st = 0x03;                  /*  指定したシリンダがない      */
                                                break;
                                        case 0x10:
                                                ret_st = 0x04;                  /*  指定したシリンダがない      */
                                                break;
                                        case 0x00:
                                                ret_st = 0x05;                  /*  指定したセクタがない        */
                                                break;
                                        default:
                                                break;
                                }
                                break;
                        case 0x20:
                                if((rslt_pt[2] & 0x20) == 0x20)
                                {
                                        ret_st = 0x07;          /*  データ部のＣＲＣエラー      */
                                }
                                else
                                {
                                        ret_st = 0x06;          /*  ＩＤ部のＣＲＣエラー        */
                                }
                                break;
                        case 0x10:
                                ret_st = 0x0a;                  /*  オーバラン  */
                                break;
                        case 0x80:
                                ret_st = 0x0b;                  /*  最終ｾｸﾀを越えてｱｸｾｽしようとした      */
                                break;
                        default:
                                break;
                }
        }
    }
    else
    {
        if(rslt_pt[2] < 0x40)
        {
                ret_st = 0x00;                                  /*  コマンドの正常終了  */
        }
        else
        {
                ret_st = 0x09;                                  /*  ＤＡＭ付きデータをリードした        */
        }
    }
    ret_prm->c = rslt_pt[3];                                    /*  ｺﾏﾝﾄﾞ終了時のリザルト・パラメータのセット  */
    ret_prm->h = rslt_pt[4];
    ret_prm->r = rslt_pt[5];
    ret_prm->n = rslt_pt[6];
    return ret_st;
}
```

保守／廃止

(L101.C) p.25

```
            /***********************************************/
            /*                                             */
            /*  ドライブ・セレクト処理      ＩＮ   : drno  */
            /*                              ＯＵＴ : なし  */
            /*                                             */
            /***********************************************/


void DRVSEL(unsigned char drno)
{
      unsigned char drs;
      union fapnt vector;
      struct REGPACK reg;

      switch(drno)
      {
            case 0:                                          /*  ドライブ０のとき    */
                  drs = 0x1c;
                  break;
            case 1:                                          /*  ドライブ１のとき    */
                  drs = 0x3d;
                  break;
            case 2:                                          /*  ドライブ２のとき    */
                  drs = 0x5e;
                  break;
            case 3:                                          /*  ドライブ３のとき    */
                  drs = 0x9f;
                  break;
      }
      outportb(DOR70, drs);                                  /*  ＤＯＲへのライト    */
      vector.pnt = (unsigned long)TIME;
      reg.r_ax = 0x0200;
      reg.r_cx = 38;
      reg.r_es = vector.pon.adseg;
      reg.r_bx = vector.pon.adoff;
      intr(0x1c, &reg);
}


            /*********************************************/
            /*                                           */
            /*  タイマ割り込み処理  ＩＮ   : なし        */
            /*                      ＯＵＴ : なし        */
            /*                                           */
            /*********************************************/


void interrupt TIME(void)
{
      time_i = 1;

}
```

保守／廃止

●DEVICE1.C

(DEVICE1.C) p.1

```
#include<dos.h>

#define             DOR70           0x00d4                  /*  μPD72070のﾃﾞｨｼﾞﾀﾙ･ｱｳﾄﾌﾟｯﾄ･ﾚｼﾞｽﾀのIOｱﾄﾞﾚｽ   */
#define             DIR70           0x00de                  /*  μPD72070のﾃﾞｨｼﾞﾀﾙ･ｲﾝﾌﾟｯﾄ･ﾚｼﾞｽﾀのIOｱﾄﾞﾚｽ   */


 struct command_pac
{
     char command_length;                                   /*  コマンド・パケットのバイト数        */
     char unit_num;                                         /*  論理装置コード      */
     char cmd;                                              /*  Ｉ／Ｏリクエスト・コマンド・コード  */

                 /*  ステータスのビット・フィールド(ﾜｰﾄﾞ)      */

     struct
     {
            unsigned err_cd:8;                              /*  エラー・コード(8ﾋﾞｯﾄ)       */
            unsigned don:1;                                 /*  doneビット  */
            unsigned bus:1;                                 /*  busyビット  */
            unsigned resvbit:5;                             /*  予約域(5ﾋﾞｯﾄ)       */
            unsigned err:1;                                 /*  エラー・ビット      */
     } st;
     long resv1, resv2;                                     /*  予約域（8ﾊﾞｲﾄ）     */

     union
     {
                 /*  BUILD BPBのコマンド・パケット       */

            struct
            {
                    unsigned char mediabpb;                 /*  ＤＰＢへのﾒﾃﾞｨｱ･ﾃﾞｨｽｸﾘﾌﾟﾀ   */
                    int transb_off;                         /*  転送アドレス        */
                    int transb_seg;
                    struct bpb_st *bpb_off;                 /*  ＢＰＢに対するポインタ      */
                    char *bpb_seg;
            } bb;

                    /*  リード，ライトのコマンド・パケット  */

            struct
            {
                    unsigned char mediarw;                  /*  ＤＰＢへのﾒﾃﾞｨｱ･ﾃﾞｨｽｸﾘﾌﾟﾀ   */
                    unsigned char *trns_off;                /*  転送アドレス        */
                    char *trns_seg;
                    unsigned int sect_coun;                 /*  セクタ・カウント    */
                    unsigned int start_sec;                 /*  開始セクタ  */
            } rw;

                    /*  メディア・チェックのコマンド・パケット      */

            struct
            {
                    char mediamc;                           /*  ＤＰＢへのﾒﾃﾞｨｱ･ﾃﾞｨｽｸﾘﾌﾟﾀ   */
                    int remove;                             /*  返す値      */
            } mc;

                    /*  イニシャライズのコマンド・パケット  */

            struct
            {
                    char sp_unit;                           /*  ユニット数  */
                    char *end_off;                          /*  ブレーク・アドレス  */
                    char *end_seg;
                    struct bpb_st **bpboff;                 /*  ＢＰＢの配列ポインタ        */
                    char *bpbseg;
            } intia;

     } prm;

};
```

保守／廃止

(DEVICE1.C) p.2

```
            /*  ＢＰＢの構造体      */

struct bpb_st
{
      unsigned int b_sector;                                  /*  １セクタあたりのバイト数    */
      unsigned char sector_c;                                 /*  １クラスタあたりのセクタ数  */
      unsigned int resv_sector;                               /*  予約セクタ数        */
      unsigned char fat_num;                                  /*  ＦＡＴ数    */
      unsigned int dir_num;                                   /*  ルート・ディレクトリの総エントリ数  */
      unsigned int total_sct;                                 /*  総論理セクタ数      */
      unsigned char med_id;                                   /*  メディア・ディスクリプタ    */
      unsigned int fat_sct;                                   /*  ＦＡＴのセクタ数    */
      unsigned int track_sec;                                 /*  １トラックあたりのセクタ数  */
      unsigned int hed_num;                                   /*  ヘッド数    */
      unsigned int secret;                                    /*  隠れたセクタ数      */
};

struct d_bpb
{
      char sbr1, sbr2, sbr3;                                  /*  ＳＨＯＲＴブランチ  */
      double oem_name;                                        /*  ＯＥＭメーカ名      */
      unsigned int byt_sec;                                   /*  １セクタあたりのバイト数    */
      unsigned char sect_c;                                   /*  １クラスタあたりのセクタ数  */
      unsigned int rsv_sect;                                  /*  予約セクタ数        */
      unsigned char fat;                                      /*  ＦＡＴ数    */
      unsigned int dir;                                       /*  ルート・ディレクトリの総エントリ数  */
      unsigned int all_sct;                                   /*  総論理セクタ数      */
      unsigned char med_dsl;                                  /*  メディア・ディスクリプタ    */
      unsigned int sect_fat;                                  /*  ＦＡＴのセクタ数    */
      unsigned int tr_sect;                                   /*  １トラックあたりのセクタ数  */
      unsigned int hed_n;                                     /*  ヘッド数    */
      unsigned int sec_sect;                                  /*  隠れたセクタ数      */
};

            /*  コマンド・パラメータの構造体        */

struct compra
{
      union                                                   /*  転送バイト数の共用体        */
      {
            unsigned int dtsize;                              /*  転送データのバイト数(int型) */
            struct                                            /*  int型をchar型にわける構造体 */
            {
                  unsigned char kai;                          /*  下位８ビット分      */
                  unsigned char jyoui;                        /*  上位８ビット分      */
            } int_hen;

      } trsize;

      unsigned int dt_off;                                    /*  転送ﾃﾞｰﾀのｵﾌｾｯﾄ･ｱﾄﾞﾚｽ      */
      unsigned int dt_seg;                                    /*  転送ﾃﾞｰﾀのｾｸﾞﾒﾝﾄ･ｱﾄﾞﾚｽ      */
      unsigned char c;                                        /*  シリンダ番号        */
      unsigned char h;                                        /*  ヘッド番号  */
      unsigned char r;                                        /*  開始セクタ番号      */
      unsigned char n;                                        /*  １セクタのデータ長  */
      unsigned char dtl;                                      /*  1ｾｸﾀあたりの処理ﾃﾞｰﾀ長     */
      unsigned char gsl;                                      /*  GAP3の読み飛ばしバイト数    */
      unsigned char eot;                                      /*  １トラックのセクタ数        */
      unsigned char d;                                        /*  ﾌｫｰﾏｯﾄの書きこみﾃﾞｰﾀ･ﾊﾟﾀｰﾝ  */
};


            /*  ﾘﾀｰﾝ･ｽﾃｰﾀｽ･ﾊﾟﾗﾒｰﾀの構造体    */

struct retpra
{
      unsigned char c;                                        /*  実行終了後のシリンダ番号    */
      unsigned char h;                                        /*  実行終了後のヘッド番号      */
```

保守／廃止

(DEVICE1.C) p.3

```
	unsigned char r;				/*  実行終了後のセクタ番号	*/
	unsigned char n;				/*  1セクタのデータ長	*/
};


		/*  シーク・リザルトの構造体	*/

struct sr
{
	unsigned char dr_st;				/*  シーク・コマンド実行後のﾘｻﾞﾙﾄ･ｽﾃｰﾀｽ */
	unsigned char dr_pcn;				/*  シーク・コマンド終了後のヘッドの位置	*/
};

union intg					/*  転送バイト数の共用体	*/
{
	unsigned int dtsize;			/*  転送データのバイト数(int型) */
	struct					/*  int型をchar型にわける構造体 */
	{
		unsigned char kai;		/*  下位８ビット分	*/
		unsigned char jyoui;		/*  上位８ビット分	*/
	} int_hen;
};

void IO_REQ(void);
void INITIAL(void);
void MD_CHEK(void);
void BUILD_BPB(void);
void IOCTL_IN(void);
void INPUT(void);
void OUTPUT(void);
void OUTPUT_V(void);
void ERROR_IO(void);
unsigned char UNIT_NO(void);
void ERRSET(unsigned char, struct retpra *, unsigned char, unsigned char);

void D_INT(void);
unsigned char MEDIAT(unsigned char);
unsigned char RECAL(unsigned char);
unsigned char SEEK(unsigned char, struct compra *);
unsigned char RSEEK(unsigned char, struct compra *);
unsigned char RDDAT(unsigned char, struct compra *, struct retpra *);
unsigned char RDID(unsigned char, struct retpra *);
unsigned char WRDAT(unsigned char, struct compra *, struct retpra *);
unsigned char VERIFY(unsigned char, struct compra *, struct retpra *);
void DRVSEL(unsigned char);
void HENKAN (unsigned char *, unsigned char);


extern struct command_pac far *packet;
extern char *pro_off;
extern char *pro_seg;
extern unsigned char d_exist;
extern unsigned char ready;
extern unsigned char rslt_pt[];
extern unsigned char time_i;

char unit_ex;
struct bpb_st bpbmx[4];
struct bpb_st *bpb_d[4];



		/************************************************/
		/*                                              */
		/*      Ｉ／Ｏリクエスト処理    ＩＮ  : packet  */
		/*                              ＯＵＴ：なし    */
		/*                                              */
		/************************************************/
```

保守／廃止

(DEVICE1.C) p.4

```
void IO_REQ(void)
{
      switch(packet->cmd)                                    /*  Ｉ／Ｏリクエスト・コマンド・コードに従って  */
                                                             /*  各処理を呼びだす    */
      {
            case 0x00:
                  INITIAL();                                 /*  イニシャライズ処理呼び出し  */
                  break;
            case 0x01:
                  MD_CHEK();                                 /*  メディア・チェック処理呼び出し      */
                  break;
            case 0x02:
                  BUILD_BPB();                               /*  ＢＵＩＬＤ　ＢＰＢ処理呼び出し      */
                  break;
            case 0x03:
                  IOCTL_IN();                                /*  コントロール文字列のインプット処理呼び出し  */
                  break;
            case 0x04:
                  INPUT();                                   /*  インプット処理呼び出し      */
                  break;
            case 0x08:
                  OUTPUT();                                  /*  アウトプット処理呼び出し    */
                  break;
            case 0x09:
                  OUTPUT_V();                                /*  アウトプット・ウィズ・ベリファイ処理呼び出し
                  break;
            default:
                  ERROR_IO();                                /*  Ｉ／Ｏリクエスト・エラー処理呼び出し        */
      }
      return;
}


            /*********************************************************/
            /*                                                       */
            /*      イニシャライズ処理       ＩＮ   : pro_seg, pro_off */
            /*                               ＯＵＴ : packet          */
            /*                                                       */
            /*********************************************************/


void INITIAL(void)
{
      static unsigned int w[30] = {'μ','Ｐ','Ｄ','７','２','０','７','０','デ','バ','イ','ス','・','ド','ラ','イ','バ',
'を','組','み','込','み','ま','し','た','¥x0d¥x0a','$'};

      unsigned char i;

      _AH = 0x09;
      _DX = (unsigned int)w;
      geninterrupt(0x21);

      D_INT();                                               /*  D_INTモジュール呼びだし     */

      for(i =0; i < 4; ++i)                                  /*  ＢＰＢ配列に２ＨＤ(18ｾｸﾀ)のデータをセット   */
      {
            bpbmx[i].b_sector = 512;                         /*  １セクタあたりのバイト数    */
            bpbmx[i].sector_c = 1;                           /*  １クラスタあたりのセクタ数  */
            bpbmx[i].resv_sector = 1;                        /*  予約セクタ数        */
            bpbmx[i].fat_num = 2;                            /*  ＦＡＴ数    */
            bpbmx[i].dir_num = 224;                          /*  ルート・ディレクトリの総エントリ数  */
            bpbmx[i].total_sct = 2880;                       /*  総論理セクタ数      */
            bpbmx[i].med_id = 0xf0;                          /*  メディア・ディスクリプタ    */
            bpbmx[i].fat_sct = 9;                            /*  ＦＡＴのセクタ数    */
            bpbmx[i].track_sec = 18;                         /*  １トラックあたりのセクタ数  */
            bpbmx[i].hed_num = 2;                            /*  ヘッド数    */
            bpbmx[i].secret = 0;                             /*  隠れたセクタ数      */
      }

      i = 0;                                                 /*  カウンタの初期化    */
```

保守／廃止

(DEVICE1.C) p.5

```
	unit_ex = 0;                                          /*  接続ドライブ数の初期化       */
	while(i < 4)                                          /*  接続ドライブ数のチェック(4台まで) */
	{
		if(((d_exist >> i) & 0x01) == 0x01)           /*  d_existをi回右ﾍｼﾌﾄして最下位ﾋﾞｯﾄが1ならば */
		{
			++unit_ex;                            /*  接続ドライブ数をインクリメントする  */
		}
		++i;
	}

	for(i = 0; i < 4; ++i)
	{
		bpb_d[i] = &bpbmx[i];
	}
	packet->prm.intia.bpboff = &bpb_d[0];                 /*  ＢＰＢのﾎﾟｲﾝﾀ配列へのｱﾄﾞﾚｽをpacketへｾｯﾄ  */
	packet->prm.intia.bpbseg = pro_seg;

	packet->prm.intia.sp_unit = unit_ex;                  /*  packetのユニット数に接続ドライブ数をセット  */
	packet->prm.intia.end_off = pro_off;                  /*  packetのブレーク・アドレスをセット  */
	packet->prm.intia.end_seg = pro_seg;
	packet->st.don = 1;                                   /*  packetのステータスのdoneビットに１をセット  */

	outportb(DOR70, 0x0c);                                /*  モータの停止         */
	time_i = 0;                                           /*  タイマ割り込みフラグのクリア         */

	return;
}



	/*************************************************************/
	/*                                                           */
	/*  ＢＵＩＬＤ　ＢＰＢ処理      ＩＮ  : packet, pro_seg      */
	/*                               ＯＵＴ : packet              */
	/*                                                           */
	/*************************************************************/


void BUILD_BPB(void)
{
	unsigned char ret_st, com_code, drvno, media;
	struct compra com_prm;
	struct retpra ret_prm;
	struct d_bpb bpb_dat;



	drvno = UNIT_NO();

	if(drvno > 3)                                         /*  ユニット番号が無効だった場合         */
	{
		packet->st.don = 1;                           /*  packetのステータスのdoneビットを     */
		return;                                       /*  セットしてリターン  */
	}

	bpb_dat.byt_sec = 0;

	com_code = 0x10 | drvno;
	ret_st = RECAL(com_code);                             /*  リキャリブレイトを実行       */
	if(ret_st != 0x00)                                    /*  リキャリブレイトが異常終了のとき     */
	{
		ERRSET(ret_st, &ret_prm, 0, 0);               /*  ｴﾗｰ･ｾｯﾄ･ﾓｼﾞｭｰﾙを呼び出す     */
		packet->st.don = 1;                           /*  packetのdoneビットに１をセットしてリターン  */
		return;
	}

	media = 0;
	while(media < 2)                                      /*  μPD72070のメディアの設定を変えて    */
	{                                                     /*  メディアの種類を確認する     */
		switch(media)
```

(DEVICE1.C) p.6

```
        {
                case 0:
                        MEDIAT(0x00);                   /*  メディアを２ＤＤの設定にする        */
                        break;
                case 1:
                        MEDIAT(0x01);                   /*  メディアを２ＨＤの設定にする        */
                        break;
        }
        com_code = 0x44 | drvno;
        ret_st = RDID(com_code, &ret_prm);              /*  リードＩＤ処理を実行        */


        if(ret_st == 0)                                 /*  リードＩＤ処理が正常終了ならば      */
        {                                               /*  ＷＨＩＬＥループから抜ける  */
                break;                                  /*  （メディアの種類の確認終了）        */
        }
        ++media;                                        /*  次のメディア設定で再実行する        */
}

com_code = 0x60 | drvno;                                /*  予約ｾｸﾀのﾘｰﾄﾞ(31ﾊﾞｲﾄ)の各ﾊﾟﾗﾒｰﾀの設定  */
com_prm.c = 0;                                          /*  シリンダ番号        */
com_prm.h = 0;                                          /*  ヘッド番号  */
com_prm.r = 1;                                          /*  セクタ番号  */
com_prm.n = 2;                                          /*  セクタあたりのバイト数      */
com_prm.dt_seg = (unsigned int)pro_seg;                 /*  転送アドレス(ｾｸﾞﾒﾝﾄ)        */
com_prm.dt_off = (unsigned int)&bpb_dat;                /*  転送アドレス(ｵﾌｾｯﾄ) */
com_prm.trsize.dtsize = 30;                             /*  転送サイズ  */
com_prm.gsl = 0x27;                                     /*  GAP3読み飛ばしバイト数      */
switch(media)                                           /*  トラックあたりのセクタ数    */
{
        case 0:
                com_prm.eot = 9;                        /*  ２ＤＤのセクタ数    */
                break;
        case 1:
                com_prm.eot = 18;                       /*  ２ＨＤのセクタ数    */
                break;
}
ret_st = RDDAT(com_code, &com_prm, &ret_prm);           /*  リード・データ実行  */

if(ret_st == 0)                                         /*  リードが正常終了    */
{                                                       /*  BPBの構造体（配列）に各値をセット   */

        bpbmx[packet->unit_num].b_sector = bpb_dat.byt_sec;
                                                        /*  バイト数／セクタ    */

        bpbmx[packet->unit_num].sector_c = bpb_dat.sect_c;
                                                        /*  セクタ数／クラスタ  */

        bpbmx[packet->unit_num].resv_sector = bpb_dat.rsv_sect;
                                                        /*  予約セクタ数        */

        bpbmx[packet->unit_num].fat_num = bpb_dat.fat;
                                                        /*  ＦＡＴ数    */

        bpbmx[packet->unit_num].dir_num = bpb_dat.dir;
                                                        /*  ルート・ディレクトリのエントリ数    */

        bpbmx[packet->unit_num].total_sct = bpb_dat.all_sct;
                                                        /*  総論理セクタ数      */

        bpbmx[packet->unit_num].med_id = bpb_dat.med_dsl;
                                                        /*  メディア・ディスクリプタ    */

        bpbmx[packet->unit_num].fat_sct = bpb_dat.sect_fat;
                                                        /*  セクタ数／ＦＡＴ    */

        bpbmx[packet->unit_num].track_sec = bpb_dat.tr_sect;
                                                        /*  １トラックあたりのセクタ数  */

        bpbmx[packet->unit_num].hed_num = bpb_dat.hed_n;
```

(DEVICE1.C) p.7

```
                                                        /*  ヘッドの数  */

        bpbmx[packet->unit_num].secret = bpb_dat.sec_sect;
                                                        /*  隠れたセクタの数    */
        packet->prm.bb.bpb_seg = pro_seg;               /*  packetにBPBの構造体へのポインタをセット    */

        packet->prm.bb.bpb_off = &bpbmx[packet->unit_num];
        packet->prm.bb.mediabpb = bpbmx[packet->unit_num].med_id;
                                                        /*  packetにメディア・ディスクリプタをセット   */

    }
    else                                                /*  リードが異常終了    */
    {
        ERRSET(ret_st, &ret_prm, 0, 0);                 /*  ｴﾗｰ･ｾｯﾄ･ﾓｼﾞｭｰﾙを呼び出す    */
    }
    packet->st.don = 1;                                 /*  packetのdoneビットに１をセットする  */
    outportb(DOR70, 0x0c);                              /*  モータの停止        */
    time_i = 0;                                         /*  タイマ割り込みフラグのクリア        */
    return;
}


        /*****************************************************/
        /*                                                   */
        /*  メディア・チェック処理      ＩＮ  : packet       */
        /*                              ＯＵＴ : packet      */
        /*                                                   */
        /*****************************************************/


void MD_CHEK(void)
{
    unsigned char drvno, dskchg1, dskchg2;
    struct compra com_prm;
    drvno = UNIT_NO();                                  /*  ユニット番号処理モジュール呼びだし  */
    if(drvno > 3)                                       /*  ユニット番号が無効だった場合        */
    {
        packet->st.don = 1;                             /*  packetのステータスのdoneビットを    */
        return;                                         /*  セットしてリターン*/
    }
    DRVSEL(drvno);
    dskchg1 = inportb(DIR70);
    if((dskchg1 & 0x80) == 0x80)
    {
        packet->prm.mc.remove = 0xff;                   /*  packetの返す値に-1(FFH)をセットする */
        com_prm.c = 2;
        SEEK(drvno, &com_prm);
        RECAL(drvno);
        dskchg2 = inportb(DIR70);
        if((dskchg2 & 0x80) == 0x80)
        {
            packet->st.err = 1;                         /*  packetのerrビットに１をセットする   */
            packet->st.err_cd = 0x02;                   /*  ｴﾗｰ･ｺｰﾄﾞにﾄﾞﾗｲﾌﾞのﾉｯﾄ･ﾚﾃﾞｨをセットする    */
        }
    }
    else
    {
        packet->prm.mc.remove = 0x01;                   /*  packetの返す値に1をセットする       */
    }

    packet->st.don = 1;                                 /*  packetのステータスのdoneビットをセットする  */
    outportb(DOR70, 0x0c);                              /*  モータの停止        */
    time_i = 0;                                         /*  タイマ割り込みフラグのクリア        */
    return;
}
```

保守／廃止

(DEVICE1.C) p.8

```
            /*****************************************/
            /*                                       */
            /*  インプット処理       ＩＮ  : packet    */
            /*                       ＯＵＴ : packet    */
            /*                                       */
            /*****************************************/


void INPUT(void)
{
      unsigned char ret_st, com_code, drvno, tr_sect, st_slnd, st_sect;
      unsigned char hed_num, seek_on, set_sect;
      unsigned int strsect, sec_count, rd_sect;
      struct compra com_prm;
      struct retpra ret_prm;

      drvno = UNIT_NO();
      if(drvno > 3)                                          /*  ユニット番号が無効だった場合          */
      {
            packet->prm.rw.sect_coun = 0;
            packet->st.don = 1;                              /*  packetのｽﾃｰﾀｽのdonetﾞｯﾄをｾｯﾄしてﾘﾀｰﾝ  */
            return;
      }
      strsect = packet->prm.rw.start_sec;                    /*  ｽﾀｰﾄ･ｾｸﾀ(論理ｾｸﾀ)packetから取り出す */
      tr_sect = bpbmx[packet->unit_num].track_sec;           /*  BPBから'セクタ数／トラック'を取り出す       */
      st_slnd = strsect / (tr_sect * 2);                     /*  ｽﾀｰﾄ･ｼﾘﾝﾀﾞをｽﾀｰﾄ･ｾｸﾀから求める       */
      st_sect = (strsect % tr_sect) + 1;                     /*ｽﾀｰﾄ･ｾｸﾀ(物理ｾｸﾀ)をｽﾀｰﾄ･ｾｸﾀ(論理ｾｸﾀ)から求める*/
      hed_num = (strsect / tr_sect) % 2;                     /*  ヘッド番号をｽﾀｰﾄ･ｾｸﾀ(論理ｾｸﾀ)から求める      */
      sec_count = packet->prm.rw.sect_coun;                  /*  転送セクタ数をpacketから取り出す     */
      rd_sect = 0;                                           /*  読み込み済みセクタ数カウンタをクリア        */

      do
      {
            com_code = drvno;
            com_prm.c = st_slnd;
            ret_st = SEEK(com_code, &com_prm);               /*  読み出すシリンダへシークする          */
            if(ret_st != 0x00)                               /*  シークが異常終了のとき       */
            {
                  ERRSET(ret_st, &ret_prm, 0, 0);            /*  ｴﾗｰ･ｾｯﾄ･ﾓｼﾞｭｰﾙを呼び出す     */
                  packet->prm.rw.sect_coun = rd_sect;
                  packet->st.don = 1;                        /*  packetのdoneビットに１をセットしてリターン  */
                  return;
            }

                  /*********************************/
                  /*  次シリンダへのシークの判別  */
                  /*********************************/

            if(hed_num == 0)                                 /*  ヘッド番号が０のとき          */
            {
                  if((sec_count + st_sect - 1) > (tr_sect * 2))
                                                           /*  転送ｾｸﾀ数＞ｽﾀｰﾄ･ｼﾘﾝﾀﾞの残りｾｸﾀ数のとき       */
                  {
                        seek_on = 1;                         /*  次シリンダへシークする       */
                  }
                  else                                     /*  転送ｾｸﾀ数≦ｽﾀｰﾄ･ｼﾘﾝﾀﾞの残りｾｸﾀ数のとき       */
                  {
                        seek_on = 0;                         /*  次シリンダへのシークは行わない       */
                  }
            }
            else                                             /*  ヘッド番号が１のとき          */
            {
                  if((sec_count + st_sect - 1) > tr_sect)    /*  転送ｾｸﾀ数＞ｽﾀｰﾄ･ﾄﾗｯｸの残りｾｸﾀ数のとき       */
                  {
                        seek_on = 1;                         /*  次シリンダへシークする       */
                  }
                  else                                     /*  転送ｾｸﾀ数≦ｽﾀｰﾄ･ﾄﾗｯｸの残りｾｸﾀ数のとき       */
                  {
                        seek_on = 0;                         /*  次シリンダへのシークは行わない       */
                  }
```

保守／廃止

(DEVICE1.C) p.9

```
            }
            if(seek_on == 1)                                    /*  シークを行う場合の読み込みｾｸﾀ数のセット     */
            {
                    if(hed_num == 0)                            /*  開始ｾｸﾀのヘッド番号が０のとき       */
                    {
                            set_sect = tr_sect * 2 - st_sect + 1;
                                                                /*  １シリンダのｾｸﾀ数－ｽﾀｰﾄ･ｾｸﾀ(物理ｾｸﾀ)＋１    */
                    }
                    else                                        /*  開始ｾｸﾀのヘッド番号が１のとき       */
                    {
                            set_sect = tr_sect - st_sect + 1;   /*  １トラックのｾｸﾀ数－ｽﾀｰﾄ･ｾｸﾀ(物理ｾｸﾀ)＋１    */
                    }
            }
            else                                                /*  シークを行わない場合の読み込みｾｸﾀ数のセット */
            {
                    set_sect = (unsigned char)sec_count;        /*  転送セクタ数をセット        */
            }

            com_code = 0xf0 | drvno;                            /*  リード・データのパラメータをセットする      */
            com_prm.c = st_slnd;                                /*  開始シリンダ番号    */
            com_prm.h = hed_num;                                /*  ヘッド番号  */
            com_prm.r = st_sect;                                /*  開始セクタ番号(物理ｾｸﾀ)    */
            com_prm.eot = tr_sect;                              /*  トラックあたりのセクタ数    */
            com_prm.n = 2;                                      /*  セクタあたりのバイト数      */
            com_prm.gsl = 0x1b;                                 /*  GAP3の読み飛ばしバイト数    */

            com_prm.dt_off = (unsigned int)packet->prm.rw.trns_off + rd_sect * 512;
                                                                /*  転送アドレス(ｵﾌｾｯﾄ) */
            com_prm.trsize.dtsize = set_sect * 512 ;            /*  転送バイト数        */

            com_prm.dt_seg = (unsigned int)packet->prm.rw.trns_seg;
                                                                /*  転送アドレス(ｾｸﾞﾒﾝﾄ)        */

            ret_st = RDDAT(com_code, &com_prm, &ret_prm);
                                                                /*  リード・データ実行  */
            if(ret_st != 0x00)                                  /*  リード・データが異常終了のとき      */
            {
                    ERRSET(ret_st, &ret_prm, 1, tr_sect);       /*  ｴﾗｰ･ｾｯﾄ･ﾓｼﾞｭｰﾙを呼び出す    */
                    packet->st.don = 1;                         /*  packetのdoneビットに１をセットしてリターン  */

                    return;
            }

            rd_sect += set_sect;                                /*  読み込み済みカウンタを更新する      */
            st_sect = 1;                                        /*  開始ｾｸﾀ番号を１にセット     */
            hed_num = 0;                                        /*  ヘッド番号を０にセット      */
            ++st_slnd;                                          /*  シリンダ番号をインクリメント        */
            sec_count -= set_sect;                              /*  未読セクタを更新する        */

    }
    while(seek_on != 0);                                        /*  シークを行わなくなるまでリードを繰り返す    */

    packet->prm.rw.sect_coun = rd_sect;                         /*  packetに転送終了セクタ数をセット    */
    packet->st.don = 1;                                         /*  packetのdoneビットに１をセットする  */
    outportb(DOR70, 0x0c);                                      /*  モータの停止        */
    time_i = 0;                                                 /*  タイマ割り込みフラグのクリア        */
    return;
}



            /*******************************************/
            /*                                         */
            /*  アウトプット処理    ＩＮ  : packet     */
            /*                      ＯＵＴ : packet    */
            /*                                         */
            /*******************************************/


void OUTPUT(void)
```

(DEVICE1.C) p.10

```
{
      unsigned char ret_st, com_code, drvno, tr_sect, st_slnd;
      unsigned char st_sect, hed_num, seek_on, set_sect;
      unsigned int strsect, sec_coun, wr_sect;
      unsigned int i;
      struct compra com_prm;
      struct retpra ret_prm;

      drvno = UNIT_NO();                                    /*  ユニット番号処理     */
      if(drvno > 3)                                         /*  ユニット番号が無効だった場合         */
      {
            packet->prm.rw.sect_coun = 0;
            packet->st.don = 1;                             /*  packetのステータスのdoneビットを     */
            return;                                         /*  セットしてリターン  */
      }
      strsect = packet->prm.rw.start_sec;                   /*  ｽﾀｰﾄ･ｾｸﾀ(論理ｾｸﾀ)packetから取り出す */
      tr_sect = bpbmx[packet->unit_num].track_sec;          /*  BPBから'セクタ数／トラック'を取り出す        */
      st_slnd = strsect / (tr_sect * 2);                    /*  ｽﾀｰﾄ･ｼﾘﾝﾀﾞをｽﾀｰﾄ･ｾｸﾀから求める        */
      st_sect = (strsect % tr_sect) + 1;                    /*ｽﾀｰﾄ･ｾｸﾀ(物理ｾｸﾀ)をｽﾀｰﾄ･ｾｸﾀ(論理ｾｸﾀ)から求める*/
      hed_num = (strsect / tr_sect) % 2;                    /*  ヘッド番号をｽﾀｰﾄ･ｾｸﾀ(論理ｾｸﾀ)から求める      */
      sec_coun = packet->prm.rw.sect_coun;                  /*  転送セクタ数をpacketから取り出す     */
      wr_sect = 0;                                          /*  書き込み済みセクタ数カウンタをクリア         */

      do
      {
            com_code = drvno;
            com_prm.c = st_slnd;
            ret_st = SEEK(com_code, &com_prm);              /*  書き込むシリンダへシークする         */
            if(ret_st != 0x00)                              /*  シークが異常終了のとき       */
            {
                  ERRSET(ret_st, &ret_prm, 0, 0);           /*  ｴﾗｰ･ｾｯﾄ･ﾓｼﾞｭｰﾙを呼び出す     */
                  packet->prm.rw.sect_coun = wr_sect;
                  packet->st.don = 1;                       /*  packetのdoneビットに１をセットしてリターン  */
                  return;
            }

                  /*********************************/
                  /*  次シリンダへのシークの判別  */
                  /*********************************/

            if(hed_num == 0)                                /*  ヘッド番号が０のとき         */
            {
                  if((sec_coun + st_sect - 1) > (tr_sect * 2))
                                                            /*  転送ｾｸﾀ数＞ｽﾀｰﾄ･ｼﾘﾝﾀﾞの残りｾｸﾀ数のとき       */
                  {
                        seek_on = 1;                        /*  次シリンダへシークする       */
                  }
                  else                                      /*  転送ｾｸﾀ数≦ｽﾀｰﾄ･ｼﾘﾝﾀﾞの残りｾｸﾀ数のとき       */
                  {
                        seek_on = 0;                        /*  次シリンダへのシークは行わない       */
                  }
            }
            else                                            /*  ヘッド番号が１のとき         */
            {
                  if((sec_coun + st_sect - 1) > tr_sect)    /*  転送ｾｸﾀ数＞ｽﾀｰﾄ･ﾄﾗｯｸの残りｾｸﾀ数のとき       */
                  {
                        seek_on = 1;                        /*  次シリンダへシークする       */
                  }
                  else                                      /*  転送ｾｸﾀ数≦ｽﾀｰﾄ･ﾄﾗｯｸの残りｾｸﾀ数のとき       */
                  {
                        seek_on = 0;                        /*  次シリンダへのシークは行わない       */
                  }
            }
            if(seek_on == 1)                                /*  シークを行う場合の読み込みｾｸﾀ数のセット     */
            {
                  if(hed_num == 0)                          /*  開始セクタのヘッド番号が０のとき     */
                  {
                        set_sect = tr_sect * 2 - st_sect + 1;
                                                            /*  １シリンダのｾｸﾀ数－ｽﾀｰﾄ･ｾｸﾀ(物理ｾｸﾀ)＋１     */
                  }
```


```
                else                                            /*  開始ｾｸﾀのヘッド番号が１のとき       */
                {
                        set_sect = tr_sect - st_sect + 1;       /*  １トラックのｾｸﾀ数－ｽﾀｰﾄ･ｾｸﾀ(物理ｾｸﾀ)＋１    */
                }
        }
        else                                                    /*  シークを行わない場合の書き込みｾｸﾀ数のセット */
        {
                set_sect = (unsigned char)sec_coun;             /*  転送セクタ数をセット        */
        }

        com_code = 0xd0 | drvno;                                /*  ライト・データのパラメータをセットする      */
        com_prm.c = st_slnd;                                    /*  開始シリンダ番号    */
        com_prm.h = hed_num;                                    /*  ヘッド番号  */
        com_prm.r = st_sect;                                    /*  開始セクタ番号(物理ｾｸﾀ)    */
        com_prm.n = 2;                                          /*  セクタあたりのバイト数      */
        com_prm.gsl = 0x1b;                                     /*  GAP3の読み飛ばしバイト数    */

        com_prm.dt_off = (unsigned int)packet->prm.rw.trns_off + wr_sect * 512;
                                                                /*  転送アドレス(ｵﾌｾｯﾄ) */
        com_prm.trsize.dtsize = set_sect * 512;                 /*  転送バイト数        */
        com_prm.eot = tr_sect;                                  /*  トラックあたりのセクタ数    */

        com_prm.dt_seg = (unsigned int)packet->prm.rw.trns_seg;
                                                                /*  転送アドレス(ｾｸﾞﾒﾝﾄ)       */

        ret_st = WRDAT(com_code, &com_prm, &ret_prm);
                                                                /*  ライト・データ実行  */

        for(i = 0; i < 0x278; ++i);                             /*  ６００μｓのウエイト        */

        if(ret_st != 0x00)                                      /*  ライト・データが異常終了のとき      */
        {
                ERRSET(ret_st, &ret_prm, 2, tr_sect);           /*  ｴﾗｰ･ｾｯﾄ･ﾓｼﾞｭｰﾙを呼び出す    */
                packet->st.don = 1;                             /*  packetのdoneビットに１をセットしてリターン  */
                return;
        }
        wr_sect += set_sect;                                    /*  書き込み済みカウンタを更新する      */
        st_sect = 1;                                            /*  開始ｾｸﾀ番号を１にセット     */
        hed_num = 0;                                            /*  ヘッド番号を０にセット      */
        ++st_slnd;                                              /*  シリンダ番号をインクリメント        */
        sec_coun -= set_sect;                                   /*  書き込み未終了セクタを更新する      */

    }
    while(seek_on != 0);                                        /*  シークを行わなくなるまでライトを繰り返す    */

    packet->prm.rw.sect_coun = wr_sect;                         /*  packetに転送終了セクタ数をセット    */
    packet->st.don = 1;                                         /*  packetのdoneビットに１をセットする  */
    outportb(DOR70, 0x0c);                                      /*  モータの停止        */
    time_i = 0;                                                 /*  タイマ割り込みフラグのクリア        */
    return;
}



        /*****************************************************************/
        /*                                                               */
        /*  アウトプット・ウィズ・ベリファイ処理        ＩＮ  : packet   */
        /*                                              ＯＵＴ : packet  */
        /*                                                               */
        /*****************************************************************/


void OUTPUT_V(void)
{
    unsigned char ret_st, com_code, drvno, tr_sect, st_slnd;
    unsigned char st_sect, hed_num, seek_on, set_sect;
    unsigned int strsect, sec_coun, wv_sect;
    struct compra com_prm;
    struct retpra ret_prm;
```

(DEVICE1.C) p.12

```
drvno = UNIT_NO();                                      /*  ユニット番号処理     */
if(drvno > 3)                                           /*  ユニット番号が無効だった場合          */
{
        packet->prm.rw.sect_coun = 0;
        packet->st.don = 1;                             /*  packetのステータスのdoneビットを     */
        return;                                         /*  セットしてリターン   */
}
strsect = packet->prm.rw.start_sec;                     /*  ｽﾀｰﾄ･ｾｸﾀ(論理ｾｸﾀ)packetから取り出す */
tr_sect = bpbmx[packet->unit_num].track_sec;            /*  BPBから'セクタ数／トラック'を取り出す          */
st_slnd = strsect / (tr_sect * 2);                      /*  ｽﾀｰﾄ･ｼﾘﾝﾀﾞをｽﾀｰﾄ･ｾｸﾀから求める       */
st_sect = (strsect % tr_sect) + 1;                      /*ｽﾀｰﾄ･ｾｸﾀ(物理ｾｸﾀ)をｽﾀｰﾄ･ｾｸﾀ(論理ｾｸﾀ)から求める*/
hed_num = (strsect / tr_sect) % 2;                      /*  ヘッド番号をｽﾀｰﾄ･ｾｸﾀ(論理ｾｸﾀ)から求める      */
sec_coun = packet->prm.rw.sect_coun;                    /*  転送セクタ数をpacketから取り出す     */
wv_sect = 0;                                            /*  書き込み済みセクタ数カウンタをクリア          */

do
{
        com_code = drvno;
        com_prm.c = st_slnd;
        ret_st = SEEK(com_code, &com_prm);              /*  書き込むシリンダへシークする          */
        if(ret_st != 0x00)                              /*  シークが異常終了のとき       */
        {
                ERRSET(ret_st, &ret_prm, 0, 0);         /*  ｴﾗｰ･ｾｯﾄ･ﾓｼﾞｭｰﾙを呼び出す     */
                packet->prm.rw.sect_coun = wv_sect;
                packet->st.don = 1;                     /*  packetのdoneビットに１をセットしてリターン  */
                return;
        }

                /*********************************/
                /*  次シリンダへのシークの判別  */
                /*********************************/

        if(hed_num == 0)                                /*  ヘッド番号が０のとき          */
        {
                if((sec_coun + st_sect - 1) > (tr_sect * 2))
                                                        /*  転送ｾｸﾀ数＞ｽﾀｰﾄ･ｼﾘﾝﾀﾞの残りｾｸﾀ数のとき       */
                {
                        seek_on = 1;                    /*  次シリンダへシークする       */
                }
                else                                    /*  転送ｾｸﾀ数≦ｽﾀｰﾄ･ｼﾘﾝﾀﾞの残りｾｸﾀ数のとき       */
                {
                        seek_on = 0;                    /*  次シリンダへのシークは行わない       */
                }
        }
        else                                            /*  ヘッド番号が１のとき          */
        {
                if((sec_coun + st_sect - 1) > tr_sect)  /*  転送ｾｸﾀ数＞ｽﾀｰﾄ･ﾄﾗｯｸの残りｾｸﾀ数のとき       */
                {
                        seek_on = 1;                    /*  次シリンダへシークする       */
                }
                else                                    /*  転送ｾｸﾀ数≦ｽﾀｰﾄ･ﾄﾗｯｸの残りｾｸﾀ数のとき       */
                {
                        seek_on = 0;                    /*  次シリンダへのシークは行わない       */
                }
        }
        if(seek_on == 1)                                /*  シークを行う場合の読み込みｾｸﾀ数のセット      */
        {
                if(hed_num == 0)                        /*  開始セクタのヘッド番号が０のとき     */
                {
                        set_sect = tr_sect * 2 - st_sect + 1;
                                                        /*  １シリンダのセクタ数－ｽﾀｰﾄ･ｾｸﾀ(物理ｾｸﾀ)＋１ */
                }
                else                                    /*  開始セクタのヘッド番号が１のとき     */
                {
                        set_sect = tr_sect - st_sect + 1;       /*  １トラックのｾｸﾀ数－ｽﾀｰﾄ･ｾｸﾀ(物理ｾｸﾀ)＋１     */
                }
        }
        else                                            /*  シークを行わない場合の読み込みｾｸﾀ数のセット */
        {
                set_sect = (unsigned char)sec_coun;     /*  転送セクタ数をセット          */
```

保守／廃止

(DEVICE1.C) p.13

```
            }

            com_code = 0xd0 | drvno;                                /*  ライト・データのパラメータをセットする      */
            com_prm.c = st_slnd;                                    /*  開始シリンダ番号    */
            com_prm.h = hed_num;                                    /*  ヘッド番号  */
            com_prm.r = st_sect;                                    /*  開始セクタ番号(物理ｾｸﾀ)     */
            com_prm.n = 2;                                          /*  セクタあたりのバイト数      */
            com_prm.gsl = 0x1b;                                     /*  GAP3の読み飛ばしバイト数    */

            com_prm.dt_off = (unsigned int)packet->prm.rw.trns_off + wv_sect * 512;
                                                                    /*  転送アドレス(ｵﾌｾｯﾄ) */
            com_prm.trsize.dtsize = set_sect * 512;                 /*  転送バイト数        */

            com_prm.eot = tr_sect;                                  /*  トラックあたりのセクタ数    */

            com_prm.dt_seg = (unsigned int)packet->prm.rw.trns_seg;
                                                                    /*  転送アドレス(ｾｸﾞﾒﾝﾄ)       */

            ret_st = WRDAT(com_code, &com_prm, &ret_prm);
                                                                    /*  ライト・データ実行  */
            if(ret_st != 0x00)                                      /*  ライト・データが異常終了のとき      */
            {
                    ERRSET(ret_st, &ret_prm, 2, tr_sect);           /*  ｴﾗｰ･ｾｯﾄ･ﾓｼﾞｭｰﾙを呼び出す    */
                    packet->st.don = 1;                             /*  packetのdoneビットに１をセットしてリターン  */
                    return;
            }
            com_code = 0xe4 | drvno;                                /*  ベリファイのパラメータをセット      */
            com_prm.dtl = set_sect;                                 /*  ベリファイを行うセクタ数    */

            ret_st = VERIFY(com_code, &com_prm, &ret_prm);
                                                                    /*  ベリファイ実行      */
            if(ret_st != 0x00)                                      /*  ベリファイが異常終了のとき  */
            {
                    ERRSET(ret_st, &ret_prm, 1, tr_sect);           /*  ｴﾗｰ･ｾｯﾄ･ﾓｼﾞｭｰﾙを呼び出す    */
                    packet->st.don = 1;                             /*  packetのdoneビットに１をセットしてリターン  */
                    return;
            }
            wv_sect += set_sect;                                    /*  書き込み済みカウンタを更新する      */
            st_sect = 1;                                            /*  開始ｾｸﾀ番号を１にセット     */
            hed_num = 0;                                            /*  ヘッド番号を０にセット      */
            ++st_slnd;                                              /*  シリンダ番号をインクリメント        */
            sec_coun -= set_sect;                                   /*  書き込み未終了セクタを更新する      */
    }
    while(seek_on != 0);                                            /*  シークを行わなくなるまでライトを繰り返す  */

    packet->prm.rw.sect_coun = wv_sect;                             /*  packetに転送終了セクタ数をセット    */
    packet->st.don = 1;                                             /*  packetのdoneビットに１をセットする  */
    outportb(DOR70, 0x0c);                                          /*  モータの停止        */
    time_i = 0;                                                     /*  タイマ割り込みフラグのクリア        */
    return;
}


            /*****************************************************************/
            /*                                                               */
            /*  ＩＯＣＴＬ　インプット処理          ＩＮ　：pro_seg,d_exist */
            /*                                      ＯＵＴ：packet           */
            /*                                                               */
            /*****************************************************************/


void IOCTL_IN(void)
{
    unsigned char far *ioctl;
    ioctl = MK_FP(packet->prm.rw.trns_seg, packet->prm.rw.trns_off);
                                                                    /*  アドレスにｆａｒポインタをセット    */
    *ioctl = d_exist;
    packet->prm.rw.sect_coun = 1;                                   /*  転送バイト数をpacketにセットする    */
    packet->st.don = 1;                                             /*  packetのdoneビットに１をセットする  */
    return;
```

保守／廃止

(DEVICE1.C) p.14

```
}

        /*************************************************************/
        /*                                                           */
        /*  ユニット番号処理     ＩＮ   : packet, d_exist, unit_ex     */
        /*                       ＯＵＴ : packet, drvno               */
        /*                                                           */
        /*************************************************************/

unsigned char UNIT_NO(void)
{
    unsigned char drvno, mdrvno, sdrvno, i;

    if(packet->unit_num > unit_ex)                          /*  packetのユニット番号が接続されている       */
    {                                                       /*  ドライブ数より大きいとき     */
        packet->st.err =1;                                  /*  packetのエラー・ビットに１をセットする     */
        packet->st.err_cd = 1;                              /*  packetのエラー・コードに'    */
        return 4;                                           /*  無効なユニット番号'をセットする     */
    }
    drvno = packet->unit_num;                               /*  デバイスの物理ドライブ番号に         */
                                                            /*  ユニット番号をセットする     */
    sdrvno = 0;
    mdrvno = 0;
    i = 0;
    do
    {
        if(((d_exist >> i) & 0x01) == 0x00)                 /*  接続されていないﾄﾞﾗｲﾌﾞ番号の判定     */
        {
            ++sdrvno;                                       /*  接続されていないﾄﾞﾗｲﾌﾞをインクリメント     */
        }
        else
        {
            ++mdrvno;                                       /*  接続されているﾄﾞﾗｲﾌﾞ数をインクリメント     */
        }
        ++i;
    }
    while((drvno + 1) > mdrvno);                            /*  選択されたﾄﾞﾗｲﾌﾞ数がｶｳﾝﾄされるまで繰り返し  */
    drvno += sdrvno;                                        /*接続されていなかったﾄﾞﾗｲﾌﾞ数をﾄﾞﾗｲﾌﾞ番号に加算*/
    return drvno;
}



        /*************************************************************************/
        /*                                                                       */
        /*  エラー・セット処理   ＩＮ   : packet, ret_st, ex_sw, tr_sect         */
        /*                       ＯＵＴ : packet                                 */
        /*                                                                       */
        /*************************************************************************/

void ERRSET(unsigned char ret_st, struct retpra *ret_prm,
      unsigned char ex_sw, unsigned char tr_sect)
{
    unsigned int ex_sect;


    packet->st.err = 1;                                     /*  packetのerrビットに１をセットする    */
    switch(ret_st)
    {
        case 0x01:                                          /*  ret_st＝01Hのとき    */
            packet->st.err_cd = 0x02;                       /*  ｴﾗｰ･ｺｰﾄﾞに'ﾄﾞﾗｲﾌﾞのﾉｯﾄ･ﾚﾃﾞｨ'をセットする     */
            break;
        case 0x02:                                          /*  ret_st＝02Hのとき    */
            packet->st.err_cd = 0x0c;                       /*  ｴﾗｰ･ｺｰﾄﾞに'一般的なエラー'をセットする       */
            break;
        case 0x03:                                          /*  ret_st＝03H-05Hのとき        */
        case 0x04:
        case 0x05:
            packet->st.err_cd = 0x08;                       /*  ｴﾗｰ･ｺｰﾄﾞに'セクタが存在しない'をセットする  */
            break;
```

(DEVICE1.C) p.15

```
            case 0x06:                                          /*  ret_st＝06H,07Hのとき       */
            case 0x07:
                    packet->st.err_cd = 0x04;                   /*  ｴﾗｰ･ｺｰﾄﾞに'ＣＲＣエラー'をセットする       */
                    break;
            case 0x08:                                          /*  ret_st＝08H,09Hのとき       */
            case 0x09:
                    packet->st.err_cd = 0x0b;                   /*  ｴﾗｰ･ｺｰﾄﾞに'読み出しエラー'をセットする     */
                    break;
            case 0x0a:                                          /*  ret_st＝0AH,0BHのとき       */
            case 0x0b:
                    if(ex_sw == 1)                              /*  ex_sw＝1(インプット処理)のとき      */
                    {
                            packet->st.err_cd = 0x0b;           /*  ｴﾗｰ･ｺｰﾄﾞに'読み出しエラー'をセットする     */
                    }
                    else                                        /*  ex_sw≠1(アウトプット処理)のとき    */
                    {
                            packet->st.err_cd = 0x0a;           /*  ｴﾗｰ･ｺｰﾄﾞに'書き込みエラー'をセットする     */
                    }
                    break;
            case 0x0c:                                          /*  ret_st＝0CHのとき   */
                    packet->st.err_cd = 0x00;                   /*  ｴﾗｰ･ｺｰﾄﾞに'ライト・プロテクト'をセットする */
                    break;
            case 0x0d:                                          /*  ret_st＝0DHのとき   */
                    packet->st.err_cd = 0x06;                   /*  ｴﾗｰ･ｺｰﾄﾞに'シーク・エラー'をセットする     */
                    break;
            case 0x0e:                                          /*  ret_st＝0EHのとき   */
                    packet->st.err_cd = 0x0b;                   /*  ｴﾗｰ･ｺｰﾄﾞに'読み出しエラー'をセットする     */
                    break;
            default:                                            /*  その他      */
                    packet->st.err_cd = 0x0c;                   /*  ｴﾗｰ･ｺｰﾄﾞに'その他のエラー'をセットする     */
                    break;
    }
    if(ex_sw != 0)                                              /*  呼び出した処理がインプットまたは    */
                                                                /*  アウトプット処理のとき      */
    {
            ex_sect = (tr_sect * 2 * ret_prm->c) + ret_prm->h * tr_sect + ret_prm->r - 1;
                                                                /*  実行が終了した論理セクタ番号の算出  */

            packet->prm.rw.sect_coun = ex_sect - packet->prm.rw.start_sec;
                                                                /*  転送されたセクタ数をセット  */
    }

    outportb(DOR70, 0x0c);                                      /*  モータの停止        */
    time_i = 0;                                                 /*  タイマ割り込みフラグのクリア        */
    return;
}

            /*****************************************************/
            /*                                                   */
            /*    Ｉ／Ｏリクエスト・エラー処理　ＩＮ　：なし    */
            /*                                  ＯＵＴ：packet   */
            /*                                                   */
            /*****************************************************/

void ERROR_IO(void)
{
    packet->st.err = 1;
    packet->st.err_cd = 0x03;
    return;
}
```

●FORMAT71.C

(FORMAT71.C) p.1

```
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>

#define         SELCT1          "[8;20H"                /*  ﾄﾞﾗｲﾌﾞ選択の初期位置(ｴｽｹｰﾌﾟ･ｼｰｹﾝｽ)          */
#define         SELCT2          "[10;20H"               /*  ﾒﾃﾞｨｱ選択の初期位置(ｴｽｹｰﾌﾟ･ｼｰｹﾝｽ)          */
#define         SYOKI1          8                       /*  ﾄﾞﾗｲﾌﾞ選択の初期位置(行)  */
#define         SYOKI2          10                      /*  ﾒﾃﾞｨｱ選択の初期位置(行)  */
#define         MESS            "[22;10H"               /*  ｴﾗｰなどのﾒｯｾｰｼﾞの表示位置(ｴｽｹｰﾌﾟ･ｼｰｹﾝｽ)  */
#define         EXECUTM         "[19;10H"               /*  実行中のﾒｯｾｰｼﾞ表示位置(ｴｽｹｰﾌﾟ･ｼｰｹﾝｽ)    */
#define         TRACKM          "[19;52H"               /*  ﾌｫｰﾏｯﾄ実行中ﾄﾗｯｸの          */
                                                        /*  表示位置(ｴｽｹｰﾌﾟ･ｼｰｹﾝｽ)      */
#define         HOMECL          "[2J"                   /*  画面のﾎｰﾑ・ｸﾘｱ(ｴｽｹｰﾌﾟ･ｼｰｹﾝｽ)  */
#define         DOR70           0x00d4                  /*  μPD72070のﾃﾞｨｼﾞﾀﾙ･ｱｳﾄﾌﾟｯﾄ･ﾚｼﾞｽﾀのIOｱﾄﾞﾚｽ  */
#define         INTVEC          0x0b                    /*  割り込みベクタ番号  */


struct compra
{                                                       /*  ｺﾏﾝﾄﾞ・ﾊﾟﾗﾒｰﾀの構造体          */
      union                                             /*  転送ﾊﾞｲﾄ数の共用体          */
      {
              unsigned int dtsz;                        /*  転送ﾃﾞｰﾀのﾊﾞｲﾄ数(int型)  */
              struct                                    /*  int型をchar型にわける構造体  */
              {
                      unsigned char kai;                /*  下位８ビット分        */
                      unsigned char jyoui;              /*  上位８ビット分        */
              } int_hen;

      } trsize;
      unsigned int dt_off;                              /*  転送ﾃﾞｰﾀのｵﾌｾｯﾄ･ｱﾄﾞﾚｽ          */
      unsigned int dt_seg;                              /*  転送ﾃﾞｰﾀのｾｸﾞﾒﾝﾄ･ｱﾄﾞﾚｽ          */
      unsigned char c;                                  /*  シリンダ番号          */
      unsigned char h;                                  /*  ヘッド番号  */
      unsigned char r;                                  /*  開始セクタ番号        */
      unsigned char n;                                  /*  １セクタのデータ長  */
      unsigned char dtl;                                /*  1ｾｸﾀあたりの処理ﾃﾞｰﾀ長        */
      unsigned char gsl;                                /*  GAP3の読み飛ばしバイト数      */
      unsigned char eot;                                /*  １トラックのセクタ数          */
      unsigned char d;                                  /*  ﾌｫｰﾏｯﾄの書きこみﾃﾞｰﾀ･ﾊﾟﾀｰﾝ  */
};

struct retpra
{                                                       /*  ﾘﾀｰﾝ･ｽﾃｰﾀｽ･ﾊﾟﾗﾒｰﾀの構造体    */
      unsigned char c;                                  /*  実行終了後のシリンダ番号      */
      unsigned char h;                                  /*  実行終了後のヘッド番号        */
      unsigned char r;                                  /*  実行終了後のセクタ番号        */
      unsigned char n;                                  /*  １セクタのデータ長  */
};

struct sr                                               /*  シーク・リザルトの構造体      */
{
      unsigned char dr_st;                              /*  シーク・コマンド実行後のﾘｻﾞﾙﾄ･ｽﾃｰﾀｽ  */
      unsigned char dr_pcn;                             /*  シーク・コマンド終了後のヘッドの位置  */
};

struct form                                             /*  ﾌｫｰﾏｯﾄ時のＩＤ部の書き込みデータの構造体  */
{
      unsigned char c;                                  /*  シリンダ番号          */
      unsigned char h;                                  /*  ヘッド番号  */
      unsigned char r;                                  /*  セクタ番号  */
      unsigned char n;                                  /*  セクタあたりのフォーマット・バイト数  */
};

struct fpnt                                             /*  ｆａｒポインタをセグメントとオフセットに  */
                                                        /*  分割する構造体        */
{
      unsigned int adoff;
      unsigned int adseg;
};
```

保守／廃止

(FORMAT71.C) p.2

```
union fapnt                                                    /*  ｆａｒポインタの共用体宣言  */
{
      unsigned long pnt;                                       /*  ｆａｒポインタ      */
      struct fpnt pon;                                         /*  オフセットとセグメント      */
};


void exit(int);
void ERRPR(unsigned char);
void COMMAND(unsigned char *, char);
void interrupt INTRPT(void);
char RESULT(void);
void MEDIAT(unsigned char);
unsigned char RECAL(unsigned char);
unsigned char SEEK(unsigned char, struct compra *);
unsigned char RSEEK(unsigned char, struct compra *);
unsigned char TRFORMAT(unsigned char, struct compra *, struct retpra *);
unsigned char WRDAT(unsigned char, struct compra *, struct retpra *);
unsigned char VERIFY(unsigned char, struct compra *, struct retpra *);
unsigned getkey(void);                                         /*  キー・コード取得関数        */
void interrupt (*OLD_VEC)(void);

extern unsigned char int_s;                                    /*  割り込みの確認変数  */
extern unsigned char time_i;                                   /*  タイマ割り込みフラグ        */

            /*************************************************************************/
            /*                                                                       */
            /*              フォーマット・コマンド・メイン・ルーチン                 */
            /*                                                                       */
            /*************************************************************************/

void main(void)
{
      struct REGPACK reg;
      struct compra com_prm;
      struct retpra ret_prm;
      union fapnt vector;
      char aldrv, drv70, drvno, meda, sldcun, hedno, errcun, sccoun, rdcoun, i, d, ichi;
      char com_code, curdrv, sdrvno, mdrvno;
      char ret_st;
      int key_in;
      unsigned char adpnt[512], d_exist;
      unsigned char far *d_point;
      unsigned char rtd_dat;

            /*****************************************/
            /*  ２ＤＤディスクの予約セクタ・データ  */
            /*****************************************/

      unsigned char dd9_res[30] = { 0xeb, 0x1c, 0x90, 0x55, 0x50, 0x44, 0x37,
                                                          0x32, 0x30, 0x37, 0x30, 0x00, 0x02, 0x02,
                                                          0x01, 0x00, 0x02, 0x70, 0x00, 0xa0, 0x05,
                                                          0xf9, 0x03, 0x00, 0x09, 0x00, 0x02, 0x00,
                                                          0x00, 0x00 };

            /*****************************************/
            /* ２ＨＤディスクの予約セクタ・データ   */
            /*****************************************/

      unsigned char hd8_res[30] = { 0xeb, 0x1c, 0x90, 0x55, 0x50, 0x44, 0x37,
                                                          0x32, 0x30, 0x37, 0x30, 0x00, 0x02, 0x01,
                                                          0x01, 0x00, 0x02, 0xe0, 0x00, 0x40, 0x0b,
                                                          0xf0, 0x09, 0x00, 0x12, 0x00, 0x02, 0x00,
                                                          0x00, 0x00 };

            /*********************************************/
            /*  ２ＤＤディスクのＦＡＴセクタのデータ    */
            /*********************************************/

      unsigned char dd9_fat[3] = { 0xf9, 0xff, 0xff};
```

保守／廃止

(FORMAT71.C) p.3

```
        /*****************************************/
        /*  ２ＨＤディスクのＦＡＴセクタのデータ    */
        /*****************************************/

unsigned char hd8_fat[3] = { 0xf0, 0xff, 0xff};


int_s = 0x00;

d_exist = 0;
d_point = &d_exist;
OLD_VEC = getvect(INTVEC);

vector.pnt = (unsigned long)INTRPT;                          /*  割り込ﾓｼﾞｭｰﾙのアドレスを共用体へセット    */

poke(0x0000, (unsigned int)(INTVEC * 4), vector.pon.adoff);
                                                             /*  割り込ﾓｼﾞｭｰﾙのｵﾌｾｯﾄ･ｱﾄﾞﾚｽをﾍﾞｸﾀにｾｯﾄ  */

poke(0x0000, (unsigned int)(INTVEC * 4 + 2), vector.pon.adseg);
                                                             /*  割り込ﾓｼﾞｭｰﾙのｾｸﾞﾒﾝﾄ･ｱﾄﾞﾚｽをﾍﾞｸﾀにｾｯﾄ  */

        /*****************************/
        /*  全接続ドライブ数の取得  */
        /*****************************/

reg.r_ax = 0x1900;                                           /*  カレント・ドライブの取得（保存）    */
intr(0x21, &reg);                                            /*  al = カレント・ドライブ      */
curdrv = (unsigned char)(reg.r_ax & 0x00ff);
do
{
        aldrv = (unsigned char)(reg.r_ax & 0x00ff);
        reg.r_dx = aldrv + 1;                                /*  カレント・ドライブを変更する        */
                                                             /*  （論理番号をインクリメント）        */
        reg.r_ax = 0x0e00;
        intr(0x21, &reg);
        reg.r_ax = 0x1900;                                   /*  カレント・ドライブの取得    */
        intr(0x21, &reg);                                    /*  al = カレント・ドライブ      */

}
while(aldrv != (unsigned char)(reg.r_ax & 0x00ff));
                                                             /*  ｶﾚﾝﾄ･ﾄﾞﾗｲﾌﾞが変わらなくなるまで繰り返す    */
reg.r_dx = curdrv;                                           /*  カレント・ドライブを元に戻す        */
reg.r_ax = 0x0e00;
intr(0x21, &reg);
++aldrv;                                                     /*  総論理ドライブ数＝最大論理ドライブ番号＋１  */


        /*************************/
        /*     d_existの取得     */
        /*************************/

_AH = 0x44;
_AL = 0x04;
_BL = aldrv;
_CX = 0x0001;
_DS = FP_SEG(d_point);
_DX = FP_OFF(d_point);
geninterrupt(0x21);

        /*************************/
        /* ドライブの接続チェック */
        /*************************/

if(d_exist == 0x00)                                          /*  ドライブが無接続か？        */
{
        printf("ドライブがありません¥n");
        exit(0);
}
i = 0;
```

保守／廃止

(FORMAT71.C) p.4

```
drv70 = 0;

while(i < 4)                                        /*  接続ドライブ数のチェック     */
{
        if(((d_exist >> i) & 0x01) == 0x01)
        {
                ++drv70;
        }
        ++i;
}
drvno = 0;
d = (char)aldrv - drv70;                            /*  ドライブ数のチェック */

        /*********************************/
        /*   コマンド・タイトルの表示   */
        /*********************************/

printf("¥x1b[>5h");                                 /*  カーソルの消去       */
printf("¥x1b%s", HOMECL);                           /*  画面のクリア         */

printf("¥x1b[3;8Hμ ＰＤ７２０７０用フォーマットコマンド");

i = 1;
while(i <= drv70)
{
        ichi = SYOKI1;
        ichi = ichi + (i - 1) * 2;
        printf("¥x1b[%d;20H", ichi);
        printf("%c：ドライブ", (0x40 + d + i));
        i++;
}
printf("¥x1b%s↑，↓キーで選択，リターンで決定", MESS);
printf("¥x1b%s", SELCT1);
printf("¥x1b[7m");                                  /*  文字の反転表示       */
printf("%c：ドライブ", (0x40 + d + drvno + 1));
printf("¥x1b%s", SELCT1);

        /************************************************/
        /*   入力キー・コードの取得およびdrvnoのセット   */
        /************************************************/

do
{
        key_in = getkey();                          /*  入力キー・コードの取得       */
        printf("¥x1b[%d;20H", ichi);                /*  カーソルの移動       */
        printf("¥x1b[0m");                          /*  文字の正常表示       */
        printf("%c：ドライブ", (0x40 + d + drvno + 1));
        switch(key_in)
        {
                case 0x3a00:                        /*  入力キーが↑の場合           */
                        --drvno;
                        if(drvno < 0)
                        {
                                drvno = drv70 - 1;
                        }
                        break;
                case 0x3d00:                        /*  入力キーが↓の場合           */
                        ++drvno;
                        if(drvno >= drv70)
                        {
                                drvno = 0;
                        }
                        break;
                default :                           /*  その他のキー入力の場合       */
                        break;
        }
        ichi = SYOKI1 + drvno * 2;
        printf("¥x1b[%d;20H", ichi);                /*  カーソルの移動       */
        printf("¥x1b[7m");                          /*  文字の反転表示       */
        printf("%c：ドライブ", (0x40 + d + drvno + 1));
```

(FORMAT71.C) p.5

```
}
while(key_in != 0x1c0d);

        /*******************************/
        /*     選択ドライブの表示      */
        /*******************************/

printf("¥x1b[0m");                                          /*  文字の正常表示        */

printf("¥x1b%s選択ドライブ  %c:", SELCT1, (0x40 + d + drvno + 1));

        /*************************************************************/
        /*     実際に接続されているドライブとdrvnoをあわせる      */
        /*************************************************************/

i = 0;
sdrvno = 0;
mdrvno = 0;
do
{
        if(((d_exist >> i) & 0x01) == 0x00)                 /*  接続されていないﾄﾞﾗｲﾌﾞの判定          */
        {
                ++sdrvno;
        }
        else                                                /*  接続されているﾄﾞﾗｲﾌﾞのカウント        */
        {
                ++mdrvno;
        }
        ++i;
}
while((drvno + 1) > mdrvno);                                /*  選択されたﾄﾞﾗｲﾌﾞ数がｶｳﾝﾄされるまで繰り返し  */

drvno += sdrvno;                                            /*  接続されていなかったﾄﾞﾗｲﾌﾞ数だけ加算  */

        /*********************************/
        /*   メディアの選択画面の表示    */
        /*********************************/

meda = 0;
printf("¥x1b[%d;20H¥x1b[J", SYOKI2);                        /*  カーソル行のクリア  */
printf("¥x1b[7m");                                          /*  表示文字を反転表示にする    */
printf("¥x1b[%d;22H２ＤＤ", SYOKI2 );
printf("¥x1b[0m");                                          /*  表示文字を正常表示にする    */
printf("¥x1b[%d;22H２ＨＤ", SYOKI2 + 2);
printf("¥x1b%s↑，↓キーで選択，リターンで決定", MESS);
printf("¥x1b[%d;22H", SYOKI2);

        /*************************************************************/
        /*      入力キー・コードの取得とmedaへメディアのセット       */
        /*************************************************************/

do
{
        key_in = getkey();                                  /*  入力キー・コードを取得    */
        switch(key_in)
        {
                case 0x3a00:                                /*   入力キーが↑の場合      */
                        --meda;
                        if(meda < 0)
                        {
                                meda = 0x01;
                        }
                        break;
                case 0x3d00:                                /*   入力キーが↓の場合      */
                        ++meda;
                        if(meda == 0x02)
                        {
                                meda = 0;
                        }
                        break;
```

保守／廃止

(FORMAT71.C) p.6

```
                default :                                       /*  その他のキー入力の場合      */
                        break;
        }
        printf("¥x1b[%d;22H２ＤＤ", SYOKI2 );                  /*  すべての表示を正常表示にする        */
        printf("¥x1b[%d;22H２ＨＤ", SYOKI2 + 2);
        printf("¥x1b[7m");                                      /*  表示文字を反転表示にする        */

        switch(meda)                                            /*  現在選択されているメディアを反転表示にする  */
        {
                case 0:
                        printf("¥x1b[%d;22H２ＤＤ", SYOKI2 );
                        break;
                case 1:
                        printf("¥x1b[%d;22H２ＨＤ", SYOKI2 + 2);
                        break;
        }
        printf("¥x1b[0m");                                      /*  表示文字を正常表示にする    */
}
while(key_in != 0x1c0d);                                        /*  入力キーがリターンで決定     */

        /*****************************/
        /*   選択メディアの表示     */
        /*****************************/

printf("¥x1b%s¥x1b[J", SELCT2);
switch(meda)
{
        case 0x00:
                printf("¥x1b%s選択メディア　２ＤＤ", SELCT2);
                break;
        case 0x01:
                printf("¥x1b%s選択メディア　２ＨＤ", SELCT2);
                break;
}
com_code = meda;                                                /*  MEDIAT用のコマンド・コードをセット*/
MEDIAT(com_code);                                               /*  MEDIATモジュール呼び出し    */

com_code = 0x0c | drvno;                                        /*  リキャリブレイト用コマンド・コードをセット  */
ret_st = RECAL(com_code);                                       /*  リキャリブレイト実行        */

if(ret_st != 0x00)                                              /*  エラー発生か？      */
{
        ERRPR(ret_st);
        exit(0);
}

printf("¥x1b%s", EXECUTM);                                      /*  実行メッセージの表示  */
printf("フォーマット中");

sldcun = 0;                                                     /*   sldcunの初期化             */
hedno = 0;                                                      /*   hednoの初期化              */

        /*********************************/
        /*    トラック・フォーマット     */
        /*********************************/

while(sldcun <= 79)
{
        if((hedno == 0) && (sldcun != 0))                       /*  シークの判断        */
        {
                com_prm.c = sldcun;
                com_code = drvno;
                ret_st = SEEK(com_code, &com_prm);              /*  シーク実行  */

                if(ret_st != 0x00)                              /*  エラー発生か？      */
                {
                        ERRPR(ret_st);
                        exit(0);
                }
        }
```

保守／廃止

(FORMAT71.C) p.7

```
if(hedno == 0x00)                                   /*  フォーマット中のトラックの表示      */
{
        printf("¥x1b%s%3dトラック", TRACKM, (sldcun * 2));
}
else
{
        printf("¥x1b%s%3dトラック", TRACKM, (sldcun * 2 + 1));
}

errcun = 0;                                         /*  エラー・カウンタの初期化    */
while(errcun < 2)
{
        if(hedno == 0x00)                           /*  フォーマットするサイドの判断 */
        {
                com_code = 0x4c | drvno;            /*  サイド0の場合       */
        }
        else
        {
                com_code = 0x6c | drvno;            /*  サイド1の場合       */
        }
        com_prm.c = sldcun;                         /*  シリンダ番号の設定  */
        com_prm.n = 0x02;                           /*  512/1ｾｸﾀに設定      */

        switch(meda)                                /*  1ﾄﾗｯｸあたりのｾｸﾀ数の設定   */
        {
                case 0x00:                          /*  2DD = 9ｾｸﾀ/ﾄﾗｯｸ       */
                        com_prm.dtl = 9;
                        break;
                case 0x01:                          /*  2HD = 18ｾｸﾀ/ﾄﾗｯｸ     */
                        com_prm.dtl = 18;
                        break;
        }

        com_prm.gsl = 84;
        com_prm.d = 0xe5;                           /*  データ領域書き込みデータ      */

        ret_st = TRFORMAT(com_code, &com_prm, &ret_prm);
                                                    /*  トラック・フォーマット実行  */

        if(ret_st != 0x00)                          /*  エラー発生？  */
        {
                ERRPR(ret_st);
                exit(0);
        }

        /************************************************/
        /*  フォーマットしたトラックをベリファイする    */
        /************************************************/

        com_code = 0x40 | drvno;
        com_prm.c = sldcun;                         /*  シリンダ番号のセット        */
        com_prm.h = hedno;                          /*  ヘッド番号のセット          */
        com_prm.r = 0x01;                           /*  開始セクタ番号のセット      */

        switch(meda)                                /*  最終セクタ番号の設定        */
        {
                case 0x00:                          /*  2DD = 9ｾｸﾀ   */
                        com_prm.eot = 9;
                        break;
                case 0x01:                          /*  2HD = 18ｾｸﾀ */
                        com_prm.eot = 18;
                        break;
        }
        com_prm.gsl = 27;                           /*  GAP3の読み飛ばしバイト数      */
        com_prm.dtl = 0xff;                         /*  規定値              */

        ret_st = VERIFY(com_code, &com_prm, &ret_prm);
                                                    /*  ベリファイ実行    */
```

```
                if(ret_st != 0x00)                          /*  エラー発生か？      */
                {
                        ++errcun;                           /*  Ｙｅｓ→ｴﾗｰ･ｶｳﾝﾀをｲﾝｸﾘﾒﾝﾄ   */
                }
                else
                {
                        break;                              /*  Ｎｏ→次の処理へ    */
                }
        }
        if(ret_st != 0x00)                                  /*  エラー発生か？      */
        {
                ERRPR(ret_st);
                exit(0);
        }
        if(hedno == 0x00)                                   /*  フォーマットしたサイドの判断 */
        {
                hedno = 0x01;
        }
        else
        {
                hedno = 0x00;
                ++sldcun;                                   /*  シリンダ番号をインクリメント */
        }
}

        /************************/
        /*  リキャリブレイト    */
        /************************/

com_code = 0x0c | drvno;                                    /*  リキャリブレイト用コマンド・コードをセット */
ret_st = RECAL(com_code);

if(ret_st != 0x00)                                          /*  エラー発生か？      */
{
        ERRPR(ret_st);
        exit(0);
}

        /************************/
        /*  予約セクタのライト  */
        /************************/

errcun = 0;
while(errcun < 2)
{
        i = 0;
        while(i < 30)
        {
                switch(meda)                                /*  メディア別に予約セクタのデータをセット      */
                {
                        case 0x00:                          /*  ２ＤＤ      */
                                adpnt[i] = dd9_res[i];
                                break;
                        case 0x01:                          /*  ２ＨＤ      */
                                adpnt[i] = hd8_res[i];
                                break;
                }
                ++i;
        }
        com_code = 0x4c | drvno;
        com_prm.trsize.dtsz = 30;                           /*  転送データ・サイズのセット */
        com_prm.dt_off = FP_OFF(&adpnt[0]);                 /*  ｱﾄﾞﾚｽ･ﾎﾟｲﾝﾀのｵﾌｾｯﾄ･ｱﾄﾞﾚｽをセットする        */
        com_prm.dt_seg = FP_SEG(&adpnt[0]);                 /*  ｱﾄﾞﾚｽ･ﾎﾟｲﾝﾀのｾｸﾞﾒﾝﾄ･ｱﾄﾞﾚｽをセットする        */
        com_prm.c = 0x00;                                   /*  シリンダ番号をセット        */
        com_prm.h = 0x00;                                   /*  ヘッド番号をセット  */
        com_prm.r = 0x01;                                   /*  実行セクタ番号をセット      */

        ret_st = WRDAT(com_code, &com_prm, &ret_prm);
                                                            /*  ライト・データ実行  */
```

(FORMAT71.C) p.9

```
        if(ret_st != 0x00)                                  /*  エラー発生？        */
        {
                ERRPR(ret_st);
                exit(0);
        }

        /*****************************/
        /*  予約セクタのベリファイ  */
        /*****************************/

        com_code = 0x44 | drvno;
        com_prm.dtl = 0x01;
        ret_st = VERIFY(com_code, &com_prm, &ret_prm);
                                                            /*  ベリファイ実行      */

        if(ret_st != 0x00)                                  /*  エラー発生？        */
        {
                ++errcun;                                   /*  Ｙｅｓ→ｴﾗｰ･ｶｳﾝﾀをｲﾝｸﾘﾒﾝﾄ   */
        }
        else
        {
                break;                                      /*  Ｎｏ→次の処理へ    */
        }
}
if(ret_st != 0x00)                                          /*  エラー発生？        */
{
        ERRPR(ret_st);
        exit(0);
}

        /*****************************/
        /*    ＦＡＴセクタのライト  */
        /*****************************/

errcun = 0;
while(errcun < 2)
{
        i = 0;
        while(i < 3)
        {
                switch(meda)                                /*  メディア別にFATセクタのデータをセット       */
                {
                        case 0x00:                          /*  ２ＤＤ      */
                                adpnt[i] = dd9_fat[i];
                                break;
                        case 0x01:                          /*  ２ＨＤ      */
                                adpnt[i] = hd8_fat[i];
                                break;
                }
                ++i;
        }
        com_code = 0x4c | drvno;
        com_prm.trsize.dtsz = 3;                            /*  転送データ・サイズのセット */
        com_prm.h = 0;
        com_prm.r = 0x02;                                   /*  実行セクタ番号をセット      */

        ret_st = WRDAT(com_code, &com_prm, &ret_prm);
                                                            /*  ライト・データ実行  */

        if(ret_st != 0x00)                                  /*  エラー発生？        */
        {
                ERRPR(ret_st);
                exit(0);
        }
        switch(meda)                                        /*  ﾒﾃﾞｨｱ別にFATｾｸﾀ(2)の実行ｾｸﾀをｾｯﾄ  */
        {
                case 0x00:                                  /*  ２ＤＤ      */
                        com_prm.r = 0x05;
                        break;
                case 0x01:                                  /*  ２ＨＤ      */
```

保守／廃止

(FORMAT71.C) p.10

```
                com_prm.r = 0x0b;
                break;
        }
        com_prm.trsize.dtsz = 3;
        ret_st = WRDAT(com_code, &com_prm, &ret_prm);
                                                /*  ライト・データ実行  */

        if(ret_st != 0x00)                      /*  エラー発生？        */
        {
                ERRPR(ret_st);
                exit(0);
        }

        /*****************************************************************/
        /*   ＦＡＴセクタの２セクタ目以降の書き込み(ﾃﾞｰﾀはすべて00H)    */
        /*****************************************************************/

        adpnt[0] = 0x00;                        /*  転送データ(1ﾊﾞｲﾄ)をメモリにセット   */
        sccoun = 0;                             /*  セクタ・カウンタをクリア    */
        switch(meda)
        {
                case 0x00:                      /*  ２ＤＤ      */
                        while(sccoun < 2)
                        {
                                com_prm.r = 3 + sccoun;
                                com_prm.trsize.dtsz = 1;        /*  転送データ・サイズをセット  */

                                ret_st = WRDAT(com_code, &com_prm, &ret_prm);
                                                        /*  ライト・データ実行  */

                                if(ret_st != 0x00)              /*  エラー発生？        */
                                {
                                        ERRPR(ret_st);
                                        exit(0);
                                }
                                com_prm.r = 6 + sccoun;
                                com_prm.trsize.dtsz = 1;        /*  転送データ・サイズをセット  */

                                ret_st = WRDAT(com_code, &com_prm, &ret_prm);
                                                        /*  ライト・データ実行  */

                                if(ret_st != 0x00)              /*  エラー発生？        */
                                {
                                        ERRPR(ret_st);
                                        exit(0);
                                }
                                ++sccoun;
                        }
                        break;

                case 0x01:                      /*  ２ＨＤ(18ｾｸﾀ)      */
                        while(sccoun < 8)
                        {
                                com_prm.r = 3 + sccoun;
                                com_prm.trsize.dtsz = 1;        /*  転送データ・サイズをセット  */

                                ret_st = WRDAT(com_code, &com_prm, &ret_prm);
                                                        /*  ライト・データ実行  */

                                if(ret_st != 0x00)              /*  エラー発生？        */
                                {
                                        ERRPR(ret_st);
                                        exit(0);
                                }
                                com_prm.r = 12 + sccoun;
                                com_prm.trsize.dtsz = 1;        /*  転送データ・サイズをセット  */

                                if((meda == 0x01) && (com_prm.r == 19))
                                                        /*  ２ＨＤの最終ＦＡＴセクタのとき      */
                                {
```

```
                                com_prm.h = 1;          /*  ヘッドをサイド1にセット     */
                                com_prm.r = 1;          /*  開始セクタを1セクタにセット         */
                        }
                        ret_st = WRDAT(com_code, &com_prm, &ret_prm);
                                                        /*  ライト・データ実行  */

                        if(ret_st != 0x00)              /*  エラー発生？        */
                        {
                                ERRPR(ret_st);
                                exit(0);
                        }
                        ++sccoun;
                }
                break;
        }


        /*********************************/
        /*   ＦＡＴセクタのベリファイ    */
        /*********************************/

        com_code = 0xc4 | drvno;
        com_prm.r = 0x02;
        com_prm.h = 0;
        switch(meda)
        {
                case 0x00:                              /*  ２ＤＤﾃｨｽｸのＦＡＴｾｸﾀ数のｾｯﾄ  */
                        com_prm.dtl = 6;
                        break;
                case 0x01:
                                                        /*  セクタ数のセット    */
                        com_prm.dtl = 18;
                        break;
        }
        ret_st = VERIFY(com_code, &com_prm, &ret_prm);
                                                        /*  ベリファイの実行    */

        if(ret_st != 0x00)                              /*  エラーの発生？      */
        {
                ++errcun;                               /*  Ｙｅｓ→ｴﾗｰ･ｶｳﾝﾀをｲﾝｸﾘﾒﾝﾄ   */
        }
        else
        {
                break;                                  /*  Ｎｏ→次の処理へ    */
        }
}
if(ret_st != 0x00)                                      /*  エラー発生？        */
{
        ERRPR(ret_st);
        exit(0);
}

        /*************************************/
        /*    ルート・ディレクトリのライト   */
        /*************************************/

errcun = 0;
while(errcun < 2)
{
        int k;
        k = 0;
        while(k < 512)                                  /*  ﾙｰﾄ･ﾃﾞｨﾚｸﾄﾘのﾃﾞｰﾀをﾒﾓﾘにｾｯﾄ  */
        {
                if((k % 32) == 0)                       /*  32ビットごとに00Hのデータが入る      */
                {
                        rtd_dat = 0x00;
                }
                else
                {
                        rtd_dat = 0xe5;                 /*  そのほかのデータはE5H        */
```

保守／廃止

(FORMAT71.C) p.12

```
        }
        adpnt[k] = rtd_dat;
        ++k;
}

com_code = 0xcc | drvno;

switch(meda)                                    /*  各ﾒﾃﾞｨｱごとにﾙｰﾄ･ﾃﾞｨﾚｸﾄﾘのｾｸﾀ数をｾｯﾄ  */
{
        case 0x00:                              /*  ２ＤＤﾃﾞｨｽｸのﾙｰﾄ･ﾃﾞｨﾚｸﾄﾘのｾｸﾀ数のｾｯﾄ  */
                rdcoun = 7;
                break;
        case 0x01:                              /*  ２ＨＤ(18ｾｸﾀ)ディスクのﾙｰﾄ･ﾃﾞｨﾚｸﾄﾘの  */
                                                /*  セクタ数のセット    */
                rdcoun = 14;
                break;
}

sccoun = 0;
while(sccoun < rdcoun)                          /*  ﾙｰﾄ･ﾃﾞｨﾚｸﾄﾘのセクタ数分のくり返し  */
{
        com_prm.h = 0x01;
        switch(meda)                            /*  各メディアのパラメータ設定  */
        {
                case 0x00:                      /*  ２ＤＤディスクのパラメータ設定      */
                        if(sccoun < 2)          /*  ﾙｰﾄ･ﾃﾞｨﾚｸﾄﾘの1,2ｾｸﾀ目(ｻｲﾄﾞ0)の設定  */
                        {
                                com_prm.h = 0x00;
                                com_prm.r = sccoun + 8;
                        }
                        else                    /*  ﾙｰﾄ･ﾃﾞｨﾚｸﾄﾘの3ｾｸﾀ目以降(ｻｲﾄﾞ1)の設定        */
                        {
                                com_prm.r = sccoun - 1;
                        }
                        break;
                case 0x01:                      /*  ２ＨＤディスクのパラメータ設定      */
                        com_prm.r = sccoun + 2;
                        break;
        }
        com_prm.trsize.dtsz = 512;              /*  転送データ・バイト数のセット  */

        ret_st = WRDAT(com_code, &com_prm, &ret_prm);
                                                /*  ライト・データ実行  */

        if(ret_st != 0x00)                      /*  エラー発生？        */
        {
                ERRPR(ret_st);
                exit(0);
        }
        ++sccoun;                               /*  ｾｸﾀ･ｶｳﾝﾀのインクリメント    */
}

/*****************************************/
/*  ルート・ディレクトリのベリファイ    */
/*****************************************/

com_code = 0xc4 | drvno;                        /*  ベリファイ用コマンド・コードをセット        */
com_prm.h = 0x00;
switch(meda)                                    /*  各メディアのパラメータ設定  */
{
        case 0x00:                              /*  ２ＤＤディスクのパラメータ設定      */
                com_prm.dtl = 7;
                com_prm.r = 8;
                break;
        case 0x01:                              /*  ２ＨＤディスクのパラメータ設定      */
                com_prm.dtl = 14;
                com_prm.h = 0x01;
                com_prm.r = 2;
                break;
}
```

保守／廃止

(FORMAT71.C) p.13

```
            ret_st = VERIFY(com_code, &com_prm, &ret_prm);
                                                        /*  ベリファイ実行      */
            if(ret_st != 0x00)                          /*  エラー発生？        */
            {
                    ++errcun;                           /*  Ｙｅｓ→ｴﾗｰ･ｶｳﾝﾀ,ｲﾝｸﾘﾒﾝﾄ    */
            }
            else
            {
                    break;                              /*  Ｎｏ→次の処理へ    */
            }
      }
      if(ret_st != 0x00)                                /*  エラー発生？        */
      {
            ERRPR(ret_st);
            exit(0);
      }

      outportb(DOR70, 0x0c);                            /*  モータの停止        */
      time_i = 0;                                       /*  タイマ割り込みフラグのクリア      */

      setvect(INTVEC, OLD_VEC);                         /*  割り込みベクタを復帰する    */

      printf("¥x1b[>51");                               /*  カーソルを表示する  */
      printf("¥x1b%s", EXECUTM);                        /*  カーソルをメッセージ表示位置へ    */
      printf("¥x1b[0J");
      printf("フォーマット終了¥n");                     /*  正常終了を表示する  */

}


            /*************************/
            /*                       */
            /* キー・コード取得関数  */
            /*                       */
            /*************************/

unsigned getkey(void)
{
      _AH = 0;
      geninterrupt(0x18);
      return(_AX);
}


            /*************************************************/
            /*                                               */
            /*  エラー表示処理      ＩＮ    : ret_st         */
            /*                      ＯＵＴ  : なし           */
            /*                                               */
            /*************************************************/

void ERRPR(unsigned char ret_st)
{

      printf("¥x1b[>51");                               /*  カーソルを表示する  */
      printf("¥x1b%s", MESS);
      printf("¥x1b[K");
      switch(ret_st)
      {
            case 0x01:
                    printf("ドライブの準備ができていません。¥n");
                    break;
            case 0x02:
            case 0x06:
            case 0x07:
            case 0x08:
            case 0x0a:
            case 0x0b:
                    printf("データ・エラーです。¥n");
```

(FORMAT71.C) p.14

```
                    break;
            case 0x03:
            case 0x04:
                    printf("シーク・エラーです。¥n");
                    break;
            case 0x05:
                    printf("セクタが見つかりません。¥n");
                    break;
            case 0x0c:
                    printf("書き込み禁止です。¥n");
                    break;
            defalut:
                    printf("エラーです。¥n");
                    break;
    }
    outportb(DOR70, 0x0c);                                  /*  モータの停止        */
    time_i = 0;                                             /*  タイマ割り込みフラグのクリア        */
    setvect(INTVEC, OLD_VEC);                               /*  割り込みベクタを復帰する    */
}
```

保守／廃止

３.３　垂直磁化記録方式4 MバイトFDD用デバイス・ドライバ編

３.３.１　BIOS部

BIOS部の説明については，３.２　１M/2 M バイトFDD用デバイス・ドライバ編と共通なので，**３.２.１ BIOS部**を参照してください。

３.３.２　LIO部

表３－19にLIOの機能を示します。

LIOのモジュール説明，変数については，３.２　１M/2 M バイトFDD用デバイス・ドライバ編と共通なので，**３.２.２　LIO部**を参照してください。

表３－19　LIO機能

モジュール名	機　　能
D_INT	FDCのイニシャライズと，接続されているFDDを調べます。パワーオン時に実行します。
MEDIAT	メディア・タイプ（2DD, 2HD, 2ED）を変更します。
SENSE	指定されたドライブの情報を得ます。
RECAL	指定されたシリンダ番号へヘッドを移動します。
SEEK	指定されたシリンダ数だけヘッドを移動します。
RSEEK	シリンダ０へヘッドを移動します。
RDDAT	DAM付きデータをリードします。
RDDDAT	DDAM付きデータをリードします。
RDID	トラック上の任意のID情報をリードします。
RDDAG	指定トラックのデータをリードし，トラックに書き込まれているデータの状態を診断します。
WRDAT	DAM付きデータをライトします。
WRDDAT	DDAM付きデータをライトします。
TRFORMAT	１トラック分フォーマットします。
VERIFY	指定したセクタのベリファイ（CRCチェック）します。
DRVSEL	指定したドライブ番号のDS信号とME信号をアクティブにします。
TIME	タイマの割り込みがあったことを示すフラグを立てます。

保守／廃止

３.３.３　デバイス・ドライバ

BUILD BPB処理以外の説明につきましては，３.２　１M/2 MバイトFDD用デバイス・ドライバ編と共通なので，**３.２.３　デバイス・ドライバ**を参照してください。

（１）デバイス・ドライバ・モジュール（BUILD BPB処理）の説明

ここではデバイス・ドライバのモジュールのBUILD BPB処理の説明をします。表３－20に示す内容に従って各モジュールを説明し，SPDチャートを示します。

表３－20　各モジュールの説明内容

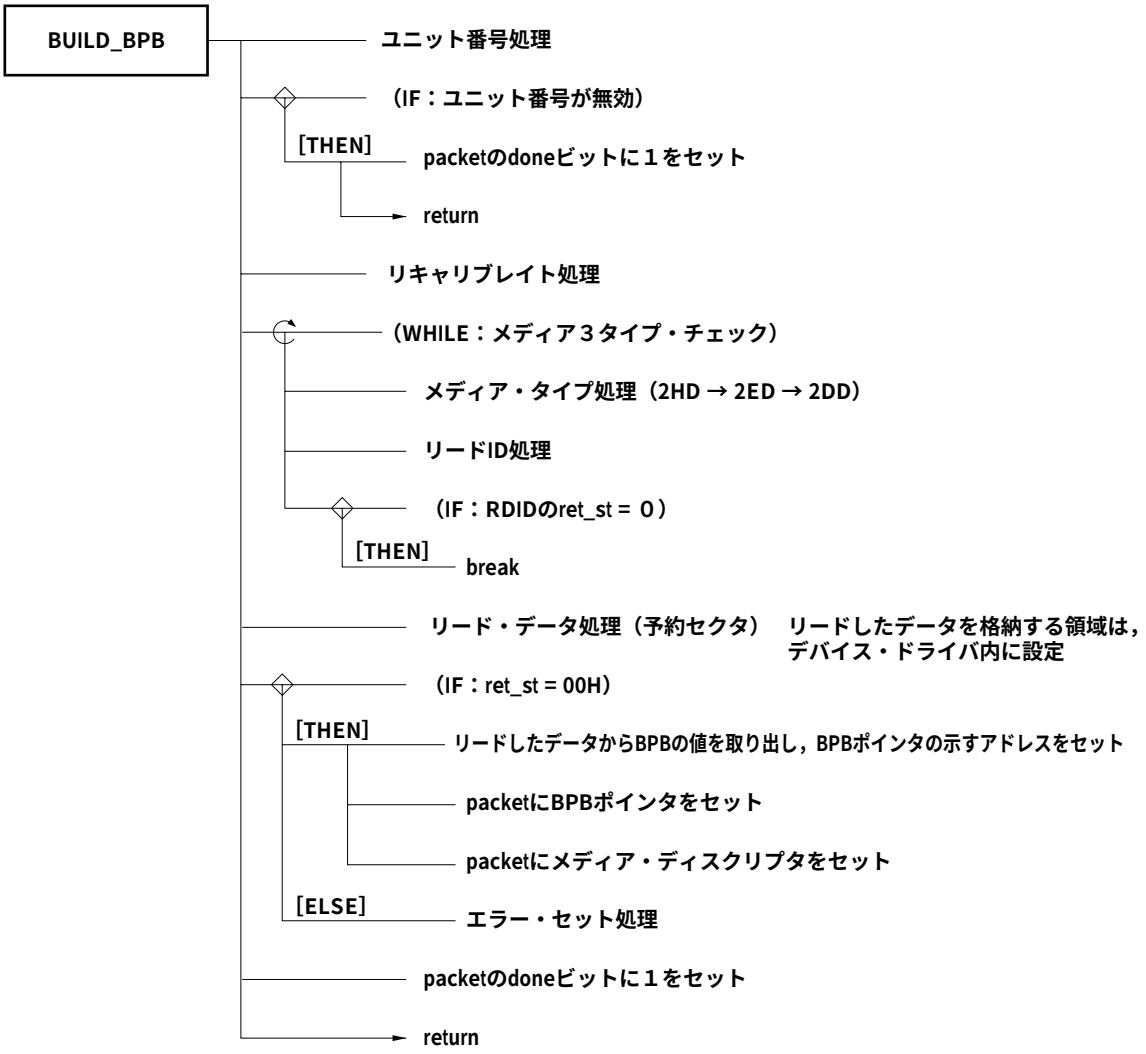
モジュール名	モジュール名を示します。
言　　語	関数がどの言語で作成されているかを示します。
入　　力	入力時の変数および，レジスタを示します。
出　　力	出力時の変数および，レジスタを示します。
［機　　能］ 関数の処理内容を示します。	

（ａ）BUILD BPB処理

モジュール名	BUILD_BPB
言　　語	C 言語
入　　力	packet, pro_seg
出　　力	packet
［機　　能］ （ｉ）リクエスト・ヘッダで示されるドライブの予約セクタ（31バイト分）をリードします。 （ⅱ）リクエスト・ヘッダで示されるドライブに対応するBPBポインタの内容に，リードした予約セクタの内容のBPBの値をセットします。 （ⅲ）コマンド・パケットにメディア・ディスクリプタをセットします。 （ⅳ）コマンド・パケットにdoneビットをセットします。	

保守／廃止

SPD-33


BUILD_BPB
ユニット番号処理
（IF：ユニット番号が無効）
[THEN]
packetのdoneビットに１をセット
return
リキャリブレイト処理
（WHILE：メディア３タイプ・チェック）
メディア・タイプ処理（2HD → 2ED → 2DD）
リードID処理
（IF：RDIDのret_st = ０）
[THEN]
break
リード・データ処理（予約セクタ）　リードしたデータを格納する領域は，
デバイス・ドライバ内に設定
（IF：ret_st = 00H）
[THEN]
リードしたデータからBPBの値を取り出し，BPBポインタの示すアドレスをセット
packetにBPBポインタをセット
packetにメディア・ディスクリプタをセット
[ELSE]
エラー・セット処理
packetのdoneビットに１をセット
return

保守／廃止

３.３.４　フォーマット・コマンド

ここではμPD72070用デバイス・ドライバを用いたMS-DOS用フォーマット・コマンドを説明します。

なおフォーマット・コマンドの表示，フォーマット・コマンドのリトライ処理，フォーマット・コマンドのプログラム構成，フォーマット・コマンド・モジュール説明については，３.２　１M/2 MバイトFDD用デバイス・ドライバ編と共通なので，**３.２.４　フォーマット・コマンド**を参照してください。

（１）フォーマット仕様

このフォーマット・コマンドは，次のフロッピィ・ディスクをサポートします。

●2DD（９セクタ／トラック）　720 Kバイト
●2HD（18セクタ／トラック）　1.44 M バイト
●2ED（36セクタ／トラック）　2.88 M バイト

2EDのフォーマット仕様を表３－21に示します。2DD，2HDのフォーマット仕様については，３.２　１M/2 M バイトFDD用デバイス・ドライバ編と共通なので，**表３－14　フォーマット仕様**を参照してください。

なおFATは12ビットFAT（標準フォーマット）でフォーマットを行います。

表３－21　2ED フォーマット仕様

項　　目	2ED
総トラック数	160
セクタ数／トラック	36
バイト数／セクタ	512
セクタ数／クラスタ	2
予約セクタ数	1
FAT数	2
ルート・ディレクトリのエントリ数	240
総論理セクタ数	5760
メディア・ディスクプリタ FAT　ID	F0H
セクタ数/FAT	9
ルート・ディレクトリ・セクタ数	15

次にフォーマット後の2EDのメディアの状態を説明します。

2EDのフォーマット後のセクタの割り当て状態を図３－16に示し，予約セクタ，FAT領域，ルート・ディレクトリ領域，およびデータ領域の内容を図３－17，図３－18，図３－19，図３－20に示します。

図３－16　2EDのセクタ割り当て

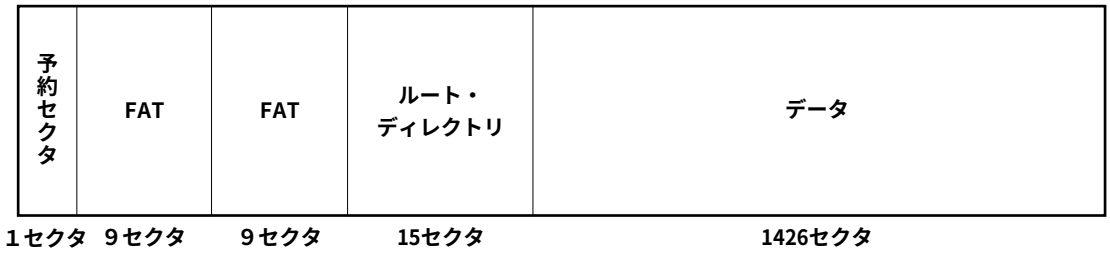


図３－17　2EDの予約セクタの内容

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
EB	1C	90	U_{55}	P_{50}	D_{44}	7_{37}	2_{32}	0_{30}	7_{37}	0_{30}	00	02	02	01	00
02	F0	00	80	16	F0	09	00	24	00	02	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	----------														00
⋮							⋮								⋮
00	----------														00

備考　書き込まれる値については，３.２　１M/2 MバイトFDD用デバイス・ドライバ編と共通なので，**表３－15　予約セクタの内容説明**を参照してください。

図３－18　2EDのFAT領域の内容

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
F0	FF	FF	00	00	00	00	00	00	00	00	00	00	00	00	00
00	----------														00
⋮							⋮								⋮
00	----------														00

図３－19　ルート・ディレクトリ領域の内容

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5
E5	E5	E5	----	----	----	----	----	----	----	----	----	----	----	----	E5
00	E5	----	----	----	----	----	----	----	----	----	----	----	----	----	E5
E5	----	----	----	----	----	----	----	----	----	----	----	----	----	----	E5
⋮							⋮								⋮
00	E5	----	----	----	----	----	----	----	----	----	----	----	----	----	E5
E5	----	----	----	----	----	----	----	----	----	----	----	----	----	----	E5

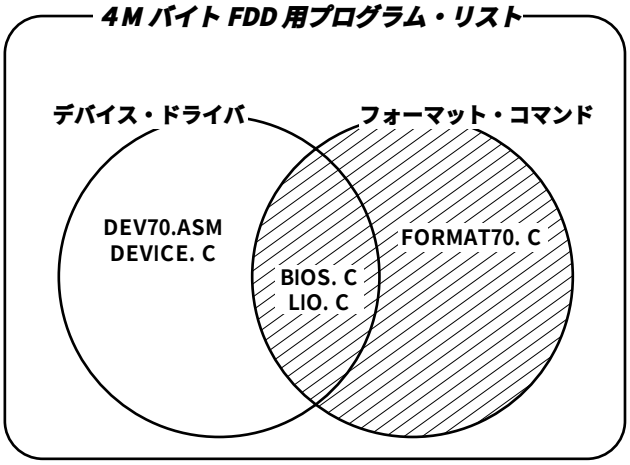
図３－20　データ領域の内容

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5
E5	----	----	----	----	----	----	----	----	----	----	----	----	----	----	E5
⋮							⋮								⋮
E5	----	----	----	----	----	----	----	----	----	----	----	----	----	----	E5

3.3.5　リ ス ト

次のプログラムのリストを示します。

●DEV70.ASM
●BIOS.C
●LIO.C
●DEVICE.C
●FORMAT70.C



●DEV70.ASM

(DEV70.ASM) p. 1

```
_TEXT SEGMENT BYTE PUBLIC 'CODE'
_TEXT ENDS

_DATA SEGMENT WORD PUBLIC 'DATA'
_DATA ENDS

_BSS SEGMENT WORD PUBLIC 'BSS'
_BSS ENDS

_PEND SEGMENT WORD PUBLIC 'PEND'
_PEND ENDS

DGROUP GROUP _TEXT, _DATA, _BSS, _PEND
      ASSUME CS:_TEXT, DS:DGROUP, SS:DGROUP, ES:DGROUP

_DATA SEGMENT WORD PUBLIC 'DATA'
;
;
_packet       DD      ?                    ;コマンド・パケットのポインタ・セーブ領域
STACKSAVE     DD      ?                    ;スタック・ポインタのセーブ領域
DGROUP@       DW      ?
_pro_off      DW      ?                    ;プログラム終了アドレスのセーブ領域
_pro_seg      DW      ?
              DB      1024*2 DUP(?)        ;
STACK_TOP     LABEL WORD                   ;スタックの先頭アドレス
;
_DATA ENDS
;
;
_PEND SEGMENT WORD PUBLIC 'PEND'
PROGEND       LABEL WORD                   ;プログラムの終了アドレス
_PEND ENDS
;
;
;
PUBLIC        _packet
PUBLIC        _pro_off
PUBLIC        _pro_seg
PUBLIC        DGROUP@
;
_TEXT SEGMENT BYTE PUBLIC 'CODE'
;
EXTRN _IO_REQ:NEAR                         ;Ｉ／Ｏリクエスト処理
;
ORG   0
;
;*********************************************************
;*                                                       *
;*              デバイス・ヘッダの構成                   *
;*                                                       *
;*********************************************************
;
header        LABEL   WORD
      DD      -1                           ;リンク情報
      DW      6000H                        ;デバイス属性
      DW      STRATEGY                     ;ストラテジ・エントリ・ポイント
      DW      INTERRUPT                    ;割り込みエントリ・ポイント
      DB      4                            ;ユニット数
;
;************************************************************************
;*                                                                      *
;*    ストラテジ・エントリ処理         ＩＮ    : BXﾚｼﾞｽﾀ, ESﾚｼﾞｽﾀ       *
;*                                     ＯＵＴ  : packet                *
;*                                                                      *
;************************************************************************
;
STRATEGY      PROC    FAR
;
      MOV     WORD PTR CS:_packet, BX      ;コマンド・パケットのポインタを待避する
      MOV     WORD PTR CS:_packet+2, ES
```

保守／廃止

(DEV70.ASM) p.2

```
      RET
;
STRATEGY      ENDP
;
;
;************************************************************************
;*                                                                      *
;*    割り込み処理エントリ処理          ＩＮ    : なし                  *
;*                                      ＯＵＴ  : pro_seg, pro_off      *
;*                                                                      *
;************************************************************************
;
INTERRUPT     PROC    FAR
;
      PUSH    AX                              ;全レジスタをスタックへ待避させる
      PUSH    BX
      PUSH    CX
      PUSH    DX
      PUSH    SI
      PUSH    DI
      PUSH    BP
      PUSH    DS
      PUSH    ES
;
      MOV     AX, CS                          ;ＤＳ＝ＣＳ
      MOV     DS, AX
      MOV     CS:DGROUP@, AX
      MOV     WORD PTR STACKSAVE, SP          ;ＭＳ－ＤＯＳのスタック・ポインタの待避
      MOV     WORD PTR STACKSAVE+2, SS
;
      CLI
      MOV     SS, AX                          ;デバイス・ドライバのスタック領域のセット
      MOV     SP, OFFSET DGROUP:STACK_TOP
      STI
;
      MOV     _pro_off, OFFSET DGROUP:PROGEND ;デバイス・ドライバの終了アドレスをpro_off, pro_segにセットする
      MOV     _pro_seg, CS
      CALL    _IO_REQ                         ;Ｉ／Ｏリクエスト処理を呼び出す
;
      CLI
      MOV     SP, WORD PTR STACKSAVE          ;ＭＳ－ＤＯＳのスタックを復帰させる
      MOV     SS, WORD PTR STACKSAVE+2
      STI
;
      POP     ES                              ;レジスタを復帰させる
      POP     DS
      POP     BP
      POP     DI
      POP     SI
      POP     DX
      POP     CX
      POP     BX
      POP     AX
      RET
;
INTERRUPT     ENDP
;
_TEXT ENDS
;
END
```

●BIOS.C

(BIOS.C) p.1

```
#include<dos.h>

#define             OCW2          0x0000                    /*  割り込みコントローラのOCW2のIOアドレス  */
#define             STR70         0x00d8                    /*  μPD72070のｽﾃｰﾀｽ･ﾚｼﾞｽﾀのIOアドレス  */
#define             DTR70         0x00da                    /*  μPD72070のﾃﾞｰﾀ･ﾚｼﾞｽﾀのIOアドレス  */



struct retpra
{                                                           /*  ﾘﾀｰﾝ･ｽﾃｰﾀｽ･ﾊﾟﾗﾒｰﾀの構造体  */
      unsigned char c;                                      /*  実行終了後のシリンダ番号  */
      unsigned char h;                                      /*  実行終了後のヘッド番号  */
      unsigned char r;                                      /*  実行終了後のセクタ番号  */
      unsigned char n;                                      /*  1セクタのデータ長  */
};
      struct sr                                             /*  シーク・リザルトの構造体  */
{
      unsigned char dr_st;                                  /*  シーク・コマンド実行後のﾘｻﾞﾙﾄ･ｽﾃｰﾀｽ  */
      unsigned char dr_pcn;                                 /*  シーク・コマンド終了後のヘッドの位置  */
};


void exit(int);
void COMMAND(unsigned char *, char);
void interrupt INTRPT(void);
char RESULT(void);


            /*****************************************************************/
            /*                                                               */
            /*  コマンド・ライト処理        ＩＮ    : com_dat[], com_num     */
            /*                              ＯＵＴ  : なし                   */
            /*                                                               */
            /*****************************************************************/

struct sr seek_r[4];
volatile unsigned char int_s;
volatile unsigned char ready;
unsigned char rslt_pt[7];

void COMMAND(unsigned char *com_pt, char com_num)
{
      unsigned char fdcst, dat;
      char count1;

      count1 = 0;                                           /*  カウンタのクリア  */
      while(count1 < com_num)                               /*  パラメータの転送ループ  */
      {
            do
            {
                  fdcst = inportb(STR70);                 /*  μPD72020のｽﾃｰﾀｽ･ﾚｼﾞｽﾀをリードする  */
                  fdcst = fdcst & 0xc0;                   /*  ｽﾃｰﾀｽ･ﾚｼﾞｽﾀのRQMビットとDIOビットのチェック */
            }
            while(fdcst != 0x80);                         /*  RQMビット＝1,DIOビット＝0になるまでくり返し */

            dat = *(com_pt + count1);
            outportb(DTR70, dat);                         /*  com_dat[count1]の内容をμPD72070の  */
                                                          /*  ﾃﾞｰﾀ･ﾚｼﾞｽﾀにライトする  */
            ++count1;                                     /*  カウンタをインクリメント  */
      }
}


            /*****************************************************/
            /*                                                   */
            /*  割り込み処理        ＩＮ    : なし               */
            /*                      ＯＵＴ  : int_s, seek_r[], ready */
            /*                                                   */
            /*****************************************************/
```

(BIOS.C) p.2

```
void interrupt INTRPT(void)
{
      unsigned char fdcst, rsltst0, hedpos;

      int_s = 0x01;                                          /*  割り込み発生のフラグを立てる          */
      do
      {
            fdcst = inportb(STR70);                          /*  μPD72070のｽﾃｰﾀｽ･ﾚｼﾞｽﾀのリード       */
      }
      while(fdcst < 0x80);                                   /*  ｽﾃｰﾀｽ･ﾚｼﾞｽﾀのRQMﾋﾞｯﾄ=1になるまで繰り返す  */

      if(fdcst < 0xc0)                                       /*  データ転送方向の判断          */
      {                                                      /*  転送方向：ﾎｽﾄ→FDC  */
            outportb(DTR70, 0x08);                           /*  SENSE INTERRUPT STATUSコマンドの発行  */
            do
            {
                  fdcst = inportb(STR70);                    /*  ｽﾃｰﾀｽ･ﾚｼﾞｽﾀのリード */
            }
            while(fdcst < 0xc0);                             /*  データの転送が許可されたか  */
            rsltst0 = inportb(DTR70);                        /*  ﾃﾞｰﾀ･ﾚｼﾞｽﾀをリードしてﾘｻﾞﾙﾄ･ｽﾃｰﾀｽ0にセット  */
            if((rsltst0 & 0xc0) != 0x80)                     /*  無効コマンドの判定  */
            {
                  do                                         /*  無効コマンドではないときの処理       */
                  {
                        fdcst = inportb(STR70);              /*  ｽﾃｰﾀｽ･ﾚｼﾞｽﾀのリード */
                  }
                  while(fdcst < 0xc0);                       /*  データの転送が許可されたか  */
                  hedpos = inportb(DTR70);                   /*  ﾃﾞｰﾀ･ﾚｼﾞｽﾀをリードしてヘッドの位置を得る     */
                  if((rsltst0 & 0xc0) == 0xc0)               /*  割り込み要因の判断  */
                  {                                          /*  割り込み要因がドライブの状態遷移によるとき  */
                        int_s = 0x00;
                        if((rsltst0 & 0x08) == 0x00)         /*  ノット・レディ→レディ       */
                        {
                              switch(rsltst0 & 0x03)
                              {
                                    case 0x00:               /*  ドライブ０  */
                                          ready = ready | 0x01;
                                          break;
                                    case 0x01:               /*  ドライブ１  */
                                          ready = ready | 0x02;
                                          break;
                                    case 0x02:               /*  ドライブ２  */
                                          ready = ready | 0x04;
                                          break;
                                    case 0x03:               /*  ドライブ３  */
                                          ready = ready | 0x08;
                                          break;
                              }
                        }
                  }
                  else                                       /*  割り込み要因がｼｰｸ系のＥﾌｪｰｽﾞ終了によるとき  */
                  {                                          /*  ｼｰｸ･ﾘｻﾞﾙﾄに結果をセット      */
                        seek_r[(rsltst0 & 0x03)].dr_st = rsltst0;
                        seek_r[(rsltst0 & 0x03)].dr_pcn = hedpos;
                  }
            }
      }
      else                                                   /*  転送方向：FDC→ホストのとき */
      {
            RESULT();                                        /*  ﾘｻﾞﾙﾄ･ﾓｼﾞｭｰﾙ呼びだし          */
      }
      outportb(OCW2, 0x63);                                  /*  ＦＩコマンド発行     */
}
```

(BIOS.C) p.3

```
        /*****************************************************************/
        /*                                                               */
        /*  リザルト・リード処理          ＩＮ    : なし                 */
        /*                                ＯＵＴ  : rslt_pt[], rslt_num  */
        /*                                                               */
        /*****************************************************************/


char RESULT(void)
{
    unsigned char fdcst;
    char rslt_num;

    rslt_num = 0;                                   /*  ﾘｻﾞﾙﾄ･ﾅﾝﾊﾞのクリア  */

    while(1)                                        /*  ループ      */
    {
        do
        {
            fdcst = inportb(STR70);                 /* fdcstにμPD72070のｽﾃｰﾀｽ･ﾚｼﾞｽﾀの内容を読み込む*/
        }
        while(fdcst < 0x80);                        /*  ｽﾃｰﾀｽ･ﾚｼﾞｽﾀのRQMﾋﾞｯﾄ=1になるまでﾘｰﾄﾞする    */

        if((fdcst & 0x40) == 0x00)                  /*  ｽﾃｰﾀｽ･ﾚｼﾞｽﾀのDIOビット=0のとき      */
        {
            break;                                  /*  ループをぬける      */
        }
        rslt_pt[rslt_num] = inportb(DTR70);         /*  μPD72070のﾃﾞｰﾀ･ﾚｼﾞｽﾀをリードして   */
                                                    /*  rslt_pt+rslt_num(ｱﾄﾞﾚｽ)にセット          */
        ++rslt_num;                                 /*  ﾘｻﾞﾙﾄ･ﾅﾝﾊﾞをインクリメント  */
    }
    return(rslt_num);                               /*  リターン値：ﾘｻﾞﾙﾄ･ﾅﾝﾊﾞ      */
}
```

保守／廃止

●LIO.C

(LIO.C) p.1

```
#include<dos.h>


#define             OCW2            0x0000          /*  割り込みコントローラのOCW2のIOアドレス  */
#define             STR70           0x00d8          /*  μPD72070のｽﾃｰﾀｽ･ﾚｼﾞｽﾀのIOアドレス  */
#define             DTR70           0x00da          /*  μPD72070のﾃﾞｰﾀ･ﾚｼﾞｽﾀのIOアドレス  */
#define             DOR70           0x00d4          /*  μPD72070のﾃﾞｨｼﾞﾀﾙ･ｱｳﾄﾌﾟｯﾄ･ﾚｼﾞｽﾀのIOｱﾄﾞﾚｽ*/
#define             DRR70           0x00d8          /*  μPD72070のﾃﾞｰﾀ･ﾚｰﾄ･ﾚｼﾞｽﾀのIOアドレス*/
#define             CCR70           0x00de          /*  μPD72070のｺﾝﾌｨｷﾞｭﾚｰｼｮﾝ･ｺﾝﾄﾛｰﾙ･ﾚｼﾞｽﾀのIOｱﾄﾞﾚｽ*/
#define             BSWRP1          0x02d0          /*  ﾊﾞｳﾝﾀﾞﾘ･ｽｷｬﾝ回路のﾗｲﾄ･ﾎﾟｰﾄ1のIOアドレス*/
#define             IRMR            0x0002          /*  割り込みコントローラのﾏｽｸ･ﾚｼﾞｽﾀのIOｱﾄﾞﾚｽ*/
#define             DADDR           0x000d          /*  DMAｺﾝﾄﾛｰﾗのｱﾄﾞﾚｽ･ﾚｼﾞｽﾀのIOアドレス*/
#define             DCOUNT          0x000f          /*  DMAｺﾝﾄﾛｰﾗのｶｳﾝﾄ･ﾚｼﾞｽﾀのIOアドレス*/
#define             DWRSMR          0x0015          /*  DMAｺﾝﾄﾛｰﾗのﾗｲﾄ･ｼﾝｸﾞﾙ･ﾏｽｸ･ﾚｼﾞｽﾀのIOｱﾄﾞﾚｽ*/
#define             DMODER          0x0017          /*  DMAｺﾝﾄﾛｰﾗのﾓｰﾄﾞ･ﾚｼﾞｽﾀのIOアドレス*/
#define             DCLBFF          0x0019          /*  DMAｺﾝﾄﾛｰﾗのｸﾘｱ･ﾊﾞｲﾄ･ﾎﾟｲﾝﾀ･FFのIOｱﾄﾞﾚｽ*/
#define             DBNKR           0x0025          /*  DMAｺﾝﾄﾛｰﾗのﾊﾞﾝｸ･ﾚｼﾞｽﾀのIOアドレス*/
#define             DBAAIM          0x0029          /*  DMAｺﾝﾄﾛｰﾗのﾊﾞﾝｸ･ｱﾄﾞﾚｽ･ｵｰﾄ･ｲﾝｸﾘﾒﾝﾄ･*/
                                                    /*  ﾓｰﾄﾞ･ﾚｼﾞｽﾀのIOアドレス      */
#define             INTVEC          0x0b            /*  割り込みベクタ番号  */


struct compra
{                                                   /*  コマンド・パラメータの構造体        */
      union                                         /*  転送バイト数の共用体        */
      {
            unsigned int dtsz;                      /*  転送データのバイト数(int型) */
            struct                                  /*  int型をchar型にわける構造体 */
            {
                  unsigned char kai;                /*  下位８ビット分      */
                  unsigned char jyoui;              /*  上位８ビット分      */
            } int_hen;

      } trsize;
      unsigned int dt_off;                          /*  転送ﾃﾞｰﾀのｵﾌｾｯﾄ･ｱﾄﾞﾚｽ      */
      unsigned int dt_seg;                          /*  転送ﾃﾞｰﾀのｾｸﾞﾒﾝﾄ･ｱﾄﾞﾚｽ      */
      unsigned char c;                              /*  シリンダ番号        */
      unsigned char h;                              /*  ヘッド番号  */
      unsigned char r;                              /*  開始セクタ番号      */
      unsigned char n;                              /*  １セクタのデータ長  */
      unsigned char dtl;                            /*  1ｾｸﾀあたりの処理ﾃﾞｰﾀ長     */
      unsigned char gsl;                            /*  GAP3の読み飛ばしバイト数    */
      unsigned char eot;                            /*  １トラックのセクタ数        */
      unsigned char d;                              /*  ﾌｫｰﾏｯﾄの書きこみﾃﾞｰﾀ･ﾊﾟﾀｰﾝ  */
};

struct retpra
{                                                   /*  ﾘﾀｰﾝ･ｽﾃｰﾀｽ･ﾊﾟﾗﾒｰﾀの構造体   */
      unsigned char c;                              /*  実行終了後のシリンダ番号    */
      unsigned char h;                              /*  実行終了後のヘッド番号      */
      unsigned char r;                              /*  実行終了後のセクタ番号      */
      unsigned char n;                              /*  １セクタのデータ長  */
};
      struct sr                                     /*  シーク・リザルトの構造体    */
{
      unsigned char dr_st;                          /*  シーク・コマンド実行後のﾘｻﾞﾙﾄ･ｽﾃｰﾀｽ */
      unsigned char dr_pcn;                         /*  シーク・コマンド終了後のヘッドの位置  */
};

struct form                                         /*  フォーマット時のＩＤ部の書き込みデータの  */
                                                    /*  構造体  */
{
      unsigned char c;                              /*  シリンダ番号        */
      unsigned char h;                              /*  ヘッド番号  */
      unsigned char r;                              /*  セクタ番号  */
      unsigned char n;                              /*  セクタあたりのフォーマット・バイト数  */
};

struct fpnt                                         /*  ｆａｒポインタをセグメントとオフセットに */
                                                    /*  分割する構造体  */
```

保守／廃止

(LIO.C) p.2

```
{
      unsigned int adoff;
      unsigned int adseg;
};

union fapnt                                                    /*  ｆａｒポインタの共用体宣言  */
{
      unsigned long pnt;                                       /*  ｆａｒポインタ      */
      struct fpnt pon;                                         /*  オフセットとセグメント      */
};


void exit(int);
void COMMAND(unsigned char *, char);
void interrupt INTRPT(void);
void D_INT(void);
char RESULT(void);
void MEDIAT(unsigned char);
unsigned char RECAL(unsigned char);
unsigned char SEEK(unsigned char, struct compra *);
unsigned char RSEEK(unsigned char, struct compra *);
unsigned char RDDAT(unsigned char, struct compra *, struct retpra *);
unsigned char RDDDAT(unsigned char, struct compra *, struct retpra *);
unsigned char RDID(unsigned char, struct retpra *);
unsigned char RDDAG(unsigned char, struct compra *, struct retpra *);
unsigned char TRFORMAT(unsigned char, struct compra *, struct retpra *);
unsigned char WRDAT(unsigned char, struct compra *, struct retpra *);
unsigned char WRDDAT(unsigned char, struct compra *, struct retpra *);
unsigned char VERIFY(unsigned char, struct compra *, struct retpra *);
void DRVSEL(unsigned char);
unsigned getkey(void);                                         /*  キー・コード取得関数  */
void interrupt TIME(void);



extern unsigned char int_s;
extern struct sr seek_r[];
extern unsigned char rslt_pt[];

unsigned char d_exist;
unsigned char time_i;

            /*************************************************************/
            /*                                                           */
            /*  デバイス・イニシャライズ処理        ＩＮ    :なし        */
            /*                                      ＯＵＴ  :d_exist     */
            /*                                                           */
            /*************************************************************/

void D_INT(void)
{
      unsigned char imrs, com_code, ret_st, drchk, bswpl, irr, pcn_b, st_r;
      char dcoun;
      union fapnt vector;

      d_exist = 0x00;                                          /*  d_existの初期化     */
      imrs = inportb(IRMR);                                    /*  割り込みｺﾝﾄﾛｰﾗのﾏｽｸ･ﾚｼﾞｽﾀをリードする  */
      imrs = imrs | 0x08;
      outportb(IRMR, imrs);                                    /*  ＩＲ３をマスクする  */

      vector.pnt = (unsigned long)INTRPT;                      /*  割り込みモジュールのアドレスを共用体へｾｯﾄ  */

      poke(0x0000, (unsigned int)(INTVEC * 4), vector.pon.adoff);
                                                               /*  割り込みﾓｼﾞｭｰﾙのｵﾌｾｯﾄ･ｱﾄﾞﾚｽをﾍﾞｸﾀにｾｯﾄ*/

      poke(0x0000, (unsigned int)(INTVEC * 4 + 2), vector.pon.adseg);
                                                               /*  割り込みﾓｼﾞｭｰﾙのｾｸﾞﾒﾝﾄ･ｱﾄﾞﾚｽをﾍﾞｸﾀにｾｯﾄ*/
      outportb(DWRSMR, 0x07);                                  /*  DMAｺﾝﾄﾛｰﾗのチャネル3をマスクする     */
      outportb(DMODER, 0x63);                                  /*  DMAｺﾝﾄﾛｰﾗのチャネル3の転送方向を初期化  */
      outportb(DBAAIM, 0x0f);                                  /*  DMAｺﾝﾄﾛｰﾗのチャネル3のﾊﾞﾝｸ･ﾓｰﾄﾞを16MBに設定*/
```

(I10.C) p.3

```
bswpl = inportb(BSWRP1);                        /*  ﾗｲﾄ･ﾎﾟｰﾄ1をリードする       */
bswpl = bswpl & 0xdf;                           /*  INT出力をマスクする */
outportb(BSWRP1, bswpl);
outportb(DOR70, 0x1c);                          /*  μPD72070の初期化   */
outportb(DRR70, 0x80);                          /*  μPD72070のソフトウエア・リセット   */
outportb(DRR70, 0x1c);                          /*  μPD72070の転送レートの初期化       */
outportb(CCR70, 0x00);                          /*  μPD72070の転送レートの初期化       */
bswpl = inportb(BSWRP1);                        /*  ﾗｲﾄ･ﾎﾟｰﾄ1をリードする       */
bswpl = bswpl | 0x20;                           /*  INT出力をマスクを解除する   */
outportb(BSWRP1, bswpl);
do
{
        outportb(OCW2, 0x0a);                   /*  割り込みコントローラのIRRのリード   */
        irr = inportb(OCW2);
}while((irr & 0x08) == 0x00);                   /*  IR3がｱｸﾃｨﾌﾞになるまで繰り返し       */
pcn_b = 0;
while(pcn_b != 0x80)                            /*  発行したSENSE INTERRUPT STATUSが    */
                                                /*  INVALID COMMANDになるまで繰り返す   */
{
        do
        {
                st_r = inportb(STR70);          /*  μPD72070のｽﾃｰﾀｽ･ﾚｼﾞｽﾀのリード      */
        }
        while(st_r != 0x80);
        outportb(DTR70, 0x08);                  /*  SENSE INTERRUPT STATUSコマンドの発行  */
        do
        {
                st_r = inportb(STR70);          /*  μPD72070のｽﾃｰﾀｽ･ﾚｼﾞｽﾀのリード      */
        }
        while((st_r & 0xf0) != 0xd0);
        while((st_r & 0xc0) != 0x80)            /*  μPD72070のｽﾃｰﾀｽ･ﾚｼﾞｽﾀが80Hになるまで繰り返す*/
        {
                pcn_b = inportb(DTR70);         /*  μPD72070のﾃﾞｰﾀ･ﾚｼﾞｽﾀのリード       */
                do
                {
                        st_r = inportb(STR70);  /*  μPD72070のｽﾃｰﾀｽ･ﾚｼﾞｽﾀのリード      */
                }
                while((st_r & 0x80) != 0x80);
        }
}
imrs = inportb(IRMR);                           /*  割り込みｺﾝﾄﾛｰﾗのﾏｽｸ･ﾚｼﾞｽﾀをリードする  */
imrs = imrs & 0xf7;
outportb(IRMR, imrs);                           /*  ＩＲ３のマスク解除  */

com_code = 0x01;                                /*  メディアを２ＨＤにセット    */
MEDIAT(com_code);                               /*  ﾒﾃﾞｨｱ･ﾀｲﾌﾟ･ﾓｼﾞｭｰﾙ呼びだし(2HDﾃﾞｨｽｸ)*/

dcoun = 0;
bswpl = inportb(BSWRP1);                        /*  ﾗｲﾄ･ﾎﾟｰﾄ1をリードする       */
bswpl = bswpl | 0x60;
outportb(BSWRP1, bswpl);                        /*  アクティブ・レディをＯＮにする      */

while(dcoun <= 3)                               /*  ドライブの接続チェック       */
{

        com_code = dcoun;                       /*  dcounドライブにリキャリブレイト・コマンドを */
                                                /*  発行する          */
        ret_st = RECAL(com_code);
        drchk = 0x01 << dcoun;
        if(ret_st == 0x00)                      /*  正常終了    */
        {
                d_exist = d_exist | drchk;      /*  ドライブありをd_existにセット       */
        }
        else                                    /*  異常終了    */
        {
                drchk = ~drchk;
                d_exist = d_exist & drchk;      /*  ドライブ未接続をd_existにセット     */
        }
        ++dcoun;                                /*  ドライブ番号をインクリメント        */
}
```

(L10.C) p.4

```
	bswpl = inportb(BSWRP1);                                /*  ﾗｲﾄ･ﾎﾟｰﾄ1をリードする       */
	bswpl = (bswpl & 0xbf) | 0x20;
	outportb(BSWRP1, bswpl);                                /*  アクティブ・レディをＯＦＦにする    */

}



	/************************************************************/
	/*                                                          */
	/*  メディア・タイプ・セレクト処理      ＩＮ    : com_code  */
	/*                                      ＯＵＴ  : なし      */
	/*                                                          */
	/************************************************************/


void MEDIAT(unsigned char com_code)
{
	unsigned char deccom, com_dat[4];
	char com_num;

	deccom = 0x03 & com_code;                               /*  メディア・タイプをcom_codeより取りだす  */
	switch(deccom)                                          /*  ﾒﾃﾞｨｱ･ﾀｲﾌﾟによって転送レートをｾｯﾄする  */
	{
		case 0x00:
			outportb(CCR70, 0x02);                  /*  ２ＤＤディスクのとき(250kbps)       */
			break;
		case 0x01:
			outportb(CCR70, 0x00);                  /*  ２ＨＤディスクのとき(500kbps)       */
			break;
		default:
			outportb(CCR70, 0x03);                  /*  ２ＥＤディスクのとき(1.0Mbps)       */
			break;
	}

		/************************************/
		/*  SPECIFYコマンドのデータ・セット */
		/************************************/

	com_dat[0] = 0x03;
	switch(deccom)
	{

		case 0x00:
			com_dat[1] = 0xd1;                      /*  ２ＤＤディスクのときのSRT,HUTのセット  */
			com_dat[2] = 0x40;                      /*  ２ＤＤディスクのときのHLTのセット   */
			break;
		case 0x01:
			com_dat[1] = 0xa1;                      /*  ２ＨＤディスクのときのSRT,HUTのセット  */
			com_dat[2] = 0x80;                      /*  ２ＨＤディスクのときのHLTのセット   */
			break;
		default:
			com_dat[1] = 0x01;                      /*  ２ＥＤディスクのときのSRT,HUTのセット  */
			com_dat[2] = 0xfe;                      /*  ２ＥＤディスクのときのHLTのセット   */
			break;
	}
	com_num = 0x03;
	COMMAND(&com_dat[0], com_num);                          /*  コマンド・モジュールの呼びだし       */

		/************************************/
		/*  CONFIGUREコマンドのデータセット */
		/************************************/

	com_dat[0] = 0x13;
	com_dat[1] = 0x00;
	if( com_code >= 0x04)                                   /*  シーク動作の設定    */
	{
		com_dat[2] = 0x68;                              /*  リード／ライト系コマンドでシークを伴う  */
	}
	else
	{
```

保守／廃止

(L10.C) p.5

```
            com_dat[2] = 0x28;                                  /*  リード／ライト系コマンドでシークを伴わない  */
      }
      com_dat[3] = 0xff;                                        /*  書き込み補償開始ﾄﾗｯｸの設定  */
      com_num = 0x04;
      COMMAND(&com_dat[0], com_num);                            /*  コマンド・モジュール呼びだし        */

            /***********************************************/
            /*  PERPENDICULAR MODEコマンドのデータ・セット  */
            /***********************************************/

      com_dat[0] = 0x12;
      if(deccom == 0x02)                                        /*  メディアが２ＥＤのとき      */
      {
            com_dat[1] = 0x03;                                  /*  垂直磁化記録方式(1Mbps)にセット     */
      }
      else                                                      /*  メディアが２ＥＤ以外のとき  */
      {
            com_dat[1] = 0x01;                                  /*  垂直磁化記録方式以外にセット        */
      }
      com_num = 0x02;
      COMMAND(&com_dat[0], com_num);                            /*  コマンド・モジュール呼びだし        */

            /***********************************************/
            /*  SELECT DRIVE TYPEコマンドのデータ・セット  */
            /***********************************************/

      com_dat[0] = 0x32;
      if(deccom > 0x02)                                         /*  メディアが13MBFDのとき      */
      {
            com_dat[1] = 0x01;
      }
      else
      {
            com_dat[1] = 0x00;                                  /*  メディアが4MBFD以下のとき   */
      }
      com_num = 0x02;
      COMMAND(&com_dat[0], com_num);                            /*  コマンド・モジュール呼びだし        */
}

            /*****************************************************/
            /*                                                   */
            /*  ドライブ・センス処理        ＩＮ   : com_code    */
            /*                              ＯＵＴ : sense_r     */
            /*                                                   */
            /*****************************************************/


void SENSE(unsigned char com_code)
{
      unsigned char com_dat[2], com_num, sense_r;

      com_dat[0] = 0x04;
      com_dat[1] = com_code;
      com_num = 2;
      COMMAND(&com_dat[0], com_num);                            /*  コマンド・モジュール呼び出し        */
      RESULT();                                                 /*  直接リザルト・モジュールで結果を受け取る  */
      sense_r = rslt_pt[0];                                     /*  リザルト・モジュールで受け取ったデータを  */
                                                                /*  sense_rにセットする     */
      return;
}


            /*************************************************************/
            /*                                                           */
            /*  リキャリブレイト処理        ＩＮ    : com_code           */
            /*                              ＯＵＴ  : ret_st, ret_prm    */
            /*                                                           */
            /*************************************************************/
```

(LIO.C) p.6

```
unsigned char RECAL(unsigned char com_code)
{
      signed char retryc, com_num;
      unsigned char com_dat[2], ret_st, coml_code, drn;
      struct compra coml_prm;

      drn = com_code & 0x03;                                    /*  ドライブ番号の抽出  */
      retryc = com_code >> 2;                                   /*  リトライ・カウンタのセット      */
      while(retryc >= 0)
      {
            com_dat[0] = 0x07;
            com_dat[1] = com_code & 0x03;
            com_num = 0x02;
            DRVSEL(com_dat[1]);
            COMMAND(&com_dat[0], com_num);
            while(int_s != 0x01);
            int_s = 0x00;
            if(seek_r[drn].dr_st < 0x40)
            {
                  ret_st = 0x00;                                /*  コマンド正常終了    */
            }
            else
            {
                  if(seek_r[drn].dr_st < 0x70)
                  {
                        ret_st = 0x01;                          /*  ドライブのノット・レディ    */
                  }
                  else
                  {
                        ret_st = 0x0d;                          /*  リキャリブレイト・エラー    */
                  }
            }
            if(ret_st == 0x0d)                                  /*  リキャリブレイト・エラーのときの処理  */
            {
                  coml_code = 0x04 | com_dat[1];                /*  20ｼﾘﾝﾀﾞ求心方向へシークしてからリトライ  */
                  coml_prm.c = 20;
                  RSEEK(coml_code, &coml_prm);
                  --retryc;
            }
            else                                                /*  リキャリブレイト・エラー以外のエラー又は  */
                                                                /*  正常終了のとき  */
            {
                  break;                                        /*  そのまま終了        */
            }
      }
      return ret_st;
}


            /*****************************************************/
            /*                                                   */
            /*  シーク処理          ＩＮ    : com_code, com_prm  */
            /*                      ＯＵＴ  : ret_st, ret_prm    */
            /*                                                   */
            /*****************************************************/


unsigned char SEEK(unsigned char com_code, struct compra *com_prm)
{
      unsigned char com_dat[3], ret_st;
      char com_num;

      DRVSEL(com_code);
      com_dat[0] = 0x0f;                                        /*  ｼｰｸ･ｺﾏﾝﾄﾞ用データのセット  */
      com_dat[1] = com_code;
      com_dat[2] = com_prm->c;
      com_num = 0x03;
      COMMAND(&com_dat[0], com_num);                            /*  ｺﾏﾝﾄﾞ･ﾓｼﾞｭｰﾙ呼び出し        */

      while(int_s != 0x01);                                     /*  コマンド終了割り込み待ち    */
```

(L10.C) p.7

```
      int_s = 0x00;
      if(seek_r[com_code].dr_st < 0x40)                          /*  コマンドの終了結果の判断    */
      {
            ret_st = 0x00;                                       /*  コマンド正常終了    */
      }
      else
      {
            ret_st = 0x01;                                       /*  ドライブのノット・レディ    */
      }
      return ret_st;
}


            /************************************************************/
            /*                                                          */
            /*  リラティブ・シーク処理      ＩＮ    : com_code, com_prm   */
            /*                              ＯＵＴ  : ret_st, ret_prm     */
            /*                                                          */
            /************************************************************/


unsigned char RSEEK(unsigned char com_code, struct compra *com_prm)
{
      unsigned char com_dat[3], ret_st, drn;
      char com_num;

      drn = com_code & 0x03;                                     /*  ドライブ番号の抽出  */
      if((com_code & 0x04) == 0x04)                              /*  ﾘﾗﾃｨﾌﾞ･ｼｰｸ用データのセット      */
      {
            com_dat[0] = 0xcf;                                   /*  求心方向へシークするデータ  */
      }
      else
      {
            com_dat[0] = 0x8f;                                   /*  遠心方向へシークするデータ  */
      }
      com_dat[1] = com_code & 0x03;
      com_dat[2] = com_prm->c;
      com_num = 0x03;
      DRVSEL(com_dat[1]);                                        /*  ドライブ・セレクト信号の選択        */
      COMMAND(&com_dat[0], com_num);                             /*  ｺﾏﾝﾄﾞ･ﾓｼﾞｭｰﾙ呼び出し        */

      while(int_s != 0x01);                                      /*  割り込み待ち        */
      int_s = 0x00;
      if(seek_r[drn].dr_st < 0x40)                               /*  コマンドの終了結果の判断    */
      {
            ret_st = 0x00;                                       /*  コマンド正常終了    */
      }
      else
      {
            ret_st = 0x01;                                       /*  ドライブのノット・レディ    */
      }
      return ret_st;
}


            /************************************************************/
            /*                                                          */
            /*  リード・データ処理  ＩＮ    : com_code, com_prm         */
            /*                      ＯＵＴ  : ret_st, ret_prm           */
            /*                                                          */
            /************************************************************/


unsigned char RDDAT(unsigned char com_code, struct compra *com_prm, struct retpra *ret_prm)
{
      unsigned char com_dat[9], adrss[3], com1_code, ret_st, ret1_st, dvno;
      signed char com_num, retryc, i;
      unsigned long madrss;
      struct compra com1_prm;
```

(LIO.C) p.8

```
dvno = com_code & 0x03;                                    /*  ドライブ番号のデコード      */
DRVSEL(dvno);                                              /*  ドライブ・セレクト信号の選択  */
retryc = (com_code & 0x1c) >> 2;                           /*  ﾘﾄﾗｲ･ｶｳﾝﾀのセット    */
com_dat[0] = 0x06 | (0xe0 & com_code);                     /*  ﾘｰﾄﾞ･ﾃﾞｰﾀ用データのセット    */
if(com_prm->h == 0x00)                                     /*  ドライブとヘッド番号の設定  */
{
        com_dat[1] = com_code & 0x03;                      /*  ヘッド番号＝０のときの設定  */
}
else
{
        com_dat[1] = (com_code | 0x04) & 0x07;             /*  ヘッド番号＝１のときの設定  */
}

--com_prm->trsize.dtsz;                                   /*  転送データ・サイズをデクリメント      */

while(retryc >= 0)
{
        madrss = com_prm->dt_seg;                          /*  転送アドレスの設定  */
        madrss = (madrss << 4) + com_prm->dt_off;
        i = 0;
        while(i <= 2)
        {
                adrss[i] = (unsigned char)(madrss >> (8 * i));
                                                           /*  転送アドレスを8ﾋﾞｯﾄごとに分解       */
                ++i;
        }

        outportb(DMODER, 0x47);                            /*  DMAｺﾝﾄﾛｰﾗのﾓｰﾄﾞ･ﾚｼﾞｽﾀに転送方向をｾｯﾄ  */
        outportb(DCLBFF, 0x00);                            /*  DMAｺﾝﾄﾛｰﾗのｸﾘｱ･ﾊﾞｲﾄ･ﾎﾟｲﾝﾀFFをﾘｾｯﾄ    */
        outportb(DADDR,adrss[0]);                          /*  DMAｺﾝﾄﾛｰﾗのｱﾄﾞﾚｽ･ﾚｼﾞｽﾀにｱﾄﾞﾚｽ(0-7ﾋﾞｯﾄ)をｾｯﾄ*/
        outportb(DADDR,adrss[1]);                          /*  DMAｺﾝﾄﾛｰﾗのｱﾄﾞﾚｽ･ﾚｼﾞｽﾀにｱﾄﾞﾚｽ(8-15ﾋﾞｯﾄ)をｾｯﾄ*/
        outportb(DBNKR,adrss[2]);                          /*  DMAｺﾝﾄﾛｰﾗのﾁｬﾈﾙ3ﾊﾞﾝｸ･ﾚｼﾞｽﾀにｱﾄﾞﾚｽ(16-23ﾋﾞｯﾄ)*/
                                                           /*  をセット  */
        outportb(DCOUNT,com_prm->trsize.int_hen.kai);
                                                           /*  DMAｺﾝﾄﾛｰﾗのｶｳﾝﾄ･ﾚｼﾞｽﾀにデータ・サイズをｾｯﾄ  */
        outportb(DCOUNT,com_prm->trsize.int_hen.jyoui);
                                                           /*  DMAｺﾝﾄﾛｰﾗのｶｳﾝﾄ･ﾚｼﾞｽﾀにデータ・サイズをｾｯﾄ  */
        outportb(DWRSMR,0x03);                             /*  DMAｺﾝﾄﾛｰﾗのﾁｬﾈﾙ3のﾏｽｸを解除する      */
        com_dat[2] = com_prm->c;                           /*  シリンダ番号の設定  */
        com_dat[3] = com_prm->h;                           /*  ヘッド番号の設定    */
        com_dat[4] = com_prm->r;                           /*  開始セクタ番号の設定        */
        com_dat[5] = com_prm->n;                           /*  セクタあたりのバイト数の設定        */
        com_dat[6] = com_prm->eot;                         /*  １トラックのセクタ数の設定  */
        com_dat[7] = com_prm->gsl;                         /*  GAP3の読み飛ばしバイト数の設定  */
        com_dat[8] = com_prm->dtl;                         /*  com_prm.n≠0なので値に意味はない    */
        com_num = 0x09;
        while(time_i != 1);                                /*  モータの回転安定時間待ち    */
        COMMAND(&com_dat[0], com_num);                     /*  ｺﾏﾝﾄﾞ･ﾓｼﾞｭｰﾙ呼びだし        */

        while(int_s != 0x01);                              /*  コマンド終了割り込み待ち    */
        int_s = 0x00;

        if(rslt_pt[0] >= 0x40)                             /*  コマンドの終了結果の判断    */
        {
                if((rslt_pt[0] & 0x08) == 0x08)
                {
                        ret_st = 0x01;                     /*  ドライブのノット・レディ    */
                }
                else
                {
                        switch(rslt_pt[1])
                        {
                                case 0x01:
                                        if((rslt_pt[2] & 0x01) == 0x01)
                                        {
                                                ret_st = 0x08;  /*  ＤＡＭ非検出        */
                                        }
                                        else
                                        {
                                                ret_st = 0x02;  /*  ＩＤＡＭ非検出      */
```

(L10.C) p.9

```
                                }
                                break;
                        case 0x04:
                                switch(rslt_pt[2])
                                {
                                        case 0x02:
                                                ret_st = 0x03;          /*  指定したシリンダがない  */
                                                break;
                                        case 0x10:
                                                ret_st = 0x04;          /*  指定したシリンダがない  */
                                                break;
                                        case 0x00:
                                                ret_st = 0x05;          /*  指定したセクタがない  */
                                                break;
                                        default:
                                                break;
                                }
                                break;
                        case 0x20:
                                if((rslt_pt[2] & 0x20) == 0x20)
                                {
                                        ret_st = 0x07;                  /*  データ部のＣＲＣエラー  */
                                }
                                else
                                {
                                        ret_st = 0x06;                  /*  ＩＤ部のＣＲＣエラー  */
                                }
                                break;
                        case 0x10:
                                ret_st = 0x0a;          /*  オーバラン・エラー  */
                                break;
                        case 0x80:
                                ret_st = 0x0b;          /*  最終セクタを越えてアクセスしようとした  */
                                break;
                        default:
                                break;
                }
        }
}
else
{
        if(rslt_pt[2] < 0x40)
        {
                ret_st = 0x00;                          /*  コマンド正常終了    */
        }
        else
        {
                ret_st = 0x09;                          /*  ＤＤＡＭ付きデータをリードした      */
        }
}

ret_prm->c = rslt_pt[3];                                /*  コマンド終了時のリザルト・パラメータのｾｯﾄ  */
ret_prm->h = rslt_pt[4];
ret_prm->r = rslt_pt[5];
ret_prm->n = rslt_pt[6];

switch(ret_st)                                          /*  リトライ実行の判定  */
{
        case 0x00:                                      /*  ﾘﾀｰﾝ･ｽﾃｰﾀｽが00H,01Hのとき  */
        case 0x01:
                retryc = 0;                             /*  リトライなし        */
                break;
        case 0x02:                                      /*  ﾘﾀｰﾝ･ｽﾃｰﾀｽが02H,05H-08Hのとき      */
        case 0x05:
        case 0x06:
        case 0x07:
        case 0x08:
                if(com_prm->c <= 50)                    /*  リードするシリンダの位置の判断(シークする  */
                                                        /*  方向の決定)     */
                {
```

```
                        com1_code = com_dat[1] | 0x04;  /*  シリンダの位置が50シリンダ以下ならば  */
                    }                                   /*  求心方向へシークする          */
                    else
                    {
                        com1_code = com_dat[1] & 0x03;  /*  シリンダの位置が51シリンダ以上ならば  */
                    }                                   /*  遠心方向にシークする          */

                    com1_prm.c = 20;                    /*  シークするシリンダ数の設定  */

                    ret1_st = RSEEK(com1_code, &com1_prm);
                                                        /*  ﾘﾗﾃｨﾌﾞ･ｼｰｸの実行     */

                    if(ret1_st != 0x00)                 /*  ﾘﾗﾃｨﾌﾞ･ｼｰｸの実行結果がエラーか？    */
                    {
                        retryc = 0;
                        break;                          /*  リトライせずに終了する        */
                    }
                    com1_code = com_code & 0x03;        /*  リードするシリンダへ再シークする    */
                    com1_prm.c = com_prm->c;
                    ret1_st = SEEK(com1_code, &com1_prm);
                                                        /*  シーク実行  */

                    if(ret1_st != 0x00)                 /*  シークの実行結果がエラーか？          */
                    {
                        retryc = 0;                     /*  リトライせずに終了  */
                    }
                    break;
                default:
                    break;
            }
            --retryc;                                   /*  ﾘﾄﾗｲ･ｶｳﾝﾀをﾃﾞｸﾘﾒﾝﾄしてリトライする  */
        }

        outportb(DWRSMR, 0x07);                         /*  DMAｺﾝﾄﾛｰﾗのﾁｬﾈﾙ3をマスクする          */
        return ret_st;
}


            /*************************************************************************/
            /*                                                                       */
            /*  リード・デリーテッド・データ処理     ＩＮ    : com_code, com_prm      */
            /*                                       ＯＵＴ  : ret_st, ret_prm        */
            /*                                                                       */
            /*************************************************************************/


unsigned char RDDDAT(unsigned char com_code, struct compra *com_prm, struct retpra *ret_prm)
{
        unsigned char com_dat[9], com1_code, adrss[3], ret_st, ret1_st, dvno;
        signed char com_num, retryc, i;
        unsigned long madrss;
        struct compra com1_prm;

        dvno = com_code & 0x03;                         /*  ドライブ番号のデコード       */
        DRVSEL(dvno);                                   /*  ドライブセレクト信号の選択  */
        retryc = (com_code & 0x1c) >> 2;                /*  ﾘﾄﾗｲ･ｶｳﾝﾀのセット    */

        com_dat[0] = 0x0c | (0xe0 & com_code);          /*  ﾘｰﾄﾞ･ﾃﾞﾘｰﾃｯﾄﾞ･ﾃﾞｰﾀ用データのセット  */

        if(com_prm->h == 0x00)                          /*  ドライブとヘッド番号の設定  */
        {
            com_dat[1] = com_code & 0x03;               /*  ヘッド番号＝０のときの設定  */
        }
        else
        {
            com_dat[1] = (com_code | 0x04) & 0x07;      /*  ヘッド番号＝１のときの設定  */
        }

        --com_prm->trsize.dtsz;                         /*  転送データ・サイズをデクリメント       */
```

保守／廃止

(L10.C) p.11

```
while(retryc >= 0)
{
        madrss = com_prm->dt_seg;                               /*  転送アドレスの設定  */
        madrss = (madrss << 4) + com_prm->dt_off;
        i = 0;
        while(i <= 2)
        {
                adrss[i] = (unsigned int)(madrss >> (8 * i));
                                                                /*  転送アドレスを8ﾋﾞｯﾄごとに分解       */
                ++i;
        }

        outportb(DMODER, 0x47);                                 /*  DMAｺﾝﾄﾛｰﾗのﾓｰﾄﾞ･ﾚｼﾞｽﾀに転送方向をセット  */
        outportb(DCLBFF, 0x00);                                 /*  DMAｺﾝﾄﾛｰﾗのｸﾘｱ･ﾊﾞｲﾄ･ﾎﾟｲﾝﾀFFをリセット  */
        outportb(DADDR, adrss[0]);                              /*  DMAｺﾝﾄﾛｰﾗのｱﾄﾞﾚｽ･ﾚｼﾞｽﾀにｱﾄﾞﾚｽ(0-7ﾋﾞｯﾄ)をｾｯﾄ */
        outportb(DADDR, adrss[1]);                              /*  DMAｺﾝﾄﾛｰﾗのｱﾄﾞﾚｽ･ﾚｼﾞｽﾀにｱﾄﾞﾚｽ(8-15ﾋﾞｯﾄ)をｾｯﾄ*/
        outportb(DBNKR, adrss[2]);                              /*  DMAｺﾝﾄﾛｰﾗのﾁｬﾈﾙ3ﾊﾞﾝｸ･ﾚｼﾞｽﾀにｱﾄﾞﾚｽ(16-23ﾋﾞｯﾄ)*/
                                                                /*  をセット  */
        outportb(DCOUNT, com_prm->trsize.int_hen.kai);
                                                                /*  DMAｺﾝﾄﾛｰﾗのｶｳﾝﾄ･ﾚｼﾞｽﾀにデータ・サイズをｾｯﾄ*/
        outportb(DCOUNT, com_prm->trsize.int_hen.jyoui);
                                                                /*  DMAｺﾝﾄﾛｰﾗのｶｳﾝﾄ･ﾚｼﾞｽﾀにデータ・サイズをｾｯﾄ*/
        outportb(DWRSMR, 0x03);                                 /*  DMAｺﾝﾄﾛｰﾗのﾁｬﾈﾙ3のﾏｽｸを解除する      */
        com_dat[2] = com_prm->c;                                /*  シリンダ番号の設定  */
        com_dat[3] = com_prm->h;                                /*  ヘッド番号の設定    */
        com_dat[4] = com_prm->r;                                /*  開始セクタ番号の設定        */
        com_dat[5] = com_prm->n;                                /*  セクタあたりのバイト数の設定        */
        com_dat[6] = com_prm->eot;                              /*  １トラックのセクタ数の設定  */
        com_dat[7] = com_prm->gsl;                              /*  GAP3の読み飛ばしバイト数の設定  */
        com_dat[8] = com_prm->dtl;                              /*  com_prm.n≠0なので値に意味はない    */
        com_num = 0x09;
        while(time_i != 1);                                     /*  モータの回転安定時間待ち    */
        COMMAND(&com_dat[0], com_num);                          /*  ｺﾏﾝﾄﾞ･ﾓｼﾞｭｰﾙ呼びだし        */

        while(int_s != 0x01);                                   /*  コマンド終了割り込み待ち    */
        int_s = 0x00;
        if(rslt_pt[0] >= 0x40)                                  /*  コマンドの終了結果の判断    */
        {
                if((rslt_pt[0] & 0x08) == 0x08)
                {
                        ret_st = 0x01;                          /*  ドライブのノット・レディ    */
                }
                else
                {
                        switch(rslt_pt[1])
                        {
                                case 0x01:
                                        if((rslt_pt[2] & 0x01) == 0x01)
                                        {
                                                ret_st = 0x08;          /*  ＤＡＭ非検出        */
                                        }
                                        else
                                        {
                                                ret_st = 0x02;          /*  ＩＤＡＭ非検出      */
                                        }
                                        break;
                                case 0x04:
                                        switch(rslt_pt[2])
                                        {
                                                case 0x02:
                                                        ret_st = 0x03;          /*  指定したシリンダがない  */
                                                        break;
                                                case 0x10:
                                                        ret_st = 0x04;          /*  指定したシリンダがない  */
                                                        break;
                                                case 0x00:
                                                        ret_st = 0x05;          /*  指定したセクタがない  */
                                                        break;
                                                default:
                                                        break;
```

保守／廃止

(L10.C) p.12

```
                                }
                                break;
                        case 0x20:
                                if((rslt_pt[2] & 0x20) == 0x20)
                                {
                                        ret_st = 0x07;          /*  データ部のＣＲＣエラー      */
                                }
                                else
                                {
                                        ret_st = 0x06;          /*  ＩＤ部のＣＲＣエラー        */
                                }
                                break;
                        case 0x10:
                                ret_st = 0x0a;                  /*  オーバラン・エラー  */
                                break;
                        case 0x80:
                                ret_st = 0x0b;          /*  最終セクタを越えてアクセスしようとした      */
                                break;
                        default:
                                break;
                }
        }
}
else
{
        if(rslt_pt[2] < 0x40)
        {
                ret_st = 0x00;                          /*  コマンド正常終了    */
        }
        else
        {
                ret_st = 0x09;                          /*  ＤＤＡＭ付きデータをリードした      */
        }
}
ret_prm->c = rslt_pt[3];                                /*  コマンド終了時のリザルト・パラメータのｾｯﾄ   */
ret_prm->h = rslt_pt[4];
ret_prm->r = rslt_pt[5];
ret_prm->n = rslt_pt[6];

switch(ret_st)                                          /*  リトライ実行の判定  */
{
        case 0x00:                                      /*  ﾘﾀｰﾝ･ｽﾃｰﾀｽが00H,01Hのとき    */
        case 0x01:
                retryc = 0;                             /*  リトライなし        */
                break;
        case 0x02:                                      /*  ﾘﾀｰﾝ･ｽﾃｰﾀｽが02H,05H-08Hのとき      */
        case 0x05:
        case 0x06:
        case 0x07:
        case 0x08:
                if(com_prm->c <= 50)                    /*  リードするシリンダの位置の判断(シークする   */
                                                        /*  方向の決定)     */
                {
                        coml_code = com_dat[1] | 0x04;  /*  シリンダの位置が50シリンダ以下ならば  */
                }                                       /*  求心方向へシークする        */
                else
                {
                        coml_code = com_dat[1] | 0x03;  /*  シリンダの位置が51シリンダ以上ならば  */
                }                                       /*  遠心方向にシークする        */

                coml_prm.c = 20;                        /*  シークするシリンダ数の設定  */

                retl_st = RSEEK(coml_code, &coml_prm);
                                                        /*  ﾘﾗﾃｨﾌﾞ･ｼｰｸの実行    */

                if(retl_st != 0x00)                     /*  ﾘﾗﾃｨﾌﾞ･ｼｰｸの実行結果がエラーか？    */
                {
                        retryc = 0;
                        break;                          /*  リトライせずに終了する      */
                }
```

保守／廃止

(L10.C) p.13

```
                        coml_code = com_code & 0x03;            /*  リードするシリンダへ再シークする    */
                        coml_prm.c = com_prm->c;
                        retl_st = SEEK(coml_code, &coml_prm);
                                                                /*  シーク実行  */

                        if(retl_st != 0x00)                     /*  シークの実行結果がエラーか？        */
                        {
                                retryc = 0;                     /*  リトライせずに終了  */
                        }
                        break;
                default:
                        break;
        }
        --retryc;                                               /*  ﾘﾄﾗｲ･ｶｳﾝﾀをﾃﾞｸﾘﾒﾝﾄしてリトライする  */
    }
    outportb(DWRSMR, 0x07);                                     /*  DMAｺﾝﾄﾛｰﾗのﾁｬﾈﾙ3をマスクする        */
    return ret_st;
}



        /*****************************************************/
        /*                                                   */
        /*  リードＩＤ処理      ＩＮ    : com_code           */
        /*                      ＯＵＴ  : ret_st, ret_prm    */
        /*                                                   */
        /*****************************************************/


unsigned char RDID(unsigned char com_code, struct retpra *ret_prm)
{
    unsigned char com_dat[2], ret_st, dvno;
    signed char com_num, retryc;

    dvno = com_code & 0x03;                                     /*  ドライブ番号のデコード      */
    DRVSEL(dvno);                                               /*  ドライブ・セレクト信号の選択  */
    retryc = (com_code &0x1c) >> 2;                             /*  ﾘﾄﾗｲ･ｶｳﾝﾀのセット   */
    com_dat[0] = 0x0a | (0x40 & com_code);                      /*  ﾘｰﾄﾞID用データのセット     */
    while(retryc >= 0)
    {
        if((com_code & 0x20) == 0x20)                           /*  ドライブとヘッド番号の設定  */
        {
            com_dat[1] = 0x04 | (0x03 & com_code);              /*  ヘッド番号＝１のときの設定  */
        }
        else
        {
            com_dat[1] = 0x03 & com_code;                       /*  ヘッド番号＝０のときの設定  */
        }
        com_num = 0x02;
        while(time_i != 1);                                     /*  モータの回転安定時間待ち    */
        COMMAND(&com_dat[0], com_num);                          /*  ｺﾏﾝﾄﾞ･ﾓｼﾞｭｰﾙ呼びだし        */

        while(int_s != 0x01);                                   /*  コマンド終了割り込み待ち    */
        int_s = 0x00;
        if(rslt_pt[0] < 0x40)                                   /*  コマンドの終了結果の判断    */
        {
            ret_st = 0x00;                                      /*  コマンド正常終了    */
        }
        else
        {
            if((rslt_pt[0] & 0x08) == 0x08)
            {
                ret_st = 0x01;                                  /*  ドライブのノット・レディ    */
            }
            if((rslt_pt[1] & 0x01) == 0x01)
            {
                ret_st = 0x02;                                  /*  ＩＤＡＭ非検出      */
            }
            if((rslt_pt[1] & 0x04) == 0x04)
            {
```

保守／廃止

(LIO.C) p.14

```
                        ret_st = 0x05;                          /*  指定したセクタがない        */
                }
        }
        ret_prm->c = rslt_pt[3];                                /*  コマンド終了時のリザルト・パラメータのｾｯﾄ   */
        ret_prm->h = rslt_pt[4];
        ret_prm->r = rslt_pt[5];
        ret_prm->n = rslt_pt[6];
        if((ret_st == 0x00) || (ret_st == 0x01))                /*  リトライ実行の判断  */
        {                                                       /*  ﾘﾀｰﾝ･ｽﾃｰﾀｽが00H,01Hのとき    */
                break;                                          /*  リトライなし        */
        }
        --retryc;                                               /*  ﾘﾀｰﾝ･ｽﾃｰﾀｽがそれ以外のときはﾘﾄﾗｲ･ｶｳﾝﾀを  */
                                                                /*  ﾃﾞｸﾘﾒﾝﾄしてリトライする      */
    }
    return ret_st;
}


        /*************************************************************************/
        /*                                                                       */
        /*  リード・ダイアグノスティック処理     ＩＮ    : com_code, com_prm      */
        /*                                       ＯＵＴ  : ret_st, ret_prm        */
        /*                                                                       */
        /*************************************************************************/

unsigned char RDDAG(unsigned char com_code, struct compra *com_prm, struct retpra *ret_prm)
{
    unsigned char com_dat[9], adrss[3], coml_code, ret_st, retl_st, dvno;
    signed char com_num, retryc, i;
    unsigned long madrss;
    struct compra coml_prm;

    dvno = com_code & 0x03;                                     /*  ドライブ番号のデコード      */
    DRVSEL(dvno);                                               /*  ドライブ・セレクト信号の選択  */
    retryc = (com_code & 0x1c) >> 2;                            /*  ﾘﾄﾗｲ･ｶｳﾝﾀのセット    */
    com_dat[0] = 0x02 | (0x40 & com_code);                      /*  ﾘｰﾄﾞ･ｱ･ﾄﾗｯｸ用データのセット */
    if(com_prm->h == 0x00)                                      /*  ドライブとヘッド番号の設定  */
    {
        com_dat[1] = com_code & 0x03;                           /*  ヘッド番号＝０のときの設定  */
    }
    else
    {
        com_dat[1] = (com_code | 0x04) & 0x07;                  /*  ヘッド番号＝１のときの設定  */
    }
    --com_prm->trsize.dtsz;                                     /*  転送データ・サイズをデクリメント      */
    while(retryc >= 0)
    {
        madrss = com_prm->dt_seg;                               /*  転送アドレスの設定  */
        madrss = (madrss << 4) + com_prm->dt_off;
        i = 0;
        while(i <= 2)
        {
            adrss[i] = (unsigned char)(madrss >> (8 * i));
                                                                /*  転送アドレスを8ﾋﾞｯﾄごとに分解       */
            ++i;
        }

        outportb(DMODER, 0x47);                                 /*  DMAｺﾝﾄﾛｰﾗのﾓｰﾄﾞ･ﾚｼﾞｽﾀに転送方向をｾｯﾄ  */
        outportb(DCLBFF, 0x00);                                 /*  DMAｺﾝﾄﾛｰﾗのｸﾘｱ･ﾊﾞｲﾄ･ﾎﾟｲﾝﾀFFをリセット*/
        outportb(DADDR, adrss[0]);                              /*  DMAｺﾝﾄﾛｰﾗのｱﾄﾞﾚｽ･ﾚｼﾞｽﾀにｱﾄﾞﾚｽ(0-7ﾋﾞｯﾄ)をｾｯﾄ */
        outportb(DADDR, adrss[1]);                              /*  DMAｺﾝﾄﾛｰﾗのｱﾄﾞﾚｽ･ﾚｼﾞｽﾀにｱﾄﾞﾚｽ(8-15ﾋﾞｯﾄ)をｾｯﾄ*/
        outportb(DBNKR, adrss[2]);                              /*  DMAｺﾝﾄﾛｰﾗのﾁｬﾈﾙ3ﾊﾞﾝｸ･ﾚｼﾞｽﾀにｱﾄﾞﾚｽ(16-23ﾋﾞｯﾄ)*/
                                                                /*  をセット  */
        outportb(DCOUNT, com_prm->trsize.int_hen.kai);
                                                                /*  DMAｺﾝﾄﾛｰﾗのｶｳﾝﾄ･ﾚｼﾞｽﾀにデータ・サイズをｾｯﾄ  */
        outportb(DCOUNT, com_prm->trsize.int_hen.jyoui);
                                                                /*  DMAｺﾝﾄﾛｰﾗのｶｳﾝﾄ･ﾚｼﾞｽﾀにデータ・サイズをｾｯﾄ  */
        outportb(DWRSMR, 0x03);                                 /*  DMAｺﾝﾄﾛｰﾗのﾁｬﾈﾙ3のﾏｽｸを解除する      */
        com_dat[2] = com_prm->c;                                /*  シリンダ番号の設定  */
        com_dat[3] = com_prm->h;                                /*  ヘッド番号の設定    */
```

保守／廃止

(L10.C) p.15

```
com_dat[4] = com_prm->r;                                /*  開始セクタ番号の設定          */
com_dat[5] = com_prm->n;                                /*  セクタあたりのバイト数の設定          */
com_dat[6] = com_prm->eot;                              /*  1トラックのセクタ数の設定  */
com_dat[7] = com_prm->gsl;                              /*  GAP3の読み飛ばしバイト数の設定  */
com_dat[8] = com_prm->dtl;                              /*  com_prm.n≠0なので値に意味はない    */
com_num = 0x09;
while(time_i != 1);                                     /*  モータの回転安定時間待ち    */
COMMAND(&com_dat[0], com_num);                          /*  ｺﾏﾝﾄﾞ･ﾓｼﾞｭｰﾙ呼びだし          */

while(int_s != 0x01);                                   /*  コマンド終了割り込み待ち    */
int_s = 0x00;
if(rslt_pt[0] >= 0x40)                                  /*  コマンドの終了結果の判断    */
{
        switch(rslt_pt[1])
        {
                case 0x01:
                        if((rslt_pt[2] & 0x01) == 0x01)
                        {
                                ret_st = 0x08;          /*  ＤＡＭ非検出          */
                        }
                        else
                        {
                                ret_st = 0x02;          /*  ＩＤＡＭ非検出        */
                        }
                        break;
                case 0x00:
                        ret_st = 0x01;                  /*  ドライブのノット・レディ    */
                        break;
                case 0x10:
                        ret_st = 0x0a;                  /*  オーバラン・エラー  */
                        break;
                case 0x80:
                        ret_st = 0x0b;                  /*  最終セクタを越えてアクセスしようとした  */
                        break;
                default:
                        break;
        }
}
else
{
        switch(rslt_pt[1])
        {
                case 0x00:
                        if((rslt_pt[2] & 0x40) == 0x40)
                        {
                                ret_st = 0x09;          /*  ＤＤＡＭ付きデータをリードした        */
                        }
                        else
                        {
                                ret_st = 0x00;          /*  コマンド正常終了    */
                        }
                        break;
                case 0x20:
                       if((rslt_pt[2] & 0x20) == 0x20)
                        {
                                ret_st = 0x07;          /*  データ部のＣＲＣエラー        */
                        }
                        else
                        {
                                ret_st = 0x06;          /*  ＩＤ部のＣＲＣエラー          */
                        }
                        break;
                case 0x04:
                        ret_st = 0x05;                  /*  指定したシリンダがない        */
                        break;
                default:
                        break;
        }
}
ret_prm->c = rslt_pt[3];                                /*  コマンド終了時のﾘｻﾞﾙﾄ･ﾊﾟﾗﾒｰﾀのｾｯﾄ   */
```

(L10.C) p.16

```
            ret_prm->h = rslt_pt[4];
            ret_prm->r = rslt_pt[5];
            ret_prm->n = rslt_pt[6];

            switch(ret_st)                                      /*  ﾘﾄﾗｲ実行の判定  */
            {
                  case 0x00:                                    /*  ﾘﾀｰﾝ･ｽﾃｰﾀｽが00H,01Hのとき   */
                  case 0x01:
                        retryc = 0;                             /*  ﾘﾄﾗｲなし        */
                        break;
                  case 0x02:                                    /*  ﾘﾀｰﾝ･ｽﾃｰﾀｽが02H,05H-08Hのとき       */
                  case 0x05:
                  case 0x06:
                  case 0x07:
                  case 0x08:
                        if(com_prm->c <= 50)                    /*  ﾘｰﾄﾞするｼﾘﾝﾀﾞの位置の判断(ｼｰｸする   */
                                                                /*  方向の決定)    */
                        {
                              coml_code = com_dat[1] | 0x04;    /*  ｼﾘﾝﾀﾞの位置が50ｼﾘﾝﾀﾞ以下ならば  */
                        }                                       /*  求心方向へｼｰｸする        */
                        else
                        {
                              coml_code = com_dat[1] & 0x03;    /*  ｼﾘﾝﾀﾞの位置が51ｼﾘﾝﾀﾞ以上ならば  */
                        }                                       /*  遠心方向にｼｰｸする        */

                        coml_prm.c = 20;                        /*  ｼｰｸするｼﾘﾝﾀﾞ数の設定  */

                        retl_st = RSEEK(coml_code, &coml_prm);
                                                                /*  ﾘﾗﾃｨﾌﾞ･ｼｰｸの実行    */

                        if(retl_st != 0x00)                     /*  ﾘﾗﾃｨﾌﾞ･ｼｰｸの実行結果がｴﾗｰか？    */
                        {
                              retryc = 0;
                              break;                            /*  ﾘﾄﾗｲせずに終了する       */
                        }
                        coml_code = com_code & 0x03;            /*  ﾘｰﾄﾞするｼﾘﾝﾀﾞへ再ｼｰｸする    */
                        coml_prm.c = com_prm->c;
                        retl_st = SEEK(coml_code, &coml_prm);
                                                                /*  ｼｰｸ実行  */

                        if(retl_st != 0x00)                     /*  ｼｰｸの実行結果がｴﾗｰか？        */
                        {
                              retryc = 0;                       /*  ﾘﾄﾗｲせずに終了  */
                        }
                        break;
                  default:
                        break;
            }
            --retryc;                                           /*  ﾘﾄﾗｲ･ｶｳﾝﾀをﾃﾞｸﾘﾒﾝﾄしてﾘﾄﾗｲする  */
      }
      outportb(DWRSMR, 0x07);                                   /*  DMAｺﾝﾄﾛｰﾗのﾁｬﾈﾙ3をﾏｽｸする        */
      return ret_st;
}


            /*********************************************************/
            /*                                                       */
            /*  ﾗｲﾄ・ﾃﾞｰﾀ処理   ＩＮ    : com_code, com_prm            */
            /*                  ＯＵＴ  : ret_st, ret_prm             */
            /*                                                       */
            /*********************************************************/


unsigned char WRDAT(unsigned char com_code, struct compra *com_prm, struct retpra *ret_prm)
{
      unsigned char com_dat[9], adrss[3], coml_code, ret_st, retl_st, dvno;
      signed char com_num, retryc, i;
      unsigned long madrss;
      struct compra coml_prm;
```

保守／廃止

(L10.C) p.17

```
dvno = com_code & 0x03;                                 /*  ドライブ番号のデコード      */
DRVSEL(dvno);                                           /*  ドライブ・セレクト信号の選択        */
retryc = (com_code & 0x1c) >> 2;                        /*  ﾘﾄﾗｲ･ｶｳﾝﾀのセット   */
com_dat[0] = 0x05 | (0xc0 & com_code);                  /*  ﾗｲﾄ･ﾃﾞｰﾀ用データのセット    */
if(com_prm->h == 0x00)                                  /*  ドライブとヘッド番号の設定  */
{
        com_dat[1] = com_code & 0x03;                   /*  ヘッド番号＝０のときの設定  */
}
else
{
        com_dat[1] = (com_code | 0x04) & 0x07;          /*  ヘッド番号＝１のときの設定  */
}
--com_prm->trsize.dtsz;                                 /*  転送データ・サイズをデクリメント      */
while(retryc >= 0)
{
        madrss = com_prm->dt_seg;                       /*  転送アドレスの設定  */
        madrss = (madrss << 4) + com_prm->dt_off;
        i = 0;
        while(i <= 2)
        {
                adrss[i] = (unsigned char)(madrss >>(8 * i));
                                                        /*  転送アドレスを8ﾋﾞｯﾄごとに分解       */
                ++i;
        }

        outportb(DMODER, 0x4b);                         /*  DMAｺﾝﾄﾛｰﾗのﾓｰﾄﾞ･ﾚｼﾞｽﾀに転送方向をセット*/
        outportb(DCLBFF, 0x00);                         /*  DMAｺﾝﾄﾛｰﾗのｸﾘｱ･ﾊﾞｲﾄ･ﾎﾟｲﾝﾀFFをリセット*/
        outportb(DADDR,adrss[0]);                       /*  DMAｺﾝﾄﾛｰﾗのｱﾄﾞﾚｽ･ﾚｼﾞｽﾀにｱﾄﾞﾚｽ(0-7ﾋﾞｯﾄ)をｾｯﾄ */
        outportb(DADDR,adrss[1]);                       /*  DMAｺﾝﾄﾛｰﾗのｱﾄﾞﾚｽ･ﾚｼﾞｽﾀにｱﾄﾞﾚｽ(8-15ﾋﾞｯﾄ)をｾｯﾄ*/
        outportb(DBNKR,adrss[2]);                       /*  DMAｺﾝﾄﾛｰﾗのﾁｬﾈﾙ3ﾊﾞﾝｸ･ﾚｼﾞｽﾀにｱﾄﾞﾚｽ(16-23ﾋﾞｯﾄ)*/
                                                        /*  をセット  */
        outportb(DCOUNT,com_prm->trsize.int_hen.kai);
                                                        /*  DMAｺﾝﾄﾛｰﾗのｶｳﾝﾄ･ﾚｼﾞｽﾀにデータ・サイズをｾｯﾄ  */

        outportb(DCOUNT,com_prm->trsize.int_hen.jyoui);
                                                        /*  DMAｺﾝﾄﾛｰﾗのｶｳﾝﾄ･ﾚｼﾞｽﾀにデータ・サイズをｾｯﾄ  */
        outportb(DWRSMR,0x03);                          /*  DMAｺﾝﾄﾛｰﾗのﾁｬﾈﾙ3のマスクを解除する  */
        com_dat[2] = com_prm->c;                        /*  シリンダ番号の設定  */
        com_dat[3] = com_prm->h;                        /*  ヘッド番号の設定    */
        com_dat[4] = com_prm->r;                        /*  開始セクタ番号の設定        */
        com_dat[5] = com_prm->n;                        /*  セクタあたりのバイト数の設定        */
        com_dat[6] = com_prm->eot;                      /*  １トラックあたりのセクタ数の設定    */
        com_dat[7] = com_prm->gsl;                      /*  GAP3の読み飛ばしバイト数の設定      */
        com_dat[8] = com_prm->dtl;                      /*  com_prm.n≠0なので値に意味はない    */
        com_num = 0x09;
        while(time_i != 1);                             /*  モータの回転安定時間待ち    */
        COMMAND(&com_dat[0], com_num);                  /*  ｺﾏﾝﾄﾞ･ﾓｼﾞｭｰﾙ呼びだし        */
        while(int_s != 0x01);                           /*  コマンド終了割り込み待ち    */
        int_s = 0x00;
        if(rslt_pt[0] >= 0x40)                          /*  コマンドの終了結果の判断    */
        {
                if((rslt_pt[0] & 0x08) == 0x08)
                {
                        ret_st = 0x01;                  /*  ドライブのノット・レディ    */
                }
                else
                {
                        switch(rslt_pt[1])
                        {
                                case 0x01:
                                        ret_st = 0x02;          /*  ＩＤＡＭ非検出      */
                                        break;
                                case 0x02:
                                        ret_st = 0x0c;          /*  ライト・プロテクト  */
                                        break;
                                case 0x04:
                                        switch(rslt_pt[2])
                                        {
                                                case 0x02:
```

保守／廃止

(LIO.C) p.18

```
                                        ret_st = 0x03;          /*  指定したシリンダがない  */
                                        break;
                                case 0x10:
                                        ret_st = 0x04;          /*  指定したシリンダがない  */
                                        break;
                                case 0x00:
                                        ret_st = 0x05;          /*  指定したセクタがない  */
                                        break;
                                deault:
                                        break;
                                }
                                break;
                        case 0x20:
                                ret_st = 0x06;          /*  ＩＤ部のＣＲＣエラー        */
                                break;
                        case 0x10:
                                ret_st = 0x0a;          /*  オーバラン・エラー  */
                                break;
                        case 0x80:
                                ret_st = 0x0b;          /*  最終セクタを越えてアクセスしようとした  */
                                break;
                        default:
                                break;
                }
        }
}
else
{
        ret_st = 0x00;                                  /*  コマンドの正常終了  */
}
ret_prm->c = rslt_pt[3];                                /*  コマンド終了時のリザルト・パラメータのｾｯﾄ  */
ret_prm->h = rslt_pt[4];
ret_prm->r = rslt_pt[5];
ret_prm->n = rslt_pt[6];

switch(ret_st)                                          /*  リトライ実行の判定  */
{
        case 0x00:                                      /*  ﾘﾀｰﾝ･ｽﾃｰﾀｽが00H,01H,03H,04H,0CHのとき  */
        case 0x01:
        case 0x03:
        case 0x04:
        case 0x0c:
                retryc = 0;                             /*  リトライなし        */
                break;
        case 0x02:                                      /*  ﾘﾀｰﾝ･ｽﾃｰﾀｽが02H,05H,06Hのとき        */
        case 0x05:
        case 0x06:
                if(com_prm->c <= 50)                    /*  リードするシリンダの位置の判断(シークする*/
                                                        /*  方向の判定)    */
                {
                        com1_code = com_dat[1] | 0x04;  /*  シリンダの位置が50シリンダ以下ならば  */
                }                                       /*  求心方向へシークする        */
                else
                {
                        com1_code = com_dat[1] & 0x03;  /*  シリンダの位置が51シリンダ以上ならば  */
                }                                       /*  遠心方向にシークする        */

                com1_prm.c = 20;                        /*  シークするシリンダ数の設定  */

                ret1_st = RSEEK(com1_code, &com1_prm);
                                                        /*  ﾘﾗﾃｨﾌﾞ･ｼｰｸの実行    */

                if(ret1_st != 0x00)                     /*  ﾘﾗﾃｨﾌﾞ･ｼｰｸの実行結果がエラーか？    */
                {
                        retryc = 0;
                        break;                          /*  リトライせずに終了  */
                }
                com1_code = com_code & 0x03;            /*  リードするシリンダへ再シークする    */
                com1_prm.c = com_prm->c;
                ret1_st = SEEK(com1_code, &com1_prm);
```

保守／廃止

(LIO.C) p.19

```
                                                        /*  シーク実行  */

                        if(ret1_st != 0x00)             /*  シークの実行結果がエラーか？        */
                        {
                                retryc = 0;             /*  リトライせずに終了  */
                        }
                        break;
                default:
                        break;
        }
        --retryc;                                       /*  ﾘﾄﾗｲ･ｶｳﾝﾀをﾃﾞｸﾘﾒﾝﾄしてリトライする  */
    }
    outportb(DWRSMR, 0x07);                             /*  DMAｺﾝﾄﾛｰﾗのﾁｬﾈﾙ3をマスクする        */
    return ret_st;
}



        /*************************************************************************/
        /*                                                                       */
        /*  ライト・デリーテッド・データ処理     ＩＮ    : com_code, com_prm      */
        /*                                       ＯＵＴ  : ret_st, ret_prm        */
        /*                                                                       */
        /*************************************************************************/


unsigned char WRDDAT(unsigned char com_code, struct compra *com_prm, struct retpra *ret_prm)
{
    unsigned char com_dat[9], adrss[3], com1_code, ret_st, ret1_st, dvno;
    signed char com_num, retryc, i;
    unsigned long madrss;
    struct compra com1_prm;

    dvno = com_code & 0x03;                             /*  ドライブ番号のデコード      */
    DRVSEL(dvno);                                       /*  ドライブ・セレクト信号の選択        */
    retryc = (com_code & 0x1c) >> 2;                    /*  ﾘﾄﾗｲ･ｶｳﾝﾀのセット   */
    com_dat[0] = 0x09 | (0xc0 & com_code);              /*  ﾗｲﾄ･ﾃﾞｰﾀ用データのセット    */
    if(com_prm->h == 0x00)                              /*  ドライブとヘッド番号の設定  */
    {
        com_dat[1] = com_code & 0x03;                   /*  ヘッド番号＝０のときの設定  */
    }
    else
    {
        com_dat[1] = (com_code | 0x04) & 0x07;          /*  ヘッド番号＝１のときの設定  */
    }
    --com_prm->trsize.dtsz;                             /*  転送データ・サイズをデクリメント      */
    while(retryc >= 0)
    {
        madrss = com_prm->dt_seg;                       /*  転送アドレスの設定  */
        madrss = (madrss << 4) + com_prm->dt_off;
        i = 0;
        while(i <= 2)
        {
            adrss[i] = (unsigned char)(madrss >>(8 * i));
                                                        /*  転送アドレスを8ﾋﾞｯﾄごとに分解       */
            ++i;
        }

        outportb(DMODER, 0x4b);                         /*  DMAｺﾝﾄﾛｰﾗのﾓｰﾄﾞ･ﾚｼﾞｽﾀに転送方向をセット  */
        outportb(DCLBFF, 0x00);                         /*  DMAｺﾝﾄﾛｰﾗのｸﾘｱ･ﾊﾞｲﾄ･ﾎﾟｲﾝﾀFFをリセット  */
        outportb(DADDR, adrss[0]);                      /*  DMAｺﾝﾄﾛｰﾗのｱﾄﾞﾚｽ･ﾚｼﾞｽﾀにｱﾄﾞﾚｽ(0-7ﾋﾞｯﾄ)をｾｯﾄ*/
        outportb(DADDR, adrss[1]);                      /*  DMAｺﾝﾄﾛｰﾗのｱﾄﾞﾚｽ･ﾚｼﾞｽﾀにｱﾄﾞﾚｽ(8-15ﾋﾞｯﾄ)をｾｯﾄ*/
        outportb(DBNKR, adrss[2]);                      /*  DMAｺﾝﾄﾛｰﾗのﾁｬﾈﾙ3ﾊﾞﾝｸ･ﾚｼﾞｽﾀにｱﾄﾞﾚｽ(16-23ﾋﾞｯﾄ)*/
                                                        /*  をセット  */
        outportb(DCOUNT, com_prm->trsize.int_hen.kai);
                                                        /*  DMAｺﾝﾄﾛｰﾗのｶｳﾝﾄ･ﾚｼﾞｽﾀにﾃﾞｰﾀ･ｻｲｽﾞをｾｯﾄ  */


        outportb(DCOUNT, com_prm->trsize.int_hen.jyoui);
                                                        /*  DMAｺﾝﾄﾛｰﾗのｶｳﾝﾄ･ﾚｼﾞｽﾀにデータ・サイズをｾｯﾄ*/
        outportb(DWRSMR, 0x03);                         /*  DMAｺﾝﾄﾛｰﾗのﾁｬﾈﾙ3のマスクを解除する  */
```

保守／廃止

(L10.C) p.20

```
com_dat[2] = com_prm->c;                        /*  シリンダ番号の設定  */
com_dat[3] = com_prm->h;                        /*  ヘッド番号の設定    */
com_dat[4] = com_prm->r;                        /*  開始セクタ番号の設定        */
com_dat[5] = com_prm->n;                        /*  セクタあたりのバイト数の設定        */
com_dat[6] = com_prm->eot;                      /*  １トラックあたりのセクタ数の設定    */
com_dat[7] = com_prm->gsl;                      /*  GAP3の読み飛ばしバイト数の設定      */
com_dat[8] = com_prm->dtl;                      /*  com_prm.n≠0なので値に意味はない    */
com_num = 0x09;
while(time_i != 1);                             /*  モータの回転安定時間待ち    */
COMMAND(&com_dat[0], com_num);                  /*  ｺﾏﾝﾄﾞ･ﾓｼﾞｭｰﾙ呼びだし        */
while(int_s != 0x01);                           /*  コマンド終了割り込み待ち    */
int_s = 0x00;
if(rslt_pt[0] >= 0x40)                          /*  コマンドの終了結果の判断    */
{
        if((rslt_pt[0] & 0x08) == 0x08)
        {
                ret_st = 0x01;                  /*  ドライブのノット・レディ    */
        }
        else
        {
                switch(rslt_pt[1])
                {
                        case 0x01:
                                ret_st = 0x02;          /*  ＩＤＡＭ非検出      */
                                break;
                        case 0x02:
                                ret_st = 0x0c;          /*  ライト・プロテクト  */
                                break;
                        case 0x04:
                                switch(rslt_pt[2])
                                {
                                        case 0x02:
                                                ret_st = 0x03;          /*  指定したシリンダがない  */
                                                break;
                                        case 0x10:
                                                ret_st = 0x04;          /*  指定したシリンダがない  */
                                                break;
                                        case 0x00:
                                                ret_st = 0x05;          /*  指定したセクタがない  */
                                                break;
                                        deault:
                                                break;
                                }
                                break;
                        case 0x20:
                                ret_st = 0x06;          /*  ＩＤ部のＣＲＣエラー        */
                                break;
                        case 0x10:
                                ret_st = 0x0a;          /*  オーバラン・エラー  */
                                break;
                        case 0x80:
                                ret_st = 0x0b;          /*  最終セクタを越えてアクセスしようとした  */
                                break;
                        default:
                                break;
                }
        }
}
else
{
        ret_st = 0x00;                          /*  コマンドの正常終了  */
}
ret_prm->c = rslt_pt[3];                        /*  コマンド終了時のリザルト・パラメータのｾｯﾄ   */
ret_prm->h = rslt_pt[4];
ret_prm->r = rslt_pt[5];
ret_prm->n = rslt_pt[6];

switch(ret_st)                                  /*  リトライ実行の判定  */
{
        case 0x00:                              /*  ﾘﾀｰﾝ･ｽﾃｰﾀｽが00H,01H,03H,04H,0CHのとき       */
```

保守／廃止

(LIO.C) p.21

```
                case 0x01:
                case 0x03:
                case 0x04:
                case 0x0c:
                        retryc = 0;                             /*  リトライなし        */
                        break;
                case 0x02:                                      /*  ﾘﾀｰﾝ･ｽﾃｰﾀｽが02H,05H,06Hのとき       */
                case 0x05:
                case 0x06:
                        if(com_prm->c <= 50)                    /*  リードするシリンダの位置の判断(シークする   */
                                                                /*  方向の判定)     */
                        {
                                com1_code = com_dat[1] | 0x04;  /*  シリンダの位置が50シリンダ以下ならば  */
                        }                                       /*  求心方向へシークする        */
                        else
                        {
                                com1_code = com_dat[1] & 0x03;  /*  シリンダの位置が51シリンダ以上ならば  */
                        }                                       /*  遠心方向にシークする        */

                        com1_prm.c = 20;                        /*  シークするシリンダ数の設定  */

                        ret1_st = RSEEK(com1_code, &com1_prm);
                                                                /*  ﾘﾗﾃｨﾌﾞ･ｼｰｸの実行     */

                        if(ret1_st != 0x00)                     /*  ﾘﾗﾃｨﾌﾞ･ｼｰｸの実行結果がエラーか？     */
                        {
                                retryc = 0;
                                break;                          /*  リトライせずに終了  */
                        }
                        com1_code = com_code & 0x03;            /*  リードするシリンダへ再シークする     */
                        com1_prm.c = com_prm->c;
                        ret1_st = SEEK(com1_code, &com1_prm);
                                                                /*  シーク実行  */

                        if(ret1_st != 0x00)                     /*  シークの実行結果がエラーか？        */
                        {
                                retryc = 0;                     /*  リトライせずに終了  */
                        }
                        break;
                default:
                        break;
        }
        --retryc;                                               /*  ﾘﾄﾗｲ･ｶｳﾝﾀをﾃﾞｸﾘﾒﾝﾄしてリトライする  */
    }
    outportb(DWRSMR, 0x07);                                     /*  DMAｺﾝﾄﾛｰﾗのﾁｬﾈﾙ3をマスクする        */
    return ret_st;
}



        /*****************************************************************/
        /*                                                               */
        /*  トラック・フォーマット処理   ＩＮ    : com_code, com_prm      */
        /*                               ＯＵＴ  : ret_st, ret_prm        */
        /*                                                               */
        /*****************************************************************/


unsigned char TRFORMAT(unsigned char com_code, struct compra *com_prm, struct retpra *ret_prm)
{
    unsigned char com_dat[9], adrss[3], ret_st, dvno;
    signed char com_num, retryc, i;
    unsigned int dt_seg, dt_off;
    unsigned long madrss;
    struct form fmtdat[36];

    com_dat[0] = 0x0d | (0x40 & com_code);                      /*  ﾄﾗｯｸ･ﾌｫｰﾏｯﾄ用データのセット */

    retryc = (com_code & 0x1c) >> 2;                            /*  ﾘﾄﾗｲ･ｶｳﾝﾀのセット    */
```

保守／廃止

(LIO.C) p.22

```
dvno = com_code & 0x03;                                 /*  ドライブ番号のデコード      */
DRVSEL(dvno);                                           /*  ドライブ・セレクト信号の選択        */

if((com_code & 0x20) == 0x00)                           /*  ドライブとヘッド番号の設定  */
{
        com_dat[1] = com_code & 0x03;                   /*  ヘッド番号＝０のときの設定  */
}
else
{
        com_dat[1] = (com_code | 0x04) & 0x07;          /*  ヘッド番号＝１のときの設定  */
}
while(retryc >= 0)
{
        for(i = 1; i <= com_prm->dtl; i++)              /*  フォーマットに使用するID部のデータのセット  */
        {
                fmtdat[i - 1].c = com_prm->c;
                if((com_code & 0x20) == 0x00)           /*  ヘッド番号の設定    */
                {
                        fmtdat[i - 1].h = 0;            /*  ヘッド番号＝0のときの設定   */
                }
                else
                {
                        fmtdat[i - 1].h = 1;            /*  ヘッド番号＝１のときの設定  */
                }
                fmtdat[i - 1].r = i;
                fmtdat[i - 1].n = com_prm->n;
        }
        dt_off = FP_OFF(&fmtdat);                       /*  フォーマット用データのアドレスの取得  */
                                                        /*  (ｵﾌｾｯﾄ･ｱﾄﾞﾚｽ)      */
        dt_seg = FP_SEG(&fmtdat);                       /*  フォーマット用データのアドレスの取得  */
                                                        /*  (ｾｸﾞﾒﾝﾄ･ｱﾄﾞﾚｽ)      */
        madrss = dt_seg;                                /*  アドレスを物理アドレスへ変換        */
        madrss = (madrss << 4) + dt_off;
        i = 0;
        while(i <= 2)
        {
                adrss[i] = (unsigned char)(madrss >> (8 * i));
                                                        /*  転送アドレスを8ﾋﾞｯﾄごとに分解       */
                ++i;
        }

        outportb(DMODER, 0x4b);                         /*  DMAｺﾝﾄﾛｰﾗのﾓｰﾄﾞ･ﾚｼﾞｽﾀに転送方向をセット  */
        outportb(DCLBFF, 0x00);                         /*  DMAｺﾝﾄﾛｰﾗのｸﾘｱ･ﾊﾞｲﾄ･ﾎﾟｲﾝﾀFFをリセット  */
        outportb(DADDR, adrss[0]);                      /*  DMAｺﾝﾄﾛｰﾗのｱﾄﾞﾚｽ･ﾚｼﾞｽﾀにｱﾄﾞﾚｽ(0-7ﾋﾞｯﾄ)をｾｯﾄ */
        outportb(DADDR, adrss[1]);                      /*  DMAｺﾝﾄﾛｰﾗのｱﾄﾞﾚｽ･ﾚｼﾞｽﾀにｱﾄﾞﾚｽ(8-15ﾋﾞｯﾄ)をｾｯﾄ*/
        outportb(DBNKR, adrss[2]);                      /*  DMAｺﾝﾄﾛｰﾗのｱﾄﾞﾚｽ･ﾚｼﾞｽﾀにｱﾄﾞﾚｽ(16-23ﾋﾞｯﾄ)
                                                        /*  をセット    */
        outportb(DCOUNT, (com_prm->dtl * 4 - 1));       /*  DMAｺﾝﾄﾛｰﾗのｶｳﾝﾄ･ﾚｼﾞｽﾀに転送ﾊﾞｲﾄ数をｾｯﾄ  */
        outportb(DCOUNT, 0);                            /*  DMAｺﾝﾄﾛｰﾗのｶｳﾝﾄ･ﾚｼﾞｽﾀに転送ﾊﾞｲﾄ数をｾｯﾄ  */
        outportb(DWRSMR, 0x03);                         /*  DMAｺﾝﾄﾛｰﾗのﾁｬﾈﾙ3のﾏｽｸを解除する      */
        com_dat[2] = com_prm->n;                        /*  シリンダ番号の設定  */
        com_dat[3] = com_prm->dtl;                      /*  １トラックあたりのセクタ数の設定    */
        com_dat[4] = com_prm->gsl;                      /*  GAP3の書き込みバイト数      */
        com_dat[5] = com_prm->d;                        /*  データ領域に書き込むデータの設定    */
        com_num = 0x06;
        while(time_i != 1);                             /*  モータの回転安定時間待ち    */
        COMMAND(&com_dat[0], com_num);                  /*  ｺﾏﾝﾄﾞ･ﾓｼﾞｭｰﾙ呼びだし        */
        while(int_s != 0x01);                           /*  コマンド終了割り込み待ち    */
        int_s = 0x00;
        if(rslt_pt[0] >= 0x40)                          /*  コマンドの終了結果の判断    */
        {
                if((rslt_pt[0] & 0x08) == 0x08)
                {
                        ret_st = 0x01;                  /*  ドライブのノット・レディ    */
                }
                else
                {
                        if((rslt_pt[1] & 0x10) == 0x10)
                        {
                                ret_st = 0x0a;          /*  オーバラン・エラー  */
```

保守／廃止

(LIO.C) p.23

```
                        }
                        else
                        {
                                ret_st = 0x0c;                  /*  ライト・プロテクト  */
                        }
                }
        }
        else
        {
                ret_st = 0x00;                                  /*  コマンドの正常終了  */
        }
        ret_prm->c = rslt_pt[3];                                /*  コマンド終了時のﾘｻﾞﾙﾄ･ﾊﾟﾗﾒｰﾀのｾｯﾄ  */
        ret_prm->h = rslt_pt[4];
        ret_prm->r = rslt_pt[5];
        ret_prm->n = rslt_pt[6];
        if(ret_st == 0x0a)                                      /*  リターン・ステータスが０ＡＨのとき  */
        {
                --retryc;                                       /*  ﾘﾄﾗｲ･ｶｳﾝﾀをデクリメントしてリトライ  */
        }
        else
        {
                break;                                          /*  そのほかはリトライせずに終了  */
        }
    }
    outportb(DWRSMR, 0x07);                                     /*  DMAｺﾝﾄﾛｰﾗのﾁｬﾈﾙ3をマスクする  */
    return ret_st;
}


        /*****************************************************/
        /*                                                   */
        /*  ベリファイ処理      ＩＮ    : com_code, com_prm  */
        /*                      ＯＵＴ  : ret_st, ret_prm    */
        /*                                                   */
        /*****************************************************/


unsigned char VERIFY(unsigned char com_code, struct compra *com_prm, struct retpra *ret_prm)
{
    unsigned char com_dat[9], ret_st, dvno;
    char com_num;

    dvno = com_code & 0x03;                                     /*  ドライブ番号のデコード  */
    DRVSEL(dvno);                                               /*  ドライブ・セレクト信号の選択  */

    com_dat[0] = 0x16 | (0xe0 & com_code);                      /*  ﾍﾞﾘﾌｧｲ用データのセット  */

    if(com_prm->h == 0x00)                                      /*  ドライブとヘッド番号の設定  */
    {
        com_dat[1] = com_code & 0x03;                           /*  ヘッド番号＝０のときの設定  */
    }
    else
    {
        com_dat[1] = (com_code | 0x04) & 0x07;                  /*  ヘッド番号＝１のときの設定  */
    }
    if((com_code & 0x04) == 0x04)                               /*  ＥＣビットの設定  */
    {
        com_dat[1] = com_dat[1] | 0x80;                         /*  ＥＣビット＝１のときの設定  */
    }
    else
    {
        com_dat[1] = com_dat[1] & 0x7f;                         /*  ＥＣビット＝０のときの設定  */
    }
    com_dat[2] = com_prm->c;                                    /*  シリンダ番号の設定  */
    com_dat[3] = com_prm->h;                                    /*  ヘッド番号の設定  */
    com_dat[4] = com_prm->r;                                    /*  開始セクタ番号の設定  */
    com_dat[5] = com_prm->n;                                    /*  セクタあたりのバイト数の設定  */
    com_dat[6] = com_prm->eot;                                  /*  １トラックあたりのセクタ数の設定  */
    com_dat[7] = com_prm->gsl;                                  /*  GAP3の読み飛ばしバイト数の設定  */
    com_dat[8] = com_prm->dtl;                                  /*  読みだしセクタ数の設定(またはFFH)  */
```

保守／廃止

(L10.C) p.24

```
com_num = 0x09;
while(time_i != 1);                                    /*  モータの回転安定時間待ち    */
COMMAND(&com_dat[0], com_num);                         /*  ｺﾏﾝﾄﾞ･ﾓｼﾞｭｰﾙ呼びだし        */
while(int_s != 0x01);                                  /*  コマンド終了割り込み待ち    */
int_s = 0x00;
if(rslt_pt[0] >= 0x40)                                 /*  コマンドの終了結果の判断    */
{
        if((rslt_pt[0] & 0x08) == 0x08)
        {
                ret_st = 0x01;                         /*  ノット・レディ      */
        }
        else
        {
                switch(rslt_pt[1])
                {
                        case 0x01:
                                if((rslt_pt[2] & 0x01) == 0x01)
                                {
                                        ret_st = 0x08;         /*  ＤＡＭ，ＤＤＡＭ非検出      */
                                }
                                else
                                {
                                        ret_st = 0x02;         /*  ＩＤＡＭ非検出      */
                                }
                                break;
                        case 0x04:
                                switch(rslt_pt[2])
                                {
                                        case 0x02:
                                                ret_st = 0x03;                  /*  指定したシリンダがない  */
                                                break;
                                        case 0x10:
                                                ret_st = 0x04;                  /*  指定したシリンダがない  */
                                                break;
                                        case 0x00:
                                                ret_st = 0x05;                  /*  指定したセクタがない  */
                                                break;
                                        default:
                                                break;
                                }
                                break;
                        case 0x20:
                                if((rslt_pt[2] & 0x20) == 0x20)
                                {
                                        ret_st = 0x07;         /*  データ部のＣＲＣエラー      */
                                }
                                else
                                {
                                        ret_st = 0x06;         /*  ＩＤ部のＣＲＣエラー        */
                                }
                                break;
                        case 0x10:
                                ret_st = 0x0a;                 /*  オーバラン  */
                                break;
                        case 0x80:
                                ret_st = 0x0b;                 /*  最終セクタを越えてアクセスしようとした  */
                                break;
                        default:
                                break;
                }
        }
}
else
{
        if(rslt_pt[2] < 0x40)
        {
                ret_st = 0x00;                         /*  コマンドの正常終了  */
        }
        else
        {
```

保守／廃止

(LIO.C) p.25

```
                ret_st = 0x09;                                  /*  ＤＡＭ付きデータをリードした  */
            }
        }
        ret_prm->c = rslt_pt[3];                                /*  コマンド終了時のﾘｻﾞﾙﾄ･ﾊﾟﾗﾒｰﾀのｾｯﾄ*/
        ret_prm->h = rslt_pt[4];
        ret_prm->r = rslt_pt[5];
        ret_prm->n = rslt_pt[6];
        return ret_st;
}


        /************************************************/
        /*                                              */
        /*  ドライブ・セレクト処理      ＩＮ    : drno  */
        /*                              ＯＵＴ  : なし  */
        /*                                              */
        /************************************************/


void DRVSEL(unsigned char drno)
{
        unsigned char drs;
        union fapnt vector;
        struct REGPACK reg;

        switch(drno)
        {
                case 0:                                         /*  ドライブ０のとき    */
                        drs = 0x1c;
                        break;
                case 1:                                         /*  ドライブ１のとき    */
                        drs = 0x3d;
                        break;
                case 2:                                         /*  ドライブ２のとき    */
                        drs = 0x5e;
                        break;
                case 3:                                         /*  ドライブ３のとき    */
                        drs = 0x9f;
                        break;
        }
        outportb(DOR70, drs);                                   /*  ＤＯＲへのライト    */
        vector.pnt = (unsigned long)TIME;
        reg.r_ax = 0x0200;
        reg.r_cx = 38;
        reg.r_es = vector.pon.adseg;
        reg.r_bx = vector.pon.adoff;
        intr(0x1c, &reg);
}


        /************************************************/
        /*                                              */
        /*  タイマ割り込み処理  ＩＮ    : なし         */
        /*                      ＯＵＴ  : なし         */
        /*                                              */
        /************************************************/


void interrupt TIME(void)
{
        time_i = 1;

}
```

●DEVICE.C

(DEVICE.C) p.1

```
#include<dos.h>

#define             DOR70          0x00d4                    /*  μPD72070のﾃﾞｨｼﾞﾀﾙ･ｱｳﾄﾌﾟｯﾄ･ﾚｼﾞｽﾀのIOｱﾄﾞﾚｽ*/
#define             DIR70          0x00de                    /*  μPD72070のﾃﾞｨｼﾞﾀﾙ･ｲﾝﾌﾟｯﾄ･ﾚｼﾞｽﾀのIOｱﾄﾞﾚｽ */


 struct command_pac
{
     char command_length;                                    /*  コマンド・パケットのバイト数        */
     char unit_num;                                          /*  論理装置コード      */
     char cmd;                                               /*  Ｉ／Ｏリクエスト・コマンド・コード  */

                 /*  ステータスのビット・フィールド(1ﾜｰﾄﾞ)  */

     struct
     {
            unsigned err_cd:8;                               /*  エラー・コード(8ﾋﾞｯﾄ)       */
            unsigned don:1;                                  /*  doneビット  */
            unsigned bus:1;                                  /*  busyビット  */
            unsigned resvbit:5;                              /*  予約域(5ﾋﾞｯﾄ)       */
            unsigned err:1;                                  /*  エラー・ビット      */
     } st;
     long resv1, resv2;                                      /*  予約域（8ﾊﾞｲﾄ）     */

     union
     {
                 /*  BUILD BPBのコマンド・パケット  */

          struct
          {
                 unsigned char mediabpb;                     /*  ＤＰＢへのﾒﾃﾞｨｱ･ﾃﾞｨｽｸﾘﾌﾟﾀ  */
                 int transb_off;                             /*  転送アドレス        */
                 int transb_seg;
                 struct bpb_st *bpb_off;                     /*  ＢＰＢに対するポインタ      */
                 char *bpb_seg;
          } bb;

                 /*  リード，ライトのコマンド・パケット  */

          struct
          {
                 unsigned char mediarw;                      /*  ＤＰＢへのﾒﾃﾞｨｱ･ﾃﾞｨｽｸﾘﾌﾟﾀ  */
                 unsigned char *trns_off;                    /*  転送アドレス        */
                 char *trns_seg;
                 unsigned int sect_coun;                     /*  セクタ・カウント    */
                 unsigned int start_sec;                     /*  開始セクタ  */
          } rw;

                 /*  メディア・チェックのコマンド・パケット  */

          struct
          {
                 char mediamc;                               /*  ＤＰＢへのﾒﾃﾞｨｱ･ﾃﾞｨｽｸﾘﾌﾟﾀ  */
                 int remove;                                 /*  返す値      */
          } mc;

                 /*  イニシャライズのコマンド・パケット  */

          struct
          {
                 char sp_unit;                               /*  ユニット数  */
                 char *end_off;                              /*  ブレーク・アドレス  */
                 char *end_seg;
                 struct bpb_st **bpboff;                     /*  ＢＰＢの配列ポインタ        */
                 char *bpbseg;
          } intia;

     } prm;

};
```

保守／廃止

(DEVICE.C) p.2

```
            /*  ＢＰＢの構造体  */

struct bpb_st
{
      unsigned int b_sector;                                  /*  １セクタあたりのバイト数    */
      unsigned char sector_c;                                 /*  １クラスタあたりのセクタ数  */
      unsigned int resv_sector;                               /*  予約セクタ数        */
      unsigned char fat_num;                                  /*  ＦＡＴ数    */
      unsigned int dir_num;                                   /*  ルート・ディレクトリの総エントリ数  */
      unsigned int total_sct;                                 /*  総論理セクタ数      */
      unsigned char med_id;                                   /*  メディア・ディスクリプタ    */
      unsigned int fat_sct;                                   /*  ＦＡＴのセクタ数    */
      unsigned int track_sec;                                 /*  １トラックあたりのセクタ数  */
      unsigned int hed_num;                                   /*  ヘッド数    */
      unsigned int secret;                                    /*  隠れたセクタ数      */
};

struct d_bpb
{
      char sbr1, sbr2, sbr3;                                  /*  ＳＨＯＲＴブランチ  */
      double oem_name;                                        /*  ＯＥＭメーカ名      */
      unsigned int byt_sec;                                   /*  １セクタあたりのバイト数    */
      unsigned char sect_c;                                   /*  １クラスタあたりのセクタ数  */
      unsigned int rsv_sect;                                  /*  予約セクタ数        */
      unsigned char fat;                                      /*  ＦＡＴ数    */
      unsigned int dir;                                       /*  ルート・ディレクトリの総エントリ数  */
      unsigned int all_sct;                                   /*  総論理セクタ数      */
      unsigned char med_dsl;                                  /*  メディア・ディスクリプタ    */
      unsigned int sect_fat;                                  /*  ＦＡＴのセクタ数    */
      unsigned int tr_sect;                                   /*  １トラックあたりのセクタ数  */
      unsigned int hed_n;                                     /*  ヘッド数    */
      unsigned int sec_sect;                                  /*  隠れたセクタ数      */
};

            /*  コマンド・パラメータの構造体  */

struct compra
{
      union                                                   /*  転送バイト数の共用体        */
      {
            unsigned int dtsize;                              /*  転送データのバイト数(int型) */
            struct                                            /*  int型をchar型にわける構造体 */
            {
                  unsigned char kai;                          /*  下位８ビット分      */
                  unsigned char jyoui;                        /*  上位８ビット分      */
            } int_hen;

      } trsize;

      unsigned int dt_off;                                    /*  転送ﾃﾞｰﾀのｵﾌｾｯﾄ･ｱﾄﾞﾚｽ      */
      unsigned int dt_seg;                                    /*  転送ﾃﾞｰﾀのｾｸﾞﾒﾝﾄ･ｱﾄﾞﾚｽ      */
      unsigned char c;                                        /*  シリンダ番号        */
      unsigned char h;                                        /*  ヘッド番号  */
      unsigned char r;                                        /*  開始セクタ番号      */
      unsigned char n;                                        /*  １セクタのデータ長  */
      unsigned char dtl;                                      /*  1ｾｸﾀあたりの処理ﾃﾞｰﾀ長     */
      unsigned char gsl;                                      /*  GAP3の読み飛ばしバイト数    */
      unsigned char eot;                                      /*  １トラックのセクタ数        */
      unsigned char d;                                        /*  ﾌｫｰﾏｯﾄの書きこみﾃﾞｰﾀ･ﾊﾟﾀｰﾝ  */
};


            /*  ﾘﾀｰﾝ･ｽﾃｰﾀｽ･ﾊﾟﾗﾒｰﾀの構造体    */

struct retpra
{
      unsigned char c;                                        /*  実行終了後のシリンダ番号    */
      unsigned char h;                                        /*  実行終了後のヘッド番号      */
```

保守／廃止

(DEVICE.C) p.3

```
	unsigned char r;                                        /*  実行終了後のセクタ番号      */
	unsigned char n;                                        /*  1セクタのデータ長  */
};


		/*  シーク・リザルトの構造体    */

struct sr
{
	unsigned char dr_st;                                    /*  シーク・コマンド実行後のﾘｻﾞﾙﾄ･ｽﾃｰﾀｽ */
	unsigned char dr_pcn;                                   /*  シーク・コマンド終了後のヘッドの位置        */
};

union intg                                                  /*  転送バイト数の共用体        */
{
	unsigned int dtsize;                                    /*  転送データのバイト数(int型) */
	struct                                                  /*  int型をchar型にわける構造体 */
	{
		unsigned char kai;                              /*  下位８ビット分      */
		unsigned char jyoui;                            /*  上位８ビット分      */
	} int_hen;
};

void IO_REQ(void);
void INITIAL(void);
void MD_CHEK(void);
void BUILD_BPB(void);
void IOCTL_IN(void);
void INPUT(void);
void OUTPUT(void);
void OUTPUT_V(void);
void ERROR_IO(void);
unsigned char UNIT_NO(void);
void ERRSET(unsigned char, struct retpra *, unsigned char, unsigned char);

void D_INT(void);
unsigned char MEDIAT(unsigned char);
unsigned char RECAL(unsigned char);
unsigned char SEEK(unsigned char, struct compra *);
unsigned char RSEEK(unsigned char, struct compra *);
unsigned char RDDAT(unsigned char, struct compra *, struct retpra *);
unsigned char RDID(unsigned char, struct retpra *);
unsigned char WRDAT(unsigned char, struct compra *, struct retpra *);
unsigned char VERIFY(unsigned char, struct compra *, struct retpra *);
void DRVSEL(unsigned char);
void HENKAN (unsigned char *, unsigned char);


extern struct command_pac far *packet;
extern char *pro_off;
extern char *pro_seg;
extern unsigned char d_exist;
extern unsigned char ready;
extern unsigned char rslt_pt[];
extern unsigned char time_i;

char unit_ex;
struct bpb_st bpbmx[4];
struct bpb_st *bpb_d[4];



		/***********************************************/
		/*                                             */
		/*      Ｉ／Ｏリクエスト処理    ＩＮ  : packet  */
		/*                              ＯＵＴ：なし   */
		/*                                             */
		/***********************************************/
```

保守／廃止

(DEVICE.C) p.4

```
void IO_REQ(void)
{
      switch(packet->cmd)                                    /*  Ｉ／Ｏリクエスト・コマンド・コードに従って  */
                                                             /*  各処理を呼びだす  */
      {
            case 0x00:
                  INITIAL();                                 /*  イニシャライズ処理呼び出し  */
                  break;
            case 0x01:
                  MD_CHEK();                                 /*  メディア・チェック処理呼び出し  */
                  break;
            case 0x02:
                  BUILD_BPB();                               /*  ＢＵＩＬＤ　ＢＰＢ処理呼び出し  */
                  break;
            case 0x03:
                  IOCTL_IN();                                /*  コントロール文字列のインプット処理呼び出し  */
                  break;
            case 0x04:
                  INPUT();                                   /*  インプット処理呼び出し  */
                  break;
            case 0x08:
                  OUTPUT();                                  /*  アウトプット処理呼び出し  */
                  break;
            case 0x09:
                  OUTPUT_V();                                /*  ｱｳﾄﾌﾟｯﾄ･ｳｨｽﾞ･ﾍﾞﾘﾌｧｲ処理呼び出し  */
                  break;
            default:
                  ERROR_IO();                                /*  Ｉ／Ｏリクエスト・エラー処理呼び出し  */
      }
      return;
}


            /*********************************************************/
            /*                                                       */
            /*      イニシャライズ処理       ＩＮ   : pro_seg, pro_off */
            /*                               ＯＵＴ : packet          */
            /*                                                       */
            /*********************************************************/


void INITIAL(void)
{
      static unsigned int w[30] = {'μ','Ｐ','Ｄ','７','２','０','７','０','デ','バ','イ','ス','・','ド','ラ','イ','バ',
'を','組','み','込','み','ま','し','た','¥x0d¥x0a','$'};

      unsigned char i;

      _AH = 0x09;
      _DX = (unsigned int)w;
      geninterrupt(0x21);

      D_INT();                                               /*  D_INTモジュール呼び出し  */

      for(i =0; i < 4; ++i)                                  /*  ＢＰＢ配列に２ＨＤ(18ｾｸﾀ)のデータをセット  */
      {
            bpbmx[i].b_sector = 512;                         /*  １セクタあたりのバイト数  */
            bpbmx[i].sector_c = 1;                           /*  １クラスタあたりのセクタ数  */
            bpbmx[i].resv_sector = 1;                        /*  予約セクタ数  */
            bpbmx[i].fat_num = 2;                            /*  ＦＡＴ数  */
            bpbmx[i].dir_num = 224;                          /*  ルート・ディレクトリの総エントリ数  */
            bpbmx[i].total_sct = 2880;                       /*  総論理セクタ数  */
            bpbmx[i].med_id = 0xf0;                          /*  メディア・ディスクリプタ  */
            bpbmx[i].fat_sct = 9;                            /*  ＦＡＴのセクタ数  */
            bpbmx[i].track_sec = 18;                         /*  １トラックあたりのセクタ数  */
            bpbmx[i].hed_num = 2;                            /*  ヘッド数  */
            bpbmx[i].secret = 0;                             /*  隠れたセクタ数  */
      }

      i = 0;                                                 /*  カウンタの初期化  */
```

(DEVICE.C) p.5

```
	unit_ex = 0;                                            /*  接続ドライブ数の初期化       */
	while(i < 4)                                            /*  接続ドライブ数のチェック(4台まで) */
	{
		if(((d_exist >> i) & 0x01) == 0x01)             /*  d_existをi回右ｼﾌﾄして最下位ﾋﾞｯﾄが1ならば*/
		{
			++unit_ex;                              /*  接続ドライブ数をインクリメントする  */
		}
		++i;
	}

	for(i = 0; i < 4; ++i)
	{
		bpb_d[i] = &bpbmx[i];
	}
	packet->prm.intia.bpboff = &bpb_d[0];                   /*  ＢＰＢのﾎﾟｲﾝﾀ配列へのｱﾄﾞﾚｽをpacketへｾｯﾄ  */
	packet->prm.intia.bpbseg = pro_seg;

	packet->prm.intia.sp_unit = unit_ex;                    /*  packetのﾕﾆｯﾄ数に接続ﾄﾞﾗｲﾌﾞ数をｾｯﾄ  */
	packet->prm.intia.end_off = pro_off;                    /*  packetのブレーク・アドレスをセット  */
	packet->prm.intia.end_seg = pro_seg;
	packet->st.don = 1;                                     /*  packetのステータスのdoneビットに1をセット  */

	outportb(DOR70, 0x0c);                                  /*  モータの停止  */
	time_i = 0;                                             /*  タイマ割り込みフラグのクリア  */

	return;
}



	/************************************************************/
	/*                                                          */
	/*  ＢＵＩＬＤ　ＢＰＢ処理      ＩＮ   : packet,pro_seg      */
	/*                               ＯＵＴ : packet             */
	/*                                                          */
	/************************************************************/


void BUILD_BPB(void)
{
	unsigned char ret_st, com_code, drvno, media;
	struct compra com_prm;
	struct retpra ret_prm;
	struct d_bpb bpb_dat;



	drvno = UNIT_NO();

	if(drvno > 3)                                           /*  ユニット番号が無効だった場合  */
	{
		packet->st.don = 1;                             /*  packetのステータスのdoneビットを    */
		return;                                         /*  セットしてリターン  */
	}

	bpb_dat.byt_sec = 0;

	com_code = 0x10 | drvno;
	ret_st = RECAL(com_code);                               /*  リキャリブレイトを実行       */
	if(ret_st != 0x00)                                      /*  リキャリブレイトが異常終了のとき    */
	{
		ERRSET(ret_st, &ret_prm, 0, 0);                 /*  ｴﾗｰ･ｾｯﾄ･ﾓｼﾞｭｰﾙを呼び出す    */
		packet->st.don = 1;                             /*  packetのdoneビットに1をセットしてﾘﾀｰﾝ  */
		return;
	}

	media = 0;
	while(media < 3)                                        /*  μPD72070のメディアの設定を変えて   */
	{                                                       /*  メディアの種類を確認する    */
		switch(media)
```

保守／廃止

(DEVICE.C) p.6

```
        {
                case 0:
                        MEDIAT(0x01);                   /*  メディアを２ＨＤの設定にする        */
                        break;
                case 1:
                        MEDIAT(0x02);                   /*  メディアを２ＥＤの設定にする        */
                        break;
                case 2:
                        MEDIAT(0x00);                   /*  メディアを２ＤＤの設定にする        */
                        break;
        }
        com_code = 0x44 | drvno;
        ret_st = RDID(com_code, &ret_prm);              /*  リードＩＤ処理を実行        */


        if(ret_st == 0)                                 /*  リードＩＤ処理が正常終了ならば      */
        {                                               /*  ＷＨＩＬＥループから抜ける  */
                break;                                  /*  （メディアの種類の確認終了）        */
        }
        ++media;                                        /*  次のメディア設定で再実行する        */
}

com_code = 0x60 | drvno;                                /*  予約ｾｸﾀのﾘｰﾄﾞ(31ﾊﾞｲﾄ)の各ﾊﾟﾗﾒｰﾀの設定  */
com_prm.c = 0;                                          /*  シリンダ番号        */
com_prm.h = 0;                                          /*  ヘッド番号  */
com_prm.r = 1;                                          /*  セクタ番号  */
com_prm.n = 2;                                          /*  セクタあたりのバイト数      */
com_prm.dt_seg = (unsigned int)pro_seg;                 /*  転送アドレス(ｾｸﾞﾒﾝﾄ)        */
com_prm.dt_off = (unsigned int)&bpb_dat;                /*  転送アドレス(ｵﾌｾｯﾄ) */
com_prm.trsize.dtsize = 30;                             /*  転送サイズ  */
com_prm.gsl = 0x27;                                     /*  GAP3読み飛ばしバイト数      */
switch(media)                                           /*  トラックあたりのセクタ数    */
{
        case 0:
                com_prm.eot = 18;                       /*  ２ＨＤのセクタ数    */
                break;
        case 1:
                com_prm.eot = 36;                       /*  ２ＥＤのセクタ数    */
                break;
        case 2:
                com_prm.eot = 9;                        /*  ２ＤＤのセクタ数    */
                break;
}
ret_st = RDDAT(com_code, &com_prm, &ret_prm);           /*  リード・データ実行  */

if(ret_st == 0)                                         /*  リードが正常終了    */
{                                                       /*  BPBの構造体（配列）に各値をセット  */

        bpbmx[packet->unit_num].b_sector = bpb_dat.byt_sec;
                                                        /*  バイト数／セクタ    */

        bpbmx[packet->unit_num].sector_c = bpb_dat.sect_c;
                                                        /*  セクタ数／クラスタ  */

        bpbmx[packet->unit_num].resv_sector = bpb_dat.rsv_sect;
                                                        /*  予約セクタ数        */

        bpbmx[packet->unit_num].fat_num = bpb_dat.fat;
                                                        /*  ＦＡＴ数    */

        bpbmx[packet->unit_num].dir_num = bpb_dat.dir;
                                                        /*  ルート・ディレクトリのエントリ数    */

        bpbmx[packet->unit_num].total_sct = bpb_dat.all_sct;
                                                        /*  総論理セクタ数      */

        bpbmx[packet->unit_num].med_id = bpb_dat.med_dsl;
                                                        /*  メディア・ディスクリプタ    */

        bpbmx[packet->unit_num].fat_sct = bpb_dat.sect_fat;
```

保守／廃止

(DEVICE.C) p.7

```
                                                        /*  セクタ数／ＦＡＴ    */

            bpbmx[packet->unit_num].track_sec = bpb_dat.tr_sect;
                                                        /*  １トラックあたりのセクタ数  */

            bpbmx[packet->unit_num].hed_num = bpb_dat.hed_n;
                                                        /*  ヘッドの数  */

            bpbmx[packet->unit_num].secret = bpb_dat.sec_sect;
                                                        /*  隠れたセクタの数    */
            packet->prm.bb.bpb_seg = pro_seg;           /*  packetにBPBの構造体へのポインタをセット  */

            packet->prm.bb.bpb_off = &bpbmx[packet->unit_num];
            packet->prm.bb.mediabpb = bpbmx[packet->unit_num].med_id;
                                                        /*  packetにメディア・ディスクリプタをセット  */

      }
      else                                              /*  リードが異常終了    */
      {
            ERRSET(ret_st, &ret_prm, 0, 0);             /*  ｴﾗｰ･ｾｯﾄ･ﾓｼﾞｭｰﾙを呼び出す    */
      }
      packet->st.don = 1;                               /*  packetのdoneビットに１をセットする  */
      outportb(DOR70, 0x0c);                            /*  モータの停止        */
      time_i = 0;                                       /*  タイマ割り込みフラグのクリア        */
      return;
}


            /*****************************************************/
            /*                                                   */
            /*  メディア・チェック処理      ＩＮ  : packet       */
            /*                              ＯＵＴ : packet      */
            /*                                                   */
            /*****************************************************/


void MD_CHEK(void)
{
      unsigned char drvno, dskchg1, dskchg2;
      struct compra com_prm;
      drvno = UNIT_NO();                                /*  ユニット番号処理モジュール呼びだし  */
      if(drvno > 3)                                     /*  ユニット番号が無効だった場合        */
      {
            packet->st.don = 1;                         /*  packetのステータスのdoneビットを    */
            return;                                     /*  セットしてリターン*/
      }
      DRVSEL(drvno);
      dskchg1 = inportb(DIR70);
      if((dskchg1 & 0x80) == 0x80)
      {
            packet->prm.mc.remove = 0xff;               /*  packetの返す値に-1(FFH)をセットする */
            com_prm.c = 2;
            SEEK(drvno, &com_prm);
            RECAL(drvno);
            dskchg2 = inportb(DIR70);
            if((dskchg2 & 0x80) == 0x80)
            {
                  packet->st.err = 1;                   /*  packetのerrビットに１をセットする    */
                  packet->st.err_cd = 0x02;             /*  ｴﾗｰ･ｺｰﾄﾞにﾄﾞﾗｲﾌﾞのﾉｯﾄ･ﾚﾃﾞｨをセットする  */
            }
      }
      else
      {
            packet->prm.mc.remove = 0x01;               /*  packetの返す値に1をセットする       */
      }

      packet->st.don = 1;                               /*  packetのステータスのdoneビットをｾｯﾄする  */
      outportb(DOR70, 0x0c);                            /*  モータの停止        */
      time_i = 0;                                       /*  タイマ割り込みフラグのクリア        */
      return;
```

保守／廃止

(DEVICE.C) p.8

```
}

            /*******************************************/
            /*                                         */
            /*  インプット処理       ＩＮ  : packet     */
            /*                       ＯＵＴ : packet     */
            /*                                         */
            /*******************************************/


void INPUT(void)
{
      unsigned char ret_st, com_code, drvno, tr_sect, st_slnd, st_sect;
      unsigned char hed_num, seek_on, set_sect;
      unsigned int strsect, sec_count, rd_sect;
      struct compra com_prm;
      struct retpra ret_prm;

      drvno = UNIT_NO();
      if(drvno > 3)                                          /*  ユニット番号が無効だった場合        */
      {
            packet->prm.rw.sect_coun = 0;
            packet->st.don = 1;                              /*  packetのｽﾃｰﾀｽのdonetﾞｯﾄをｾｯﾄしてﾘﾀｰﾝ  */
            return;
      }
      strsect = packet->prm.rw.start_sec;                    /*  ｽﾀｰﾄ･ｾｸﾀ(論理ｾｸﾀ)packetから取り出す */
      tr_sect = bpbmx[packet->unit_num].track_sec;           /*  BPBから'セクタ数／トラック'を取り出す  */
      st_slnd = strsect / (tr_sect * 2);                     /*  ｽﾀｰﾄ･ｼﾘﾝﾀﾞをｽﾀｰﾄ･ｾｸﾀから求める       */
      st_sect = (strsect % tr_sect) + 1;                     /*  ｽﾀｰﾄ･ｾｸﾀ(物理ｾｸﾀ)をｽﾀｰﾄ･ｾｸﾀ(論理ｾｸﾀ)から求める  */
      hed_num = (strsect / tr_sect) % 2;                     /*  ヘッド番号をｽﾀｰﾄ･ｾｸﾀ(論理ｾｸﾀ)から求める  */
      sec_count = packet->prm.rw.sect_coun;                  /*  転送セクタ数をpacketから取り出す     */
      rd_sect = 0;                                           /*  読み込み済みセクタ数カウンタをクリア  */

      do
      {
            com_code = drvno;
            com_prm.c = st_slnd;
            ret_st = SEEK(com_code, &com_prm);               /*  読み出すシリンダへシークする        */
            if(ret_st != 0x00)                               /*  シークが異常終了のとき       */
            {
                  ERRSET(ret_st, &ret_prm, 0, 0);            /*  ｴﾗｰ･ｾｯﾄ･ﾓｼﾞｭｰﾙを呼び出す    */
                  packet->prm.rw.sect_coun = rd_sect;
                  packet->st.don = 1;                        /*  packetのdoneビットに１をセットしてリターン  */
                  return;
            }

                  /*********************************/
                  /*  次シリンダへのシークの判別  */
                  /*********************************/

            if(hed_num == 0)                                 /*  ヘッド番号が０のとき         */
            {
                  if((sec_count + st_sect - 1) > (tr_sect * 2))
                                                           /*  転送ｾｸﾀ数＞ｽﾀｰﾄ･ｼﾘﾝﾀﾞの残りｾｸﾀ数のとき  */
                  {
                        seek_on = 1;                         /*  次シリンダへシークする       */
                  }
                  else                                       /*  転送ｾｸﾀ数≦ｽﾀｰﾄ･ｼﾘﾝﾀﾞの残りｾｸﾀ数のとき  */
                  {
                        seek_on = 0;                         /*  次シリンダへのシークは行わない       */
                  }
            }
            else                                             /*  ヘッド番号が１のとき         */
            {
                  if((sec_count + st_sect - 1) > tr_sect)    /*  転送ｾｸﾀ数＞ｽﾀｰﾄ･ﾄﾗｯｸの残りｾｸﾀ数のとき  */
                  {
                        seek_on = 1;                         /*  次シリンダへシークする       */
                  }
                  else                                       /*  転送ｾｸﾀ数≦ｽﾀｰﾄ･ﾄﾗｯｸの残りｾｸﾀ数のとき  */
                  {
```

保守／廃止

(DEVICE.C) p.9

```
                        seek_on = 0;                            /*  次シリンダへのシークは行わない      */
                }
        }
        if(seek_on == 1)                                        /*  シークを行う場合の読み込ｾｸﾀ数のセット    */
        {
                if(hed_num == 0)                                /*  開始セクタのヘッド番号が0のとき    */
                {
                        set_sect = tr_sect * 2 - st_sect + 1;
                                                                /*  1シリンダのｾｸﾀ数－ｽﾀｰﾄ･ｾｸﾀ(物理ｾｸﾀ)＋1    */
                }
                else                                            /*  開始セクタのヘッド番号が1のとき    */
                {
                        set_sect = tr_sect - st_sect + 1;       /*  1トラックのセクタ数－ｽﾀｰﾄ･ｾｸﾀ(物理ｾｸﾀ)＋1 */
                }
        }
        else                                                    /*  ｼｰｸを行わない場合の読み込ｾｸﾀ数のｾｯﾄ  */
        {
                set_sect = (unsigned char)sec_count;            /*  転送セクタ数をセット        */
        }

        com_code = 0xf0 | drvno;                                /*  リード・データのパラメータをセットする  */
        com_prm.c = st_slnd;                                    /*  開始シリンダ番号    */
        com_prm.h = hed_num;                                    /*  ヘッド番号  */
        com_prm.r = st_sect;                                    /*  開始セクタ番号(物理ｾｸﾀ)    */
        com_prm.eot = tr_sect;                                  /*  トラックあたりのセクタ数    */
        com_prm.n = 2;                                          /*  セクタあたりのバイト数      */
        com_prm.gsl = 0x1b;                                     /*  GAP3の読み飛ばしバイト数    */

        com_prm.dt_off = (unsigned int)packet->prm.rw.trns_off + rd_sect * 512;
                                                                /*  転送アドレス(ｵﾌｾｯﾄ) */
        com_prm.trsize.dtsize = set_sect * 512 ;                /*  転送バイト数        */

        com_prm.dt_seg = (unsigned int)packet->prm.rw.trns_seg;
                                                                /*  転送アドレス(ｾｸﾞﾒﾝﾄ)        */

        ret_st = RDDAT(com_code, &com_prm, &ret_prm);
                                                                /*  リード・データ実行  */
        if(ret_st != 0x00)                                      /*  リード・データが異常終了のとき      */
        {
                ERRSET(ret_st, &ret_prm, 1, tr_sect);           /*  ｴﾗｰ･ｾｯﾄ･ﾓｼﾞｭｰﾙを呼び出す    */
                packet->st.don = 1;                             /*  packetのdoneビットに1をセットしてリターン  */

                return;
        }

        rd_sect += set_sect;                                    /*  読み込み済みカウンタを更新する      */
        st_sect = 1;                                            /*  開始ｾｸﾀ番号を1にセット      */
        hed_num = 0;                                            /*  ヘッド番号を0にセット       */
        ++st_slnd;                                              /*  シリンダ番号をインクリメント        */
        sec_count -= set_sect;                                  /*  未読セクタを更新する        */

    }
    while(seek_on != 0);                                        /*  シークを行わなくなるまでリードを繰り返す  */

    packet->prm.rw.sect_coun = rd_sect;                         /*  packetに転送終了セクタ数をセット    */
    packet->st.don = 1;                                         /*  packetのdoneビットに1をセットする  */
    outportb(DOR70, 0x0c);                                      /*  モータの停止        */
    time_i = 0;                                                 /*  タイマ割り込みフラグのクリア        */
    return;
}



        /*******************************************/
        /*                                         */
        /*  アウトプット処理    ＩＮ  : packet     */
        /*                      ＯＵＴ : packet    */
        /*                                         */
        /*******************************************/
```

(DEVICE.C) p.10

```
void OUTPUT(void)
{
      unsigned char ret_st, com_code, drvno, tr_sect, st_slnd;
      unsigned char st_sect, hed_num, seek_on, set_sect;
      unsigned int strsect, sec_coun, wr_sect;
      unsigned int i;
      struct compra com_prm;
      struct retpra ret_prm;

      drvno = UNIT_NO();                                        /*  ユニット番号処理     */
      if(drvno > 3)                                             /*  ユニット番号が無効だった場合  */
      {
            packet->prm.rw.sect_coun = 0;
            packet->st.don = 1;                                 /*  packetのステータスのdoneビットを     */
            return;                                             /*  セットしてリターン  */
      }
      strsect = packet->prm.rw.start_sec;                       /*  ｽﾀｰﾄ･ｾｸﾀ(論理ｾｸﾀ)packetから取り出す */
      tr_sect = bpbmx[packet->unit_num].track_sec;              /*  BPBから'セクタ数／トラック'を取り出す        */
      st_slnd = strsect / (tr_sect * 2);                        /*  ｽﾀｰﾄ･ｼﾘﾝﾀﾞをｽﾀｰﾄ･ｾｸﾀから求める       */
      st_sect = (strsect % tr_sect) + 1;                        /*  ｽﾀｰﾄ･ｾｸﾀ(物理ｾｸﾀ)をｽﾀｰﾄ･ｾｸﾀ(論理ｾｸﾀ)から求める  */
      hed_num = (strsect / tr_sect) % 2;                        /*  ヘッド番号をｽﾀｰﾄ･ｾｸﾀ(論理ｾｸﾀ)から求める      */
      sec_coun = packet->prm.rw.sect_coun;                      /*  転送セクタ数をpacketから取り出す     */
      wr_sect = 0;                                              /*  書き込み済みセクタ数カウンタをクリア         */

      do
      {
            com_code = drvno;
            com_prm.c = st_slnd;
            ret_st = SEEK(com_code, &com_prm);                  /*  書き込むシリンダへシークする         */
            if(ret_st != 0x00)                                  /*  シークが異常終了のとき       */
            {
                  ERRSET(ret_st, &ret_prm, 0, 0);               /*  ｴﾗｰ･ｾｯﾄ･ﾓｼﾞｭｰﾙを呼び出す     */
                  packet->prm.rw.sect_coun = wr_sect;
                  packet->st.don = 1;                           /*  packetのdoneビットに１をセットしてリターン  */
                  return;
            }

                  /*********************************/
                  /*  次シリンダへのシークの判別  */
                  /*********************************/

            if(hed_num == 0)                                    /*  ヘッド番号が０のとき         */
            {
                  if((sec_coun + st_sect - 1) > (tr_sect * 2))
                                                              /*  転送ｾｸﾀ数＞ｽﾀｰﾄ･ｼﾘﾝﾀﾞの残りｾｸﾀ数のとき       */
                  {
                        seek_on = 1;                            /*  次シリンダへシークする       */
                  }
                  else                                        /*  転送ｾｸﾀ数≦ｽﾀｰﾄ･ｼﾘﾝﾀﾞの残りｾｸﾀ数のとき       */
                  {
                        seek_on = 0;                            /*  次シリンダへのシークは行わない       */
                  }
            }
            else                                              /*  ヘッド番号が１のとき         */
            {
                  if((sec_coun + st_sect - 1) > tr_sect)        /*  転送ｾｸﾀ数＞ｽﾀｰﾄ･ﾄﾗｯｸの残りｾｸﾀ数のとき       */
                  {
                        seek_on = 1;                            /*  次シリンダへシークする       */
                  }
                  else                                        /*  転送ｾｸﾀ数≦ｽﾀｰﾄ･ﾄﾗｯｸの残りｾｸﾀ数のとき       */
                  {
                        seek_on = 0;                            /*  次シリンダへのシークは行わない       */
                  }
            }
            if(seek_on == 1)                                  /*  シークを行う場合の読み込みｾｸﾀ数のセット     */
            {
                  if(hed_num == 0)                            /*  開始セクタのヘッド番号が０のとき     */
                  {
                        set_sect = tr_sect * 2 - st_sect + 1;
```

```
                                                        /*  1ｼﾘﾝﾀﾞのｾｸﾀ数－ｽﾀｰﾄ･ｾｸﾀ(物理ｾｸﾀ)＋1    */
                }
                else                                    /*  開始セクタのヘッド番号が1のとき    */
                {
                        set_sect = tr_sect - st_sect + 1;       /*  1ﾄﾗｯｸのｾｸﾀ数－ｽﾀｰﾄ･ｾｸﾀ(物理ｾｸﾀ)＋1    */
                }
        }
        else                                            /*  シークを行わない場合の書き込みｾｸﾀ数のセット */
        {
                set_sect = (unsigned char)sec_coun;     /*  転送セクタ数をセット        */
        }

        com_code = 0xd0 | drvno;                        /*  ライト・データのパラメータをセットする      */
        com_prm.c = st_slnd;                            /*  開始シリンダ番号    */
        com_prm.h = hed_num;                            /*  ヘッド番号  */
        com_prm.r = st_sect;                            /*  開始セクタ番号(物理ｾｸﾀ)    */
        com_prm.n = 2;                                  /*  セクタあたりのバイト数      */
        com_prm.gsl = 0x1b;                             /*  GAP3の読み飛ばしバイト数    */

        com_prm.dt_off = (unsigned int)packet->prm.rw.trns_off + wr_sect * 512;
                                                        /*  転送アドレス(ｵﾌｾｯﾄ) */
        com_prm.trsize.dtsize = set_sect * 512;         /*  転送バイト数        */
        com_prm.eot = tr_sect;                          /*  トラックあたりのセクタ数    */

        com_prm.dt_seg = (unsigned int)packet->prm.rw.trns_seg;
                                                        /*  転送アドレス(ｾｸﾞﾒﾝﾄ)       */

        ret_st = WRDAT(com_code, &com_prm, &ret_prm);
                                                        /*  ライト・データ実行  */

        for(i = 0; i < 0x278; ++i);                     /*  ６００μｓのウエイト        */

        if(ret_st != 0x00)                              /*  ライト・データが異常終了のとき      */
        {
                ERRSET(ret_st, &ret_prm, 2, tr_sect);   /*  ｴﾗｰ･ｾｯﾄ･ﾓｼﾞｭｰﾙを呼び出す    */
                packet->st.don = 1;                     /*  packetのdoneビットに1をセットしてリターン  */
                return;
        }
        wr_sect += set_sect;                            /*  書き込み済みカウンタを更新する      */
        st_sect = 1;                                    /*  開始ｾｸﾀ番号を1にセット      */
        hed_num = 0;                                    /*  ヘッド番号を0にセット       */
        ++st_slnd;                                      /*  シリンダ番号をインクリメント        */
        sec_coun -= set_sect;                           /*  書き込み未終了セクタを更新する      */

    }
    while(seek_on != 0);                                /*  シークを行わなくなるまでライトを繰り返す    */

    packet->prm.rw.sect_coun = wr_sect;                 /*  packetに転送終了セクタ数をセット    */
    packet->st.don = 1;                                 /*  packetのdoneビットに1をセットする  */
    outportb(DOR70, 0x0c);                              /*  モータの停止        */
    time_i = 0;                                         /*  タイマ割り込みフラグのクリア        */
    return;
}



        /*****************************************************************/
        /*                                                               */
        /*  アウトプット・ウィズ・ベリファイ処理        ＩＮ  : packet  */
        /*                                              ＯＵＴ : packet  */
        /*                                                               */
        /*****************************************************************/


void OUTPUT_V(void)
{
    unsigned char ret_st, com_code, drvno, tr_sect, st_slnd;
    unsigned char st_sect, hed_num, seek_on, set_sect;
    unsigned int strsect, sec_coun, wv_sect;
    struct compra com_prm;
```

(DEVICE.C) p.12

```
struct retpra ret_prm;

drvno = UNIT_NO();                                      /*  ユニット番号処理     */
if(drvno > 3)                                           /*  ユニット番号が無効だった場合         */
{
        packet->prm.rw.sect_coun = 0;
        packet->st.don = 1;                             /*  packetのステータスのdoneビットを     */
        return;                                         /*  セットしてリターン  */
}
strsect = packet->prm.rw.start_sec;                     /*  ｽﾀｰﾄ･ｾｸﾀ(論理ｾｸﾀ)packetから取り出す */
tr_sect = bpbmx[packet->unit_num].track_sec;            /*  BPBから'セクタ数／トラック'を取り出す        */
st_slnd = strsect / (tr_sect * 2);                      /*  ｽﾀｰﾄ･ｼﾘﾝﾀﾞをｽﾀｰﾄ･ｾｸﾀから求める        */
st_sect = (strsect % tr_sect) + 1;                      /*ｽﾀｰﾄ･ｾｸﾀ(物理ｾｸﾀ)をｽﾀｰﾄ･ｾｸﾀ(論理ｾｸﾀ)から求める*/
hed_num = (strsect / tr_sect) % 2;                      /*  ヘッド番号をｽﾀｰﾄ･ｾｸﾀ(論理ｾｸﾀ)から求める      */
sec_coun = packet->prm.rw.sect_coun;                    /*  転送セクタ数をpacketから取り出す     */
wv_sect = 0;                                            /*  書き込み済みセクタ数カウンタをクリア         */

do
{
        com_code = drvno;
        com_prm.c = st_slnd;
        ret_st = SEEK(com_code, &com_prm);              /*  書き込むシリンダへシークする         */
        if(ret_st != 0x00)                              /*  シークが異常終了のとき       */
        {
                ERRSET(ret_st, &ret_prm, 0, 0);         /*  ｴﾗｰ･ｾｯﾄ･ﾓｼﾞｭｰﾙを呼び出す     */
                packet->prm.rw.sect_coun = wv_sect;
                packet->st.don = 1;                     /*  packetのdoneビットに１をセットしてリターン  */
                return;
        }

                /*********************************/
                /*  次シリンダへのシークの判別  */
                /*********************************/

        if(hed_num == 0)                                /*  ヘッド番号が０のとき         */
        {
                if((sec_coun + st_sect - 1) > (tr_sect * 2))
                                                        /*  転送ｾｸﾀ数＞ｽﾀｰﾄ･ｼﾘﾝﾀﾞの残りｾｸﾀ数のとき  */
                {
                        seek_on = 1;                    /*  次シリンダへシークする       */
                }
                else                                    /*  転送ｾｸﾀ数≦ｽﾀｰﾄ･ｼﾘﾝﾀﾞの残りｾｸﾀ数のとき       */
                {
                        seek_on = 0;                    /*  次シリンダへのシークは行わない       */
                }
        }
        else                                            /*  ヘッド番号が１のとき         */
        {
                if((sec_coun + st_sect - 1) > tr_sect)  /*  転送ｾｸﾀ数＞ｽﾀｰﾄ･ﾄﾗｯｸの残りｾｸﾀ数のとき       */
                {
                        seek_on = 1;                    /*  次シリンダへシークする       */
                }
                else                                    /*  転送ｾｸﾀ数≦ｽﾀｰﾄ･ﾄﾗｯｸの残りｾｸﾀ数のとき       */
                {
                        seek_on = 0;                    /*  次シリンダへのシークは行わない       */
                }
        }
        if(seek_on == 1)                                /*  シークを行う場合の読み込みｾｸﾀ数のセット     */
        {
                if(hed_num == 0)                        /*  開始セクタのヘッド番号が０のとき     */
                {
                        set_sect = tr_sect * 2 - st_sect + 1;
                                                        /*  1ｼﾘﾝﾀﾞのｾｸﾀ数－ｽﾀｰﾄ･ｾｸﾀ(物理ｾｸﾀ)＋1   */
                }
                else                                    /*  開始セクタのヘッド番号が１のとき     */
                {
                        set_sect = tr_sect - st_sect + 1;       /*  1ﾄﾗｯｸのｾｸﾀ数－ｽﾀｰﾄ･ｾｸﾀ(物理ｾｸﾀ)＋1   */
                }
        }
```

保守／廃止

(DEVICE.C) p.13

```
            else                                                    /*  シークを行わない場合の読み込みｾｸﾀ数のｾｯﾄ  */
            {
                    set_sect = (unsigned char)sec_coun;             /*  転送セクタ数をセット        */
            }

            com_code = 0xd0 | drvno;                                /*  ライト・データのパラメータをセットする  */
            com_prm.c = st_slnd;                                    /*  開始シリンダ番号    */
            com_prm.h = hed_num;                                    /*  ヘッド番号  */
            com_prm.r = st_sect;                                    /*  開始セクタ番号(物理ｾｸﾀ)    */
            com_prm.n = 2;                                          /*  セクタあたりのバイト数      */
            com_prm.gsl = 0x1b;                                     /*  GAP3の読み飛ばしバイト数    */

            com_prm.dt_off = (unsigned int)packet->prm.rw.trns_off + wv_sect * 512;
                                                                    /*  転送アドレス(ｵﾌｾｯﾄ) */
            com_prm.trsize.dtsize = set_sect * 512;                 /*  転送バイト数        */

            com_prm.eot = tr_sect;                                  /*  トラックあたりのセクタ数    */

            com_prm.dt_seg = (unsigned int)packet->prm.rw.trns_seg;
                                                                    /*  転送アドレス(ｾｸﾞﾒﾝﾄ)       */

            ret_st = WRDAT(com_code, &com_prm, &ret_prm);
                                                                    /*  ライト・データ実行  */
            if(ret_st != 0x00)                                      /*  ライト・データが異常終了のとき      */
            {
                    ERRSET(ret_st, &ret_prm, 2, tr_sect);           /*  ｴﾗｰ･ｾｯﾄ･ﾓｼﾞｭｰﾙを呼び出す    */
                    packet->st.don = 1;                             /*  packetのdoneビットに１をｾｯﾄしてﾘﾀｰﾝ  */
                    return;
            }
            com_code = 0xe4 | drvno;                                /*  ベリファイのパラメータをセット      */
            com_prm.dtl = set_sect;                                 /*  ベリファイを行うセクタ数    */

            ret_st = VERIFY(com_code, &com_prm, &ret_prm);
                                                                    /*  ベリファイ実行      */
            if(ret_st != 0x00)                                      /*  ベリファイが異常終了のとき  */
            {
                    ERRSET(ret_st, &ret_prm, 1, tr_sect);           /*  ｴﾗｰ･ｾｯﾄ･ﾓｼﾞｭｰﾙを呼び出す    */
                    packet->st.don = 1;                             /*  packetのdoneビットに１をｾｯﾄしてﾘﾀｰﾝ  */
                    return;
            }
            wv_sect += set_sect;                                    /*  書き込み済みカウンタを更新する      */
            st_sect = 1;                                            /*  開始ｾｸﾀ番号を１にセット     */
            hed_num = 0;                                            /*  ヘッド番号を０にセット      */
            ++st_slnd;                                              /*  シリンダ番号をインクリメント        */
            sec_coun -= set_sect;                                   /*  書き込み未終了セクタを更新する      */
    }
    while(seek_on != 0);                                            /*  シークを行わなくなるまでライトを繰り返す    */

    packet->prm.rw.sect_coun = wv_sect;                             /*  packetに転送終了セクタ数をセット    */
    packet->st.don = 1;                                             /*  packetのdoneビットに１をセットする  */
    outportb(DOR70, 0x0c);                                          /*  モータの停止        */
    time_i = 0;                                                     /*  タイマ割り込みフラグのクリア        */
    return;
}


            /*****************************************************************/
            /*                                                               */
            /*  ＩＯＣＴＬ　インプット処理          ＩＮ　：pro_seg,d_exist */
            /*                                      ＯＵＴ：packet           */
            /*                                                               */
            /*****************************************************************/


void IOCTL_IN(void)
{
    unsigned char far *ioctl;
    ioctl = MK_FP(packet->prm.rw.trns_seg, packet->prm.rw.trns_off);
                                                                    /*  アドレスにｆａｒポインタをセット    */
    *ioctl = d_exist;
```

(DEVICE.C) p.14

```
	packet->prm.rw.sect_coun = 1;                                   /*  転送バイト数をpacketにセットする    */
	packet->st.don = 1;                                             /*  packetのdoneビットに１をセットする  */
	return;
}

	/*************************************************************/
	/*                                                           */
	/*  ユニット番号処理     ＩＮ   : packet,d_exist,unit_ex      */
	/*                       ＯＵＴ : packet,drvno                */
	/*                                                           */
	/*************************************************************/

unsigned char UNIT_NO(void)
{
	unsigned char drvno, mdrvno, sdrvno, i;

	if(packet->unit_num > unit_ex)                                  /*  packetのユニット番号が接続されている        */
	{                                                               /*  ドライブ数より大きいとき    */
		packet->st.err =1;                                      /*  packetのエラー・ビットに１をセットする  */
		packet->st.err_cd = 1;                                  /*  packetのエラー・コードに'   */
		return 4;                                               /*  無効なユニット番号'をセットする     */
	}
	drvno = packet->unit_num;                                       /*  デバイスの物理ドライブ番号に        */
                                                                        /*  ユニット番号をセットする    */
	sdrvno = 0;
	mdrvno = 0;
	i = 0;
	do
	{
		if(((d_exist >> i) & 0x01) == 0x00)                     /*  接続されていないﾄﾞﾗｲﾌﾞ番号の判定    */
		{
			++sdrvno;                                       /*  接続されていないﾄﾞﾗｲﾌﾞをインクリメント      */
		}
		else
		{
			++mdrvno;                                       /*  接続されているﾄﾞﾗｲﾌﾞ数をインクリメント      */
		}
		++i;
	}
	while((drvno + 1) > mdrvno);                                    /*  選択されたﾄﾞﾗｲﾌﾞ数がｶｳﾝﾄされるまで繰り返し  */
	drvno += sdrvno;                                                /*接続されていなかったﾄﾞﾗｲﾌﾞ数をﾄﾞﾗｲﾌﾞ番号に加算*/
	return drvno;
}



	/*************************************************************************/
	/*                                                                       */
	/*  エラー・セット処理          ＩＮ   : packet,ret_st,ex_sw,tr_sect     */
	/*                              ＯＵＴ : packet                          */
	/*                                                                       */
	/*************************************************************************/

void ERRSET(unsigned char ret_st, struct retpra *ret_prm,
	unsigned char ex_sw, unsigned char tr_sect)
{
	unsigned int ex_sect;


	packet->st.err = 1;                                             /*  packetのerrビットに１をセットする   */
	switch(ret_st)
	{
		case 0x01:                                              /*  ret_st＝01Hのとき   */
			packet->st.err_cd = 0x02;                       /*  ｴﾗｰ･ｺｰﾄﾞに'ﾄﾞﾗｲﾌﾞのﾉｯﾄ･ﾚﾃﾞｨ'をセットする    */
			break;
		case 0x02:                                              /*  ret_st＝02Hのとき   */
			packet->st.err_cd = 0x0c;                       /*  ｴﾗｰ･ｺｰﾄﾞに'一般的なエラー'をセットする      */
			break;
		case 0x03:                                              /*  ret_st＝03H-05Hのとき       */
		case 0x04:
```

保守／廃止

(DEVICE.C) p.15

```
            case 0x05:
                    packet->st.err_cd = 0x08;                   /*  ｴﾗｰｺｰﾄﾞに'セクタが存在しない'をセットする  */
                    break;
            case 0x06:                                          /*  ret_st＝06H,07Hのとき       */
            case 0x07:
                    packet->st.err_cd = 0x04;                   /*  ｴﾗｰｺｰﾄﾞに'ＣＲＣエラー'をセットする       */
                    break;
            case 0x08:                                          /*  ret_st＝08H,09Hのとき       */
            case 0x09:
                    packet->st.err_cd = 0x0b;                   /*  ｴﾗｰｺｰﾄﾞに'読み出しエラー'をセットする     */
                    break;
            case 0x0a:                                          /*  ret_st＝0AH,0BHのとき       */
            case 0x0b:
                    if(ex_sw == 1)                              /*  ex_sw＝1(インプット処理)のとき      */
                    {
                            packet->st.err_cd = 0x0b;           /*  ｴﾗｰｺｰﾄﾞに'読み出しエラー'をセットする     */
                    }
                    else                                        /*  ex_sw≠1(アウトプット処理)のとき    */
                    {
                            packet->st.err_cd = 0x0a;           /*  ｴﾗｰｺｰﾄﾞに'書き込みエラー'をセットする     */
                    }
                    break;
            case 0x0c:                                          /*  ret_st＝0CHのとき   */
                    packet->st.err_cd = 0x00;                   /*  ｴﾗｰｺｰﾄﾞに'ライト・プロテクト'をセットする */
                    break;
            case 0x0d:                                          /*  ret_st＝0DHのとき   */
                    packet->st.err_cd = 0x06;                   /*  ｴﾗｰｺｰﾄﾞに'シーク・エラー'をセットする     */
                    break;
            case 0x0e:                                          /*  ret_st＝0EHのとき   */
                    packet->st.err_cd = 0x0b;                   /*  ｴﾗｰｺｰﾄﾞに'読み出しエラー'をセットする     */
                    break;
            default:                                            /*  その他      */
                    packet->st.err_cd = 0x0c;                   /*  ｴﾗｰｺｰﾄﾞに'その他のエラー'をセットする     */
                    break;
    }
    if(ex_sw != 0)                                              /*  呼び出した処理がインプットまたは    */
                                                                /*  アウトプット処理のとき      */
    {
            ex_sect = (tr_sect * 2 * ret_prm->c) + ret_prm->h * tr_sect + ret_prm->r - 1;
                                                                /*  実行が終了した論理セクタ番号の算出  */

            packet->prm.rw.sect_coun = ex_sect - packet->prm.rw.start_sec;
                                                                /*  転送されたセクタ数をセット  */
    }

    outportb(DOR70, 0x0c);                                      /*  モータの停止        */
    time_i = 0;                                                 /*  タイマ割り込みフラグのクリア        */
    return;
}

            /*****************************************************/
            /*                                                   */
            /*      Ｉ／Ｏﾘｸｴｽﾄ･ｴﾗｰ処理     ＩＮ　：なし        */
            /*                              ＯＵＴ：packet       */
            /*                                                   */
            /*****************************************************/

void ERROR_IO(void)
{
    packet->st.err = 1;
    packet->st.err_cd = 0x03;
    return;
}
```

●FORMAT70.C

(FORMAT70.C) p.1

```
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>

#define         SELCT1          "[8;20H"                /*  ﾄﾞﾗｲﾌﾞ選択の初期位置(ｴｽｹｰﾌﾟ･ｼｰｹﾝｽ)  */
#define         SELCT2          "[10;20H"               /*  ﾒﾃﾞｨｱ選択の初期位置(ｴｽｹｰﾌﾟ･ｼｰｹﾝｽ)  */
#define         SYOKI1          8                       /*  ドライブ選択の初期位置(行)  */
#define         SYOKI2          10                      /*  メディア選択の初期位置(行)  */
#define         MESS            "[22;10H"               /*  ｴﾗｰなどのﾒｯｾｰｼﾞの表示位置(ｴｽｹｰﾌﾟ･ｼｰｹﾝｽ)*/
#define         EXECUTM         "[19;10H"               /*  実行中のﾒｯｾｰｼﾞ表示位置(ｴｽｹｰﾌﾟ･ｼｰｹﾝｽ)*/
#define         TRACKM          "[19;52H"               /*  フォーマット実行中トラックの          */
                                                        /*  表示位置(ｴｽｹｰﾌﾟ･ｼｰｹﾝｽ)      */
#define         HOMECL          "[2J"                   /*  画面のホーム・クリア(ｴｽｹｰﾌﾟ･ｼｰｹﾝｽ)  */
#define         DOR70           0x00d4                  /*  μPD72070のﾃﾞｨｼﾞﾀﾙ･ｱｳﾄﾌﾟｯﾄ･ﾚｼﾞｽﾀのIOｱﾄﾞﾚｽ*/
#define         INTVEC          0x0b                    /*  割り込みベクタ番号  */


struct compra
{                                                       /*  コマンド・パラメータの構造体  */
      union                                             /*  転送バイト数の共用体          */
      {
            unsigned int dtsz;                          /*  転送データのバイト数(int型)  */
            struct                                      /*  int型をchar型にわける構造体  */
            {
                  unsigned char kai;                    /*  下位８ビット分        */
                  unsigned char jyoui;                  /*  上位８ビット分        */
            } int_hen;

      } trsize;
      unsigned int dt_off;                              /*  転送ﾃﾞｰﾀのｵﾌｾｯﾄ･ｱﾄﾞﾚｽ        */
      unsigned int dt_seg;                              /*  転送ﾃﾞｰﾀのｾｸﾞﾒﾝﾄ･ｱﾄﾞﾚｽ       */
      unsigned char c;                                  /*  シリンダ番号          */
      unsigned char h;                                  /*  ヘッド番号  */
      unsigned char r;                                  /*  開始セクタ番号        */
      unsigned char n;                                  /*  １セクタのデータ長  */
      unsigned char dtl;                                /*  1ｾｸﾀあたりの処理ﾃﾞｰﾀ長       */
      unsigned char gsl;                                /*  GAP3の読み飛ばしバイト数      */
      unsigned char eot;                                /*  １トラックのセクタ数          */
      unsigned char d;                                  /*  ﾌｫｰﾏｯﾄの書きこみﾃﾞｰﾀ･ﾊﾟﾀｰﾝ  */
};

struct retpra
{                                                       /*  ﾘﾀｰﾝ･ｽﾃｰﾀｽ･ﾊﾟﾗﾒｰﾀの構造体    */
      unsigned char c;                                  /*  実行終了後のシリンダ番号      */
      unsigned char h;                                  /*  実行終了後のヘッド番号        */
      unsigned char r;                                  /*  実行終了後のセクタ番号        */
      unsigned char n;                                  /*  １セクタのデータ長  */
};

struct sr                                               /*  シーク・リザルトの構造体      */
{
      unsigned char dr_st;                              /*  シーク・コマンド実行後のﾘｻﾞﾙﾄ･ｽﾃｰﾀｽ */
      unsigned char dr_pcn;                             /*  シーク・コマンド終了後のヘッドの位置*/
};

struct form                                             /*  ﾌｫｰﾏｯﾄ時のID部の書き込みﾃﾞｰﾀの構造体*/
{
      unsigned char c;                                  /*  シリンダ番号          */
      unsigned char h;                                  /*  ヘッド番号  */
      unsigned char r;                                  /*  セクタ番号  */
      unsigned char n;                                  /*  セクタあたりのフォーマット・バイト数  */
};

struct fpnt                                             /*  ｆａｒﾎﾟｲﾝﾀをｾｸﾞﾒﾝﾄとｵﾌｾｯﾄに          */
                                                        /*  分割する構造体        */
{
      unsigned int adoff;
      unsigned int adseg;
```

保守／廃止

(FORMAT70.C) p.2

```
};

union fapnt                                                    /*  ｆａｒポインタの共用体宣言  */
{
      unsigned long pnt;                                       /*  ｆａｒポインタ      */
      struct fpnt pon;                                         /*  オフセットとセグメント      */
};


void exit(int);
void ERRPR(unsigned char);
void COMMAND(unsigned char *, char);
void interrupt INTRPT(void);
char RESULT(void);
void MEDIAT(unsigned char);
unsigned char RECAL(unsigned char);
unsigned char SEEK(unsigned char, struct compra *);
unsigned char RSEEK(unsigned char, struct compra *);
unsigned char TRFORMAT(unsigned char, struct compra *, struct retpra *);
unsigned char WRDAT(unsigned char, struct compra *, struct retpra *);
unsigned char VERIFY(unsigned char, struct compra *, struct retpra *);
unsigned getkey(void);                                         /*  キー・コード取得関数  */
void interrupt (*OLD_VEC)(void);

extern unsigned char int_s;                                    /*  割り込みの確認変数  */
extern unsigned char time_i;                                   /*  タイマ割り込みフラグ        */

            /*************************************************************************/
            /*              フォーマット・コマンド・メイン・ルーチン                 */
            /*************************************************************************/

void main(void)
{
      struct REGPACK reg;
      struct compra com_prm;
      struct retpra ret_prm;
      union fapnt vector;
      char aldrv, drv70, drvno, meda, sldcun, hedno, errcun, sccoun, rdcoun, i, d, ichi;
      char com_code, curdrv, sdrvno, mdrvno;
      char ret_st;
      int key_in;
      unsigned char adpnt[512], d_exist;
      unsigned char far *d_point;
      unsigned char rtd_dat;

            /*************************************/
            /*  ２ＤＤディスクの予約セクタ・データ  */
            /*************************************/

      unsigned char dd9_res[30] = { 0xeb, 0x1c, 0x90, 0x55, 0x50, 0x44, 0x37,
                                    0x32, 0x30, 0x37, 0x30, 0x00, 0x02, 0x02,
                                    0x01, 0x00, 0x02, 0x70, 0x00, 0xa0, 0x05,
                                    0xf9, 0x03, 0x00, 0x09, 0x00, 0x02, 0x00,
                                    0x00, 0x00 };

            /*************************************/
            /* ２ＨＤディスクの予約セクタ・データ  */
            /*************************************/

      unsigned char hd8_res[30] = { 0xeb, 0x1c, 0x90, 0x55, 0x50, 0x44, 0x37,
                                    0x32, 0x30, 0x37, 0x30, 0x00, 0x02, 0x01,
                                    0x01, 0x00, 0x02, 0xe0, 0x00, 0x40, 0x0b,
                                    0xf0, 0x09, 0x00, 0x12, 0x00, 0x02, 0x00,
                                    0x00, 0x00 };
```

保守／廃止

(FORMAT70.C) p.3

```
        /*****************************************/
        /* ２ＥＤディスクの予約セクタ・データ   */
        /*****************************************/

unsigned char ed4_res[30] = { 0xeb, 0x1c, 0x90, 0x55, 0x50, 0x44, 0x37,
                              0x32, 0x30, 0x37, 0x30, 0x00, 0x02, 0x02,
                              0x01, 0x00, 0x02, 0xf0, 0x00, 0x80, 0x16,
                              0xf0, 0x09, 0x00, 0x24, 0x00, 0x02, 0x00,
                              0x00, 0x00 };

        /*****************************************/
        /*  ２ＤＤディスクのＦＡＴセクタのデータ    */
        /*****************************************/

unsigned char dd9_fat[3] = { 0xf9, 0xff, 0xff};

        /*****************************************/
        /*  ２ＨＤディスクのＦＡＴセクタのデータ    */
        /*****************************************/

unsigned char hd8_fat[3] = { 0xf0, 0xff, 0xff};

        /*****************************************/
        /*  ２ＥＤディスクのＦＡＴセクタのデータ    */
        /*****************************************/

unsigned char ed4_fat[3] = { 0xf0, 0xff, 0xff};


int_s = 0x00;

d_exist = 0;
d_point = &d_exist;
OLD_VEC = getvect(INTVEC);

vector.pnt = (unsigned long)INTRPT;                          /*  割り込ﾓｼﾞｭｰﾙのｱﾄﾞﾚｽを共用体へｾｯﾄ*/

poke(0x0000, (unsigned int)(INTVEC * 4), vector.pon.adoff);
                                                             /*  割り込ﾓｼﾞｭｰﾙのｵﾌｾｯﾄ･ｱﾄﾞﾚｽをﾍﾞｸﾀにｾｯﾄ*/

poke(0x0000, (unsigned int)(INTVEC * 4 + 2), vector.pon.adseg);
                                                             /*  割り込ﾓｼﾞｭｰﾙのｾｸﾞﾒﾝﾄ･ｱﾄﾞﾚｽをﾍﾞｸﾀにｾｯﾄ*/

        /*****************************/
        /*  全接続ドライブ数の取得  */
        /*****************************/

reg.r_ax = 0x1900;                                           /*  カレント・ドライブの取得（保存）*/
intr(0x21, &reg);                                            /*  al = カレント・ドライブ*/
curdrv = (unsigned char)(reg.r_ax & 0x00ff);
do
{
        aldrv = (unsigned char)(reg.r_ax & 0x00ff);
        reg.r_dx = aldrv + 1;                                /*  カレント・ドライブを変更する*/
                                                             /*  （論理番号をインクリメント)*/
        reg.r_ax = 0x0e00;
        intr(0x21, &reg);
        reg.r_ax = 0x1900;                                   /*  カレント・ドライブの取得      */
        intr(0x21, &reg);                                    /*  al = カレント・ドライブ       */

}
while(aldrv != (unsigned char)(reg.r_ax & 0x00ff));
                                                             /*  ｶﾚﾝﾄ･ﾄﾞﾗｲﾌﾞが変わらなくなるまで繰り返す*/
reg.r_dx = curdrv;                                           /*  ｶﾚﾝﾄ･ﾄﾞﾗｲﾌﾞを元に戻す*/
reg.r_ax = 0x0e00;
intr(0x21, &reg);
++aldrv;                                                     /*  総論理ﾄﾞﾗｲﾌﾞ数＝最大論理ﾄﾞﾗｲﾌﾞ番号＋１*/
```

保守／廃止

(FORMAT70.C) p.4

```
        /*************************/
        /*     d_existの取得     */
        /*************************/

_AH = 0x44;
_AL = 0x04;
_BL = aldrv;
_CX = 0x0001;
_DS = FP_SEG(d_point);
_DX = FP_OFF(d_point);
geninterrupt(0x21);

        /*************************/
        /* ドライブの接続チェック */
        /*************************/

if(d_exist == 0x00)                                     /*  ドライブが無接続か？        */
{
        printf("ドライブがありません¥n");
        exit(0);
}
i = 0;
drv70 = 0;

while(i < 4)                                            /*  接続ドライブ数のチェック    */
{
        if(((d_exist >> i) & 0x01) == 0x01)
        {
                ++drv70;
        }
        ++i;
}
drvno = 0;
d = (char)aldrv - drv70;                                /*  ドライブ数のチェック */

        /*********************************/
        /*   コマンド・タイトルの表示    */
        /*********************************/

printf("¥x1b[>5h");                                     /*  カーソルの消去      */
printf("¥x1b%s", HOMECL);                               /*  画面のクリア        */

printf("¥x1b[3;8Hμ ＰＤ７２０７０用フォーマットコマンド");

i = 1;
while(i <= drv70)
{
        ichi = SYOKI1;
        ichi = ichi + (i - 1) * 2;
        printf("¥x1b[%d;20H", ichi);
        printf("%c：ドライブ", (0x40 + d + i));
        i++;
}
printf("¥x1b%s↑，↓キーで選択，リターンで決定", MESS);
printf("¥x1b%s", SELCT1);
printf("¥x1b[7m");                                      /*  文字の反転表示      */
printf("%c：ドライブ", (0x40 + d + drvno + 1));
printf("¥x1b%s", SELCT1);

        /************************************************/
        /*   入力キー・コードの取得およびdrvnoのセット  */
        /************************************************/

do
{
        key_in = getkey();                              /*  入力キー・コードの取得      */
        printf("¥x1b[%d;20H", ichi);                    /*  カーソルの移動      */
        printf("¥x1b[0m");                              /*  文字の正常表示      */
        printf("%c：ドライブ", (0x40 + d + drvno + 1));
        switch(key_in)
```

保守／廃止

(FORMAT70.C) p.5

```
        {
                case 0x3a00:                                    /*  入力キーが↑の場合          */
                        --drvno;
                        if(drvno < 0)
                        {
                                drvno = drv70 - 1;
                        }
                        break;
                case 0x3d00:                                    /*  入力キーが↓の場合          */
                        ++drvno;
                        if(drvno >= drv70)
                        {
                                drvno = 0;
                        }
                        break;
                default :                                       /*  その他のキー入力の場合      */
                        break;
        }
        ichi = SYOKI1 + drvno * 2;
        printf("¥x1b[%d;20H", ichi);                            /*  カーソルの移動      */
        printf("¥x1b[7m");                                      /*  文字の反転表示      */
        printf("%c：ドライブ", (0x40 + d + drvno + 1));
}
while(key_in != 0x1c0d);

        /*****************************/
        /*     選択ドライブの表示    */
        /*****************************/

printf("¥x1b[0m");                                              /*  文字の正常表示      */

printf("¥x1b%s選択ドライブ　%c：", SELCT1, (0x40 + d + drvno + 1));

        /*********************************************************/
        /*    実際に接続されているドライブとdrvnoをあわせる     */
        /*********************************************************/

i = 0;
sdrvno = 0;
mdrvno = 0;
do
{
        if(((d_exist >> i) & 0x01) == 0x00)                     /*  接続されていないﾄﾞﾗｲﾌﾞの判定        */
        {
                ++sdrvno;
        }
        else                                                    /*  接続されているﾄﾞﾗｲﾌﾞのカウント      */
        {
                ++mdrvno;
        }
        ++i;
}
while((drvno + 1) > mdrvno);                                    /*  選択されたﾄﾞﾗｲﾌﾞ数がｶｳﾝﾄされるまで繰り返し*/

drvno += sdrvno;                                                /*  接続されていなかったﾄﾞﾗｲﾌﾞ数だけ加算*/

        /***********************************/
        /*   メディアの選択画面の表示      */
        /***********************************/

meda = 0;
printf("¥x1b[%d;20H¥x1b[J", SYOKI2);                            /*  カーソル行のクリア  */
printf("¥x1b[7m");                                              /*  表示文字を反転表示にする    */
printf("¥x1b[%d;22H２ＤＤ", SYOKI2 );
printf("¥x1b[0m");                                              /*  表示文字を正常表示にする    */
printf("¥x1b[%d;22H２ＨＤ", SYOKI2 + 2);
printf("¥x1b[%d;22H２ＥＤ", SYOKI2 + 4);
printf("¥x1b%s↑，↓キーで選択，リターンで決定", MESS);
printf("¥x1b[%d;22H", SYOKI2);
```

保守／廃止

(FORMAT70.C) p.6

```
        /*********************************************************/
        /*     入力キー・コードの取得とmedaへメディアのセット    */
        /*********************************************************/

do
{
        key_in = getkey();                                      /*  入力キー・コードを取得     */
        switch(key_in)
        {
                case 0x3a00:                                    /*   入力キーが↑の場合       */
                        --meda;
                        if(meda < 0)
                        {
                                meda = 0x02;
                        }
                        break;
                case 0x3d00:                                    /*   入力キーが↓の場合       */
                        ++meda;
                        if(meda == 0x03)
                        {
                                meda = 0;
                        }
                        break;
                default :                                       /*  その他のキー入力の場合       */
                        break;
        }
        printf("¥x1b[%d;22H２ＤＤ", SYOKI2 );                  /*  すべての表示を正常表示にする          */
        printf("¥x1b[%d;22H２ＨＤ", SYOKI2 + 2);
        printf("¥x1b[%d;22H２ＥＤ", SYOKI2 + 4);
        printf("¥x1b[7m");                                      /*  表示文字を反転表示にする          */

        switch(meda)                                            /*  現在選択されているメディアを反転表示にする*/
        {
                case 0:
                        printf("¥x1b[%d;22H２ＤＤ", SYOKI2 );
                        break;
                case 1:
                        printf("¥x1b[%d;22H２ＨＤ", SYOKI2 + 2);
                        break;
                case 2:
                        printf("¥x1b[%d;22H２ＥＤ", SYOKI2 + 4);
                        break;
        }
        printf("¥x1b[0m");                                      /*  表示文字を正常表示にする     */
}
while(key_in != 0x1c0d);                                        /*  入力キーがリターンで決定      */

        /*****************************/
        /*   選択メディアの表示      */
        /*****************************/

printf("¥x1b%s¥x1b[J", SELCT2);
switch(meda)
{
        case 0x00:
                printf("¥x1b%s選択メディア　２ＤＤ", SELCT2);
                break;
        case 0x01:
                printf("¥x1b%s選択メディア　２ＨＤ", SELCT2);
                break;
        case 0x02:
                printf("¥x1b%s選択メディア　２ＥＤ", SELCT2);
                break;
}
com_code = meda;                                                /*  MEDIAT用のコマンド・コードをセット*/
MEDIAT(com_code);                                               /*  MEDIATモジュール呼び出し     */

com_code = 0x0c | drvno;                                        /*  リキャリブレイト用コマンド・コードをセット*/
ret_st = RECAL(com_code);                                       /*  リキャリブレイト実行          */
```

(FORMAT70.C) p.7

```
if(ret_st != 0x00)                                  /*  エラー発生か？      */
{
        ERRPR(ret_st);
        exit(0);
}

printf("¥x1b%s", EXECUTM);                          /*  実行メッセージの表示              */
printf("フォーマット中");

sldcun = 0;                                         /*   sldcunの初期化             */
hedno = 0;                                          /*   hednoの初期化              */

        /*********************************/
        /*     トラック・フォーマット     */
        /*********************************/

while(sldcun <= 79)
{
        if((hedno == 0) && (sldcun != 0))           /*  シークの判断        */
        {
                com_prm.c = sldcun;
                com_code = drvno;
                ret_st = SEEK(com_code, &com_prm);  /*  シーク実行  */

                if(ret_st != 0x00)                  /*  エラー発生か？      */
                {
                        ERRPR(ret_st);
                        exit(0);
                }
        }

        if(hedno == 0x00)                           /*  フォーマット中のトラックの表示      */
        {
                printf("¥x1b%s%3dトラック", TRACKM, (sldcun * 2));
        }
        else
        {
                printf("¥x1b%s%3dトラック", TRACKM, (sldcun * 2 + 1));
        }

        errcun = 0;                                 /*  エラー・カウンタの初期化      */
        while(errcun < 2)
        {
                if(hedno == 0x00)                   /*  フォーマットするサイドの判断 */
                {
                        com_code = 0x4c | drvno;    /*  サイド０の場合      */
                }
                else
                {
                        com_code = 0x6c | drvno;    /*  サイド１の場合      */
                }
                com_prm.c = sldcun;                 /*  シリンダ番号の設定  */
                com_prm.n = 0x02;                   /*  512/1ｾｸﾀに設定      */

                switch(meda)                        /*  1ﾄﾗｯｸあたりのｾｸﾀ数の設定   */
                {
                        case 0x00:                  /*  2DD = 9ｾｸﾀ/ﾄﾗｯｸ       */
                                com_prm.dtl = 9;
                                break;
                        case 0x01:                  /*  2HD = 18ｾｸﾀ/ﾄﾗｯｸ     */
                                com_prm.dtl = 18;
                                break;
                        case 0x02:                  /*  2ED = 36ｾｸﾀ/ﾄﾗｯｸ     */
                                com_prm.dtl = 36;
                                break;
                }

                com_prm.gsl = 84;
                com_prm.d = 0xe5;                   /* データ領域書き込みデータ      */
```

保守／廃止

(FORMAT70.C) p.8

```
                ret_st = TRFORMAT(com_code, &com_prm, &ret_prm);
                                                        /*  トラック・フォーマット実行  */

                if(ret_st != 0x00)                      /*  エラー発生？              */
                {
                        ERRPR(ret_st);
                        exit(0);
                }

                /************************************************/
                /*  フォーマットしたトラックをベリファイする    */
                /************************************************/

                com_code = 0x40 | drvno;
                com_prm.c = sldcun;                     /*  シリンダ番号のセット        */
                com_prm.h = hedno;                      /*  ヘッド番号のセット          */
                com_prm.r = 0x01;                       /*  開始セクタ番号のセット      */

                switch(meda)                            /*  最終セクタ番号の設定        */
                {
                        case 0x00:                      /*  2DD = 9ｾｸﾀ   */
                                com_prm.eot = 9;
                                break;
                        case 0x01:                      /*  2HD = 18ｾｸﾀ */
                                com_prm.eot = 18;
                                break;
                        case 0x02:                      /*  2ED = 36ｾｸﾀ  */
                                com_prm.eot = 36;
                                break;
                }
                com_prm.gsl = 27;                       /*  GAP3の読み飛ばしバイト数    */
                com_prm.dtl = 0xff;                     /*  規定値              */

                ret_st = VERIFY(com_code, &com_prm, &ret_prm);
                                                        /*  ベリファイ実行    */

                if(ret_st != 0x00)                      /*  エラー発生か？      */
                {
                        ++errcun;                       /*  Ｙｅｓ→ｴﾗｰ･ｶｳﾝﾀをｲﾝｸﾘﾒﾝﾄ   */
                }
                else
                {
                        break;                          /*  Ｎｏ→次の処理へ    */
                }
        }
        if(ret_st != 0x00)                              /*  エラー発生か？      */
        {
                ERRPR(ret_st);
                exit(0);
        }
        if(hedno == 0x00)                               /* フォーマットしたサイドの判断 */
        {
                hedno = 0x01;
        }
        else
        {
                hedno = 0x00;
                ++sldcun;                               /* シリンダ番号をインクリメント */
        }
}

        /************************/
        /*   リキャリブレイト   */
        /************************/

com_code = 0x0c | drvno;                                /*  リキャリブレイト用コマンド・コードをセット*/
ret_st = RECAL(com_code);

if(ret_st != 0x00)                                      /*  エラー発生か？      */
{
```

(FORMAT70.C) p.9

```
        ERRPR(ret_st);
        exit(0);
}

        /************************/
        /*  予約セクタのライト  */
        /************************/

errcun = 0;
while(errcun < 2)
{
        i = 0;
        while(i < 30)
        {
                switch(meda)                                    /*  メディア別に予約セクタのデータをセット*/
                {
                        case 0x00:                              /*  ２ＤＤ      */
                                adpnt[i] = dd9_res[i];
                                break;
                        case 0x01:                              /*  ２ＨＤ      */
                                adpnt[i] = hd8_res[i];
                                break;
                        case 0x02:                              /*  ２ＥＤ      */
                                adpnt[i] = ed4_res[i];
                                break;
                }
                ++i;
        }
        com_code = 0x4c | drvno;
        com_prm.trsize.dtsz = 30;                               /*  転送データサイズのセット    */
        com_prm.dt_off = FP_OFF(&adpnt[0]);                     /*  ｱﾄﾞﾚｽ･ﾎﾟｲﾝﾀのｵﾌｾｯﾄ･ｱﾄﾞﾚｽをセットする*/
        com_prm.dt_seg = FP_SEG(&adpnt[0]);                     /*  ｱﾄﾞﾚｽ･ﾎﾟｲﾝﾀのｾｸﾞﾒﾝﾄ･ｱﾄﾞﾚｽをセットする*/
        com_prm.c = 0x00;                                       /*  シリンダ番号をセット        */
        com_prm.h = 0x00;                                       /*  ヘッド番号をセット  */
        com_prm.r = 0x01;                                       /*  実行セクタ番号をセット      */

        ret_st = WRDAT(com_code, &com_prm, &ret_prm);
                                                                /*  ライト・データ実行  */

        if(ret_st != 0x00)                                      /*  エラー発生？        */
        {
                ERRPR(ret_st);
                exit(0);
        }

        /****************************/
        /*  予約セクタのベリファイ  */
        /****************************/

        com_code = 0x44 | drvno;
        com_prm.dtl = 0x01;
        ret_st = VERIFY(com_code, &com_prm, &ret_prm);
                                                                /*  ベリファイ実行      */

        if(ret_st != 0x00)                                      /*   エラー発生？       */
        {
                ++errcun;                                       /*  Ｙｅｓ→ｴﾗｰ･ｶｳﾝﾀをｲﾝｸﾘﾒﾝﾄ   */
        }
        else
        {
                break;                                          /*  Ｎｏ→次の処理へ    */
        }
}
if(ret_st != 0x00)                                              /*   エラー発生？       */
{
        ERRPR(ret_st);
        exit(0);
}
```

(FORMAT70.C) p.10

```
        /*****************************/
        /*    ＦＡＴセクタのライト  */
        /*****************************/

errcun = 0;
while(errcun < 2)
{
        i = 0;
        while(i < 3)
        {
                switch(meda)                                    /*  メディア別にFATセクタのデータをセット*/
                {
                        case 0x00:                              /*  ２ＤＤ      */
                                adpnt[i] = dd9_fat[i];
                                break;
                        case 0x01:                              /*  ２ＨＤ      */
                                adpnt[i] = hd8_fat[i];
                                break;
                        case 0x02:                              /*  ２ＥＤ      */
                                adpnt[i] = ed4_fat[i];
                                break;
                }
                ++i;
        }
        com_code = 0x4c | drvno;
        com_prm.trsize.dtsz = 3;                                /*  転送データサイズのセット    */
        com_prm.h = 0;
        com_prm.r = 0x02;                                       /*  実行セクタ番号をセット      */

        ret_st = WRDAT(com_code, &com_prm, &ret_prm);
                                                                /*  ライト・データ実行  */

        if(ret_st != 0x00)                                      /*   エラー発生？       */
        {
                ERRPR(ret_st);
                exit(0);
        }
        switch(meda)                                            /*  メディア別にFATセクタ(2)の実行ｾｸﾀをセット*/
        {
                case 0x00:                                      /*  ２ＤＤ      */
                        com_prm.r = 0x05;
                        break;
                case 0x01:                                      /*  ２ＨＤ      */
                case 0x02:                                      /*  ２ＥＤ      */
                        com_prm.r = 0x0b;
                        break;
        }
        com_prm.trsize.dtsz = 3;
        ret_st = WRDAT(com_code, &com_prm, &ret_prm);
                                                                /*  ライト・データ実行  */

        if(ret_st != 0x00)                                      /*  エラー発生？        */
        {
                ERRPR(ret_st);
                exit(0);
        }

        /*****************************************************************/
        /*   ＦＡＴセクタの２セクタ目以降の書き込み(ﾃﾞｰﾀはすべて00H)    */
        /*****************************************************************/

        adpnt[0] = 0x00;                                        /*  転送データ(1ﾊﾞｲﾄ)をメモリにセット   */
        sccoun = 0;                                             /*  セクタ・カウンタをクリア    */
        switch(meda)
        {
                case 0x00:                                      /*  ２ＤＤ      */
                        while(sccoun < 2)
                        {
                                com_prm.r = 3 + sccoun;
                                com_prm.trsize.dtsz = 1;        /*  転送データ・サイズをセット  */
```

(FORMAT70.C) p.11

```
                        ret_st = WRDAT(com_code, &com_prm, &ret_prm);
                                                /*  ライト・データ実行  */

                        if(ret_st != 0x00)      /*  エラー発生？        */
                        {
                                ERRPR(ret_st);
                                exit(0);
                        }
                        com_prm.r = 6 + sccoun;
                        com_prm.trsize.dtsz = 1;        /*  転送データ・サイズをセット  */

                        ret_st = WRDAT(com_code, &com_prm, &ret_prm);
                                                /*  ライト・データ実行  */

                        if(ret_st != 0x00)      /*  エラー発生？        */
                        {
                                ERRPR(ret_st);
                                exit(0);
                        }
                        ++sccoun;
                }
                break;

        case 0x01:                              /*  ２ＨＤ(18ｾｸﾀ)      */
        case 0x02:                              /*  ２ＥＤ      */
                while(sccoun < 8)
                {
                        com_prm.r = 3 + sccoun;
                        com_prm.trsize.dtsz = 1;        /*  転送データ・サイズをセット  */

                        ret_st = WRDAT(com_code, &com_prm, &ret_prm);
                                                /*  ライト・データ実行  */

                        if(ret_st != 0x00)      /*  エラー発生？        */
                        {
                                ERRPR(ret_st);
                                exit(0);
                        }
                        com_prm.r = 12 + sccoun;
                        com_prm.trsize.dtsz = 1;        /*  転送データ・サイズをセット  */

                        if((meda == 0x01) && (com_prm.r == 19))
                                                /*  ２ＨＤの最終ＦＡＴセクタのとき      */
                        {
                                com_prm.h = 1;          /*  ヘッドをサイド１にセット    */
                                com_prm.r = 1;          /*  開始セクタを１セクタにセット        */
                        }
                        ret_st = WRDAT(com_code, &com_prm, &ret_prm);
                                                /*  ライト・データ実行  */

                        if(ret_st != 0x00)      /*  エラー発生？        */
                        {
                                ERRPR(ret_st);
                                exit(0);
                        }
                        ++sccoun;
                }
                break;
}


/*********************************/
/*   ＦＡＴセクタのベリファイ   */
/*********************************/

com_code = 0xc4 | drvno;
com_prm.r = 0x02;
com_prm.h = 0;
switch(meda)
```

(FORMAT70.C) p.12

```
        {
                case 0x00:                                      /*  ２ＤＤディスクのＦＡＴセクタ数のセット*/
                        com_prm.dtl = 6;
                        break;
                case 0x01:
                case 0x02:                                      /*  ２ＨＤ(18ｾｸﾀ), ２ＥＤディスクのＦＡＴ*/
                                                                /*  セクタ数のセット        */
                        com_prm.dtl = 18;
                        break;
        }
        ret_st = VERIFY(com_code, &com_prm, &ret_prm);
                                                                /*  ベリファイの実行    */

        if(ret_st != 0x00)                                      /*  エラーの発生？      */
        {
                ++errcun;                                       /*  Ｙｅｓ→ｴﾗｰ･ｶｳﾝﾀをｲﾝｸﾘﾒﾝﾄ   */
        }
        else
        {
                break;                                          /*   Ｎｏ→次の処理へ   */
        }
}
if(ret_st != 0x00)                                              /*   エラー発生？       */
{
        ERRPR(ret_st);
        exit(0);
}

        /************************************/
        /*    ルート・ディレクトリのライト  */
        /************************************/

errcun = 0;
while(errcun < 2)
{
        int k;
        k = 0;
        while(k < 512)                                          /*  ﾙｰﾄ･ﾃﾞｨﾚｸﾄﾘのデータをメモリにセット*/
        {
                if((k % 32) == 0)                               /*  32ビットごとに00Hのデータが入る     */
                {
                        rtd_dat = 0x00;
                }
                else
                {
                        rtd_dat = 0xe5;                         /*  そのほかのデータはE5H       */
                }
                adpnt[k] = rtd_dat;
                ++k;
        }

        com_code = 0xcc | drvno;

        switch(meda)                                            /*  各ﾒﾃﾞｨｱごとにﾙｰﾄ･ﾃﾞｨﾚｸﾄﾘのｾｸﾀ数をｾｯﾄ*/
        {
                case 0x00:                                      /*  ２ＤＤﾃﾞｨｽｸのﾙｰﾄ･ﾃﾞｨﾚｸﾄﾘのｾｸﾀ数のｾｯﾄ*/
                        rdcoun = 7;
                        break;
                case 0x01:                                      /*  ２ＨＤ(18ｾｸﾀ)ディスクのﾙｰﾄ･ﾃﾞｨﾚｸﾄﾘの*/
                                                                /*  セクタ数のセット        */
                        rdcoun = 14;
                        break;
                case 0x02:                                      /*  ２ＥＤディスクのﾙｰﾄ･ﾃﾞｨﾚｸﾄﾘの      */
                                                                /*  セクタ数のセット        */
                        rdcoun = 15;
                        break;
        }

        sccoun = 0;
        while(sccoun < rdcoun)                                  /*   ﾙｰﾄ･ﾃﾞｨﾚｸﾄﾘのセクタ数分のくり返し  */
```

(FORMAT70.C) p.13

```
        {
                com_prm.h = 0x01;
                switch(meda)                                    /*  各メディアのパラメータ設定      */
                {
                        case 0x00:                              /*  ２ＤＤディスクのパラメータ設定      */
                                if(sccoun < 2)                  /*  ﾙｰﾄ･ﾃﾞｨﾚｸﾄﾘの1,2ｾｸﾀ目(ｻｲﾄﾞ0)の設定  */
                                {
                                        com_prm.h = 0x00;
                                        com_prm.r = sccoun + 8;
                                }
                                else                            /*  ﾙｰﾄ･ﾃﾞｨﾚｸﾄﾘの3ｾｸﾀ目以降(ｻｲﾄﾞ1)の設定*/
                                {
                                        com_prm.r = sccoun - 1;
                                }
                                break;
                        case 0x01:                              /*  ２ＨＤディスクのパラメータ設定      */
                                com_prm.r = sccoun + 2;
                                break;
                        case 0x02:                              /*  ２ＥＤディスクのパラメータ設定      */
                                com_prm.h = 0x00;
                                com_prm.r = sccoun + 20;
                                break;
                }
                com_prm.trsize.dtsz = 512;                      /*  転送データバイト数のセット  */

                ret_st = WRDAT(com_code, &com_prm, &ret_prm);
                                                                /*  ライト・データ実行  */

                if(ret_st != 0x00)                              /*  エラー発生？        */
                {
                        ERRPR(ret_st);
                        exit(0);
                }
                ++sccoun;                                       /*  ｾｸﾀ･ｶｳﾝﾀのインクリメント    */
        }

        /****************************************/
        /*  ルート・ディレクトリのベリファイ    */
        /****************************************/

        com_code = 0xc4 | drvno;                                /*  ベリファイ用コマンド・コードをセット*/
        com_prm.h = 0x00;
        switch(meda)                                            /*  各メディアのパラメータ設定  */
        {
                case 0x00:                                      /*  ２ＤＤディスクのパラメータ設定      */
                        com_prm.dtl = 7;
                        com_prm.r = 8;
                        break;
                case 0x01:                                      /*  ２ＨＤディスクのパラメータ設定      */
                        com_prm.dtl = 14;
                        com_prm.h = 0x01;
                        com_prm.r = 2;
                        break;
                case 0x02:                                      /*  ２ＥＤディスクのパラメータ設定      */
                        com_prm.dtl = 15;
                        com_prm.r = 20;
                        break;
        }
        ret_st = VERIFY(com_code, &com_prm, &ret_prm);
                                                                /*  ベリファイ実行      */

        if(ret_st != 0x00)                                      /*  エラー発生？        */
        {
                ++errcun;                                       /*  Ｙｅｓ→ｴﾗｰ･ｶｳﾝﾀ,ｲﾝｸﾘﾒﾝﾄ    */
        }
        else
        {
                break;                                          /*  Ｎｏ→次の処理へ    */
        }
}
```

保守／廃止

(FORMAT70.C) p.14

```
	if(ret_st != 0x00)                                          /*  エラー発生？        */
	  {
		ERRPR(ret_st);
		exit(0);
	}

	outportb(DOR70, 0x0c);                                      /*  モータの停止        */
	time_i = 0;                                                 /*  タイマ割り込みフラグのクリア        */

	setvect(INTVEC, OLD_VEC);                                   /*  割り込みベクタを復帰する    */

	printf("¥x1b[>5l");                                        /*  カーソルを表示する  */
	printf("¥x1b%s", EXECUTM);                                 /*  カーソルをメッセージ表示位置へ      */
	printf("¥x1b[0J");
	printf("フォーマット終了¥n");                               /*  正常終了を表示する  */

}


		/*************************/
		/* キー・コード取得関数 */
		/*************************/

unsigned getkey(void)
{
	_AH = 0;
	geninterrupt(0x18);
	return(_AX);
}


		/************************************************/
		/*  エラー表示処理      ＩＮ    : ret_st        */
		/*                      ＯＵＴ  : なし          */
		/************************************************/

void ERRPR(unsigned char ret_st)
{

	printf("¥x1b[>5l");                                        /*  カーソルを表示する  */
	printf("¥x1b%s", MESS);
	printf("¥x1b[K");
	switch(ret_st)
	{
		case 0x01:
			printf("ドライブの準備ができていません。¥n");
			break;
		case 0x02:
		case 0x06:
		case 0x07:
		case 0x08:
		case 0x0a:
		case 0x0b:
			printf("データ・エラーです。¥n");
			break;
		case 0x03:
		case 0x04:
			printf("シーク・エラーです。¥n");
			break;
		case 0x05:
			printf("セクタが見つかりません。¥n");
			break;
		case 0x0c:
			printf("書き込み禁止です。¥n");
			break;
		defalut:
			printf("エラーです。¥n");
			break;
	}
	outportb(DOR70, 0x0c);                                      /*  モータの停止        */
```

(FORMAT70.C) p.15

```
    time_i = 0;                                    /*  タイマ割り込みフラグのクリア        */
    setvect(INTVEC, OLD_VEC);                      /*  割り込みベクタを復帰する    */
}
```

第４章　コンパイル／リンク方法

この章では，μPD72070用デバイス・ドライバとフォーマット・コマンドのコンパイルおよびリンク方法を説明します。

μPD72070用デバイス・ドライバとフォーマット・コマンドは，以下に示す８つのソース・ファイルをアセンブル・コンパイルしたあとリンクして作成します。

DEV70. ASM	３.２.３　デバイス・ドライバおよび３.３.３　デバイス・ドライバで説明した，アセンブリ言語で記述されたソース・ファイル
BIOS. C	３.２.１　BIOS部および３.３.１　BIOS部で説明したすべてのモジュールのC言語ソース・ファイル
LIO1. C	３.２.２　LIO部で説明したすべてのモジュールのC言語ソース・ファイル
DEVICE1. C	３.２.３　デバイス・ドライバで説明したC言語で記述されたモジュールのソース・ファイル
FORMAT71. C	３.２.４　フォーマット・コマンドで説明したすべてのモジュールのC言語ソース・ファイル
LIO. C	３.３.２　LIO部で説明したすべてのモジュールのC言語ソース・ファイル
DEVICE. C	３.３.３　デバイス・ドライバで説明したC言語で記述されたモジュールのソース・ファイル
FORMAT70. C	３.３.４　フォーマット・コマンドで説明したすべてのモジュールのC言語ソース・ファイル

プログラムのファイル構成を表４－１に示します。コンパイラ，アセンブラ，リンカは表４－２に示す製品を使用しました。

保守／廃止

表４－１　プログラムのファイル構成

プログラム名	ファイル名
1 M/2 MバイトFDD用 フォーマット・コマンド （FORMAT71. EXE）	BIOS. C[注], LIO1. C[注], FORMAT71. C
1 M/2 MバイトFDD用 デバイス・ドライバ （DEVICE71. SYS）	BIOS. C[注], LIO1. C[注], DEVICE1. C, DEV70. ASM
4 MバイトFDD用 フォーマット・コマンド （FORMAT71. EXE）	BIOS. C[注], LIO. C[注], FORMAT70. C
4 MバイトFDD用 デバイス・ドライバ （DEVICE71. SYS）	BIOS. C[注], LIO. C[注], DEVICE. C, DEV70. ASM

注　フォーマット・コマンドとデバイス・ドライバは，ともにBIOS. C, LIO1. CまたはLIO. C とリンクします。

表４－２　使用ツール

ツール	製品名
コンパイラ	TURBO　C＋＋　Ver. 1. 01　PC-9801用
リンカ	TURBO　LINK　Ver. 3. 0　PC-9801用
アセンブラ	TURBO　ASSEMBLER　Ver. 2. 0　PC-9801用

（a）フォーマット・コマンドのコンパイル方法

（ｉ）tcc -c -mt bios
（ii）tcc -c -mt lio1　または　tcc -c -mt lio
（iii）tcc -c -mt format71　または　tcc -c -mt format70
（iv）tlink /c /m c0t format71 lio1 bios, format71, , cs. lib
　　または
　　tlink /c /m c0t format70 lio bios, format70, , cs. lib

（b）デバイス・ドライバのコンパイル方法

（ｉ）tasm /ml dev70
（ii）tcc -c -mt bios
（iii）tcc -c -mt device1　または　tcc -c -mt device
（iv）tcc -S -mt lio1　または　tcc -S -mt lio
　　asmファイルが作成されます。このasmファイルをエディタによって図４- 1で示す２箇所を修正します。

保守／廃止

図４－１　修正箇所

```
;            vector. pnt = (unsigned long) INTRPT ;          ／＊ 割り込みモジュールのアドレス
                                                                を共用体にセット ＊／
;
      mov       word ptr [bp-14] , offset _INTRPT
      mov       word ptr [bp-12] , seg _INTRPT
                                                          このように修正します。
      mov       word ptr [bp-12] , cs
                        .
                        .
                        .
;                 vector. pnt = (unsigned long) TIME ;
;
      mov       word ptr [bp-6] , offset _TIME
      mov       word ptr [bp-4] , seg _TIME
                                                          このように修正します。
      mov       word ptr [bp-4] , cs
```

（ｖ）tasm /ml lio1　または　tasm /ml lio

（ｖｉ）tlink /c /m dev70 device1 lio1 bios, device71, , cs. lib

または

tlink /c /m dev70 device lio bios, device70, , cs. lib

exeファイルが作成されます。これをsys ファイルに変換します。

（ｖｉｉ）exe2bin device71. exe device71. sys

または

exe2bin device70.exe device70.sys

図４－１のように修正を行わないと，sysファイルに変換できません。

保守／廃止

第５章　CONFIG．SYSへの登録方法

この章ではμPD72070用デバイス・ドライバのMS-DOSへの登録方法を説明します。
次にCONFIG．SYSへの登録方法を示します。
CONFIG．SYSの内容が，図５－１のようになっているとします。

図５－１　CONFIG．SYSの内容１

```
FILES = 20
BUFFERS = 20
DEVICE = PRINT．SYS
DEVICE = RSDRV．SYS
```


図５－１をエディタを使用して，図５－２のようにμPD72070用デバイス・ドライバのファイル名を追加します。

図５－２　CONFIG．SYSの内容２

```
FILES = 20
BUFFERS = 20
DEVICE = PRINT．SYS
DEVICE = RSDRV．SYS
DEVICE = DEVICE71．SYS
または
DEVICE = DEVICE70．SYS
```

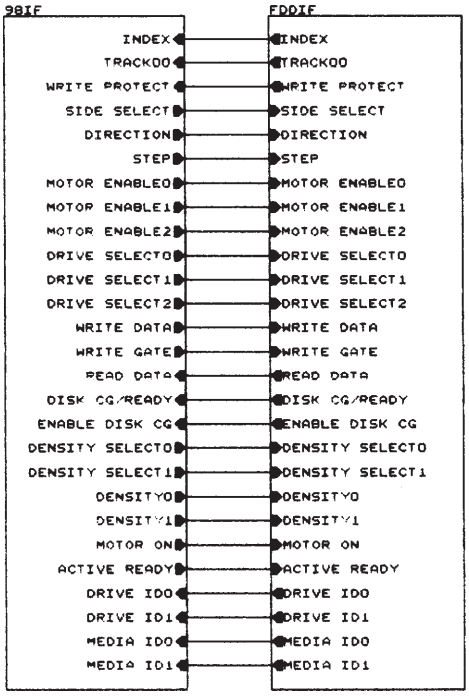
これにより，μPD72070用デバイス・ドライバがCONFIG．SYSに登録されます。
これでMS-DOSシステムを起動すれば，μPD72070用デバイス・ドライバがMS-DOSに組み込まれます。

注意　μPD72070用デバイス・ドライバは，必ずCONFIG．SYSの一番最後に登録してください。

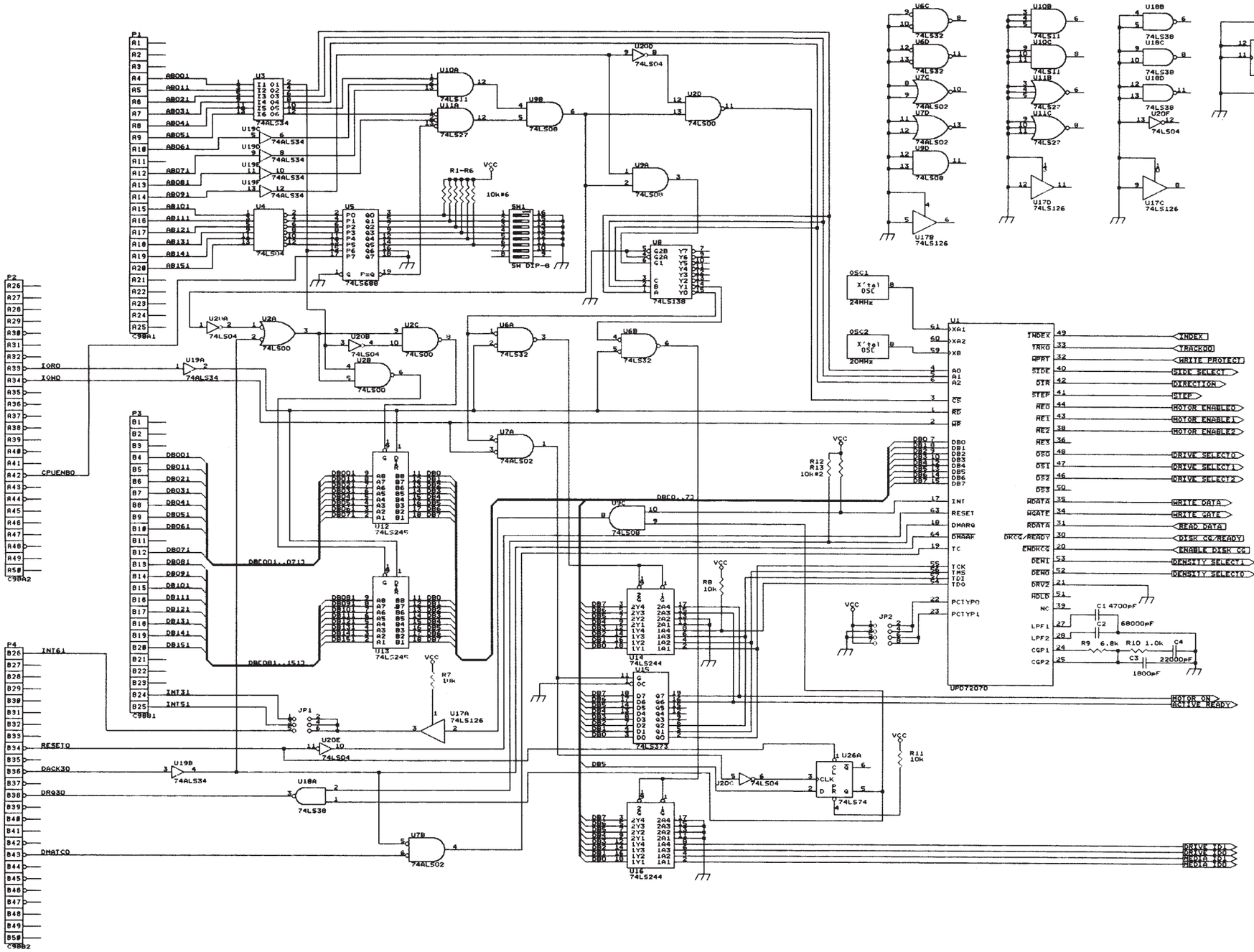
保守／廃止

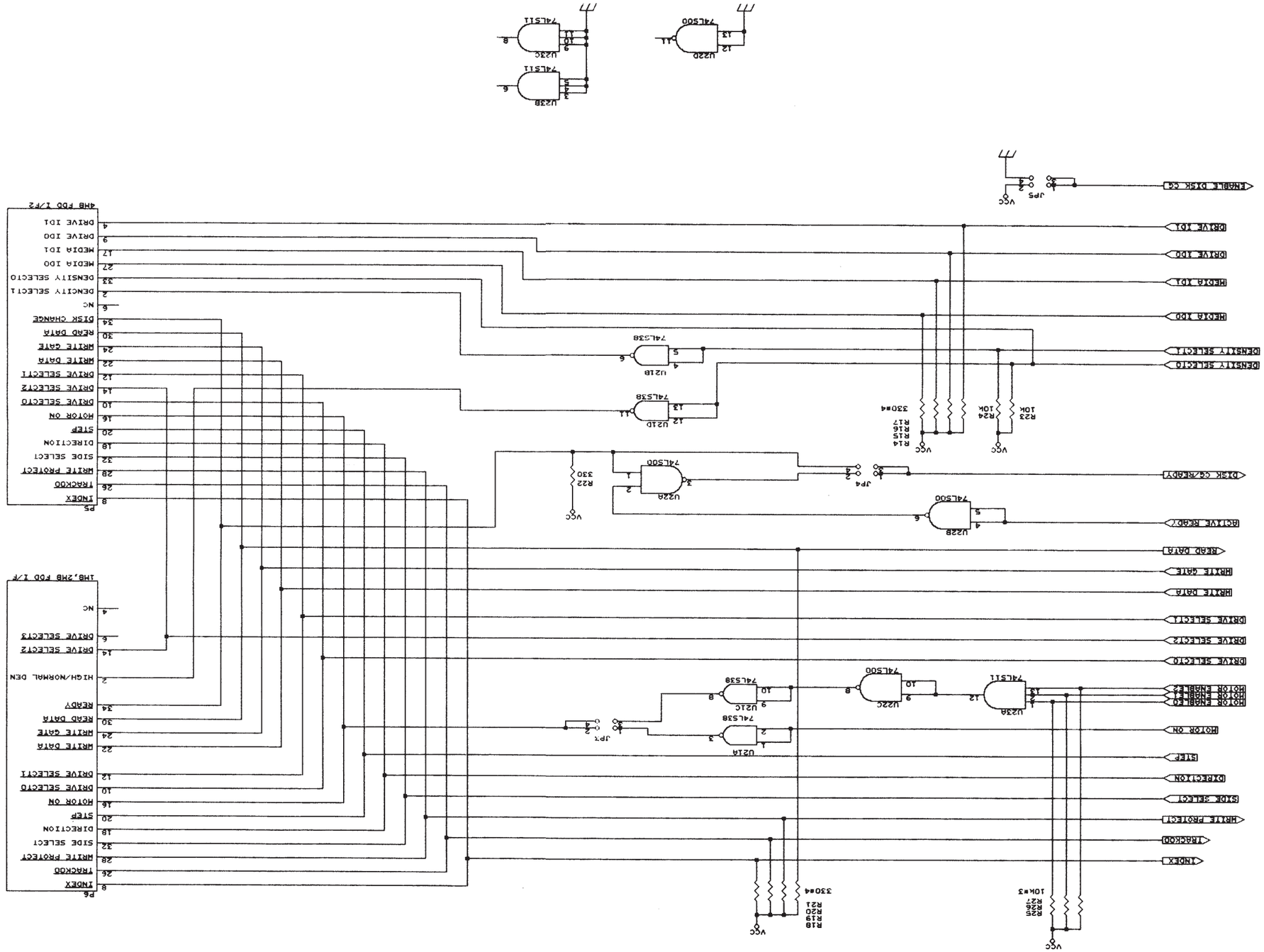
付録A　回 路 図

μPD72070ホスト・アダプタ・ボードの回路図を示します。


98IF
INDEX
TRACK00
WRITE PROTECT
SIDE SELECT
DIRECTION
STEP
MOTOR ENABLE0
MOTOR ENABLE1
MOTOR ENABLE2
DRIVE SELECT0
DRIVE SELECT1
DRIVE SELECT2
WRITE DATA
WRITE GATE
READ DATA
DISK CG/READY
ENABLE DISK CG
DENSITY SELECT0
DENSITY SELECT1
DENSITY0
DENSITY1
MOTOR ON
ACTIVE READY
DRIVE ID0
DRIVE ID1
MEDIA ID0
MEDIA ID1
FDDIF
INDEX
TRACK00
WRITE PROTECT
SIDE SELECT
DIRECTION
STEP
MOTOR ENABLE0
MOTOR ENABLE1
MOTOR ENABLE2
DRIVE SELECT0
DRIVE SELECT1
DRIVE SELECT2
WRITE DATA
WRITE GATE
READ DATA
DISK CG/READY
ENABLE DISK CG
DENSITY SELECT0
DENSITY SELECT1
DENSITY0
DENSITY1
MOTOR ON
ACTIVE READY
DRIVE ID0
DRIVE ID1
MEDIA ID0
MEDIA ID1

保守／廃止


P1
P2
P3
P4
74LS534
74LS14
74LS688
74LS138
74LS04
74LS00
74LS32
74LS08
74LS11
74LS27
74LS38
74LS02
74LS126
74LS245
74LS244
74LS373
74LS74
74LS30
74ALS34
74ALS02
SW DIP-8
24MHz
20MHz
UPD72070
INDEX
TRK0
WPRT
SIDE
DIR
STEP
MEO
ME1
ME2
ME3
DS0
DS1
DS2
DS3
WDATA
WGATE
RDATA
DRCC/READY
ENDRCC
DEN1
DEN0
DRV2
HDLD
NC
LPF1
LPF2
CGP1
CGP2
INT
RESET
DMARQ
DMAAK
TC
TCK
TMS
TDI
TDO
PCTYP0
PCTYP1
MOTOR ON
ACTIVE READY
DRIVE SELECT0
DRIVE SELECT1
DRIVE SELECT2
WRITE DATA
WRITE GATE
READ DATA
DISK CC/READY
ENABLE DISK CC
DENSITY SELECT1
DENSITY SELECT0
MOTOR ENABLE0
MOTOR ENABLE1
MOTOR ENABLE2
WRITE PROTECT
SIDE SELECT
DIRECTION
STEP
IORQ
IOWQ
CPUEMBQ
RESETQ
DACK3Q
DRQ3Q
DMATCQ
INT6J
INT3J
INT5J
R1-R6
10k*6
R7
1.0k
R8
10k
R11
10k
R12
R13
10k*2
C1 4700pF
C2
68000pF
R9 6.8k
R10 1.0k
C4
C3
22000pF
1800pF
JP1
JP2

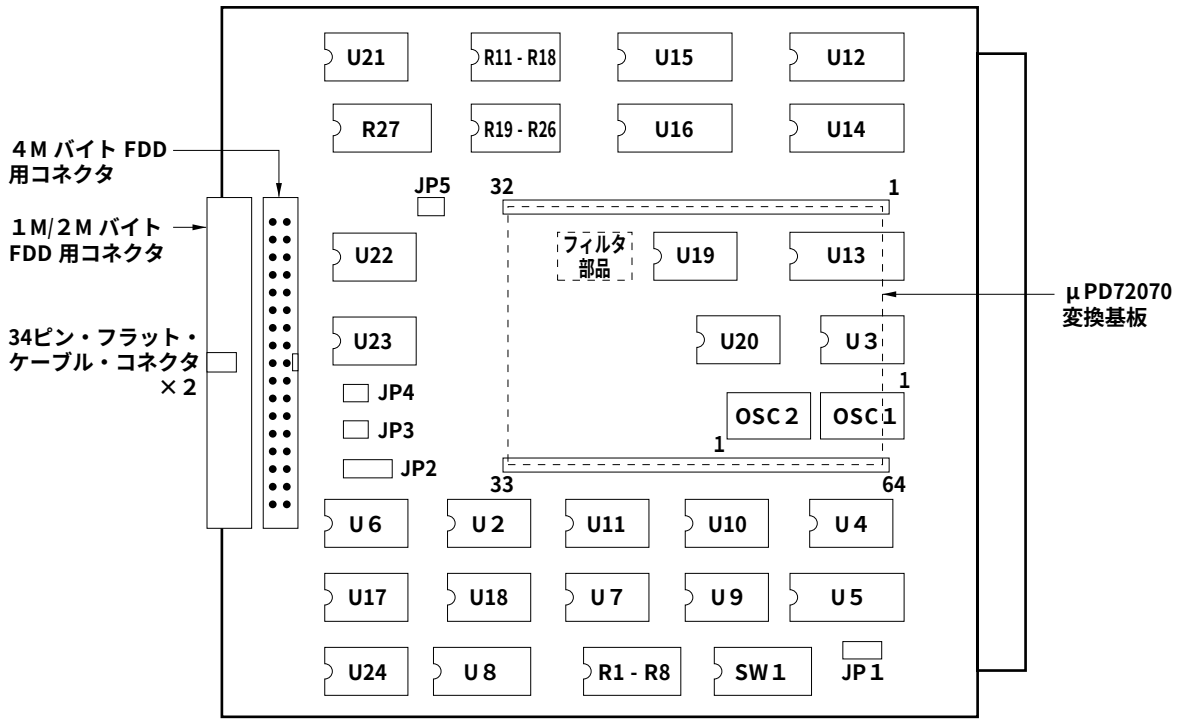

4MB FDD I/F2
DRIVE ID1
DRIVE ID0
MEDIA ID1
MEDIA ID0
DENSITY SELECT0
DENSITY SELECT1
NC
DISK CHANGE
READ DATA
WRITE GATE
WRITE DATA
DRIVE SELECT1
DRIVE SELECT2
DRIVE SELECT0
MOTOR ON
STEP
DIRECTION
SIDE SELECT
WRITE PROTECT
TRACK00
INDEX
P5
1MB,2MB FDD I/F
NC
DRIVE SELECT3
HIGH/NORMAL DEN
P6
ENABLE DISK CG
DISK CG/READY
ACTIVE READY
MOTOR ENABLE0
MOTOR ENABLE1
MOTOR ENABLE2
74LS38
74LS00
74LS11
330*4
10K*3
JP5
JP4
JP3
VCC

付録B　部品配置図

μPD72070ホスト・アダプタ・ボードの部品配置図を示します。

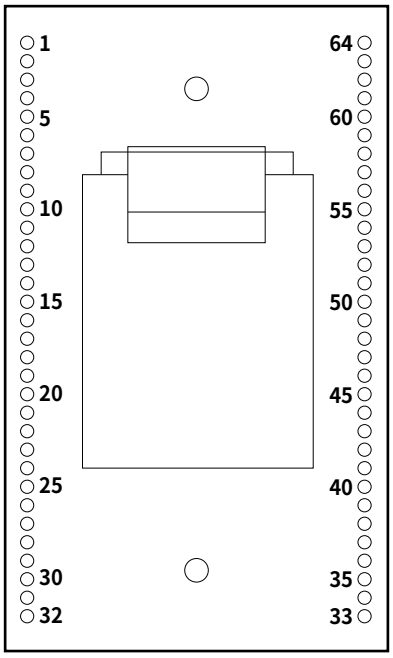
保守／廃止

μPD72070ホスト・アダプタ・ボード部品配置図



μPD72070ホスト・アダプタ・ボード部品配置図

QFP → DIP変換基板（μPD72070）



保守／廃止

付録C　SPDチャートの説明

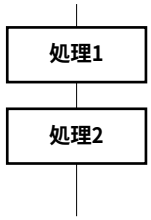
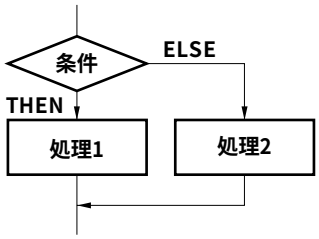
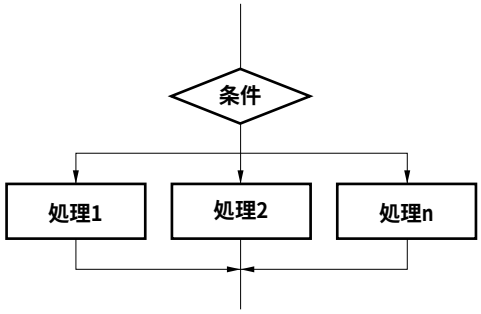
SPDとは，Structured Programming Diagramsの頭文字をとったもので，文字どおり訳せば「構造化プログラム図」と言えます。

構造化というのは，プログラムの論理処理の構造化のことで，論理の基本構造を用いて論理の設計，組み立てを行うことです。

すべてのプログラムは，論理の基本構造（順次，選択，繰り返し）の組み合わせのみで作成することができ（これを構造化定理といいます），構造化することでプログラムの流れが明確になり，信頼性が向上します。プログラムの構造化を表現するには，いろいろな方法がありますが，当社では，SPDという図式化技法を用いています。

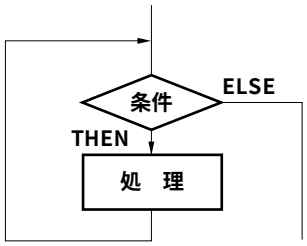
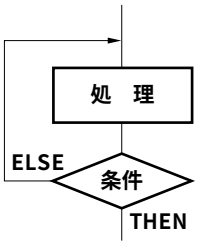
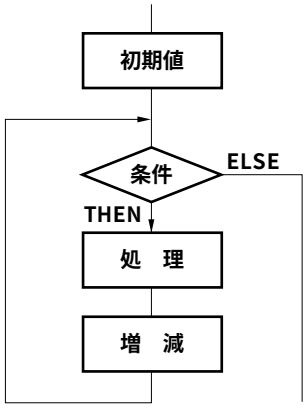
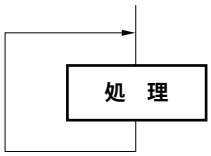
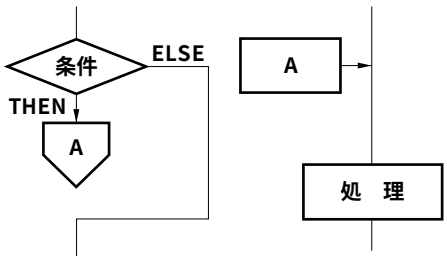
以下に，SPD技法で用いるSPD記号の説明，およびフロー・チャート記号との対比を示します。

表C－１　SPD記号とフロー・チャートの対比（1/2）

処理名称	SPD記号	フロー・チャート記号
順次処理	処理1 処理2	
条件分岐 （IF）	（IF：条件） [THEN] 処理1 [ELSE] 処理2	
条件分岐 （SWITCH）	（SWITCH：条件） [CASE：1] 処理1 [CASE：2] 処理2 ⋮ [CASE：n] 処理n	

保守／廃止

表C－１　SPD記号とフロー・チャートの対比（2/2）

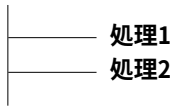
処理名称	SPD記号	フロー・チャート記号
条件ループ（WHILE）	（WHILE：条件） 処理	
条件ループ（UNTIL）	（UNTIL：条件） 処理	
条件ループ（FOR）	（FOR：初期値；条件；増減指定） 処理	
無限ループ	（LOOP） 処理	
結合子	（IF：条件） [THEN] GOTO A A 処理	

保守／廃止

C.1　順次処理

順次処理は，処理を上から下へ出現順に実行します。

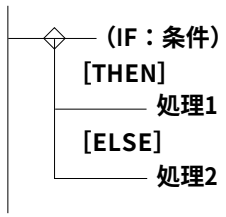
・SPDチャート



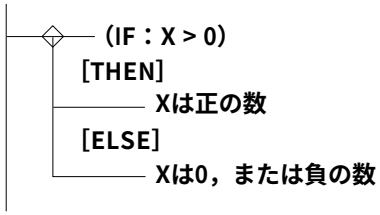
C.2　条件分岐：２分岐（IF）

IFに示した条件の真偽（THEN/ELSE）により処理内容を選択します。

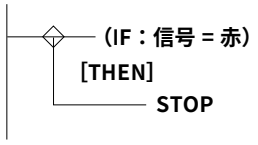
・SPDチャート



例１．Xの正負判別



２．信号が赤ならSTOPする



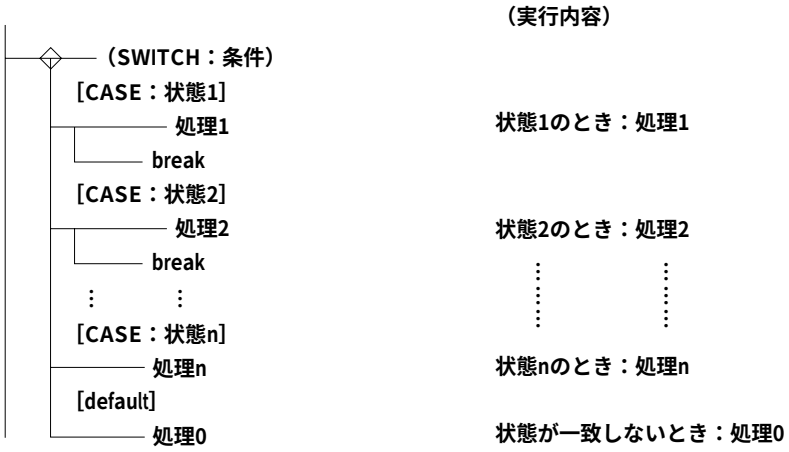
保守／廃止

C.3　条件分岐：多分岐（SWITCH）

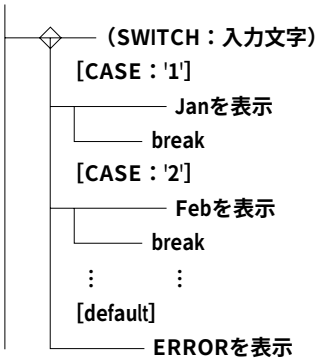
SWITCHに示した条件を，CASEで示された状態と比較し，処理を選択します。SWITCH文の処理は，一致した状態のみの処理を実行する場合と，一致した状態から下へ処理を続ける場合の二通りがあります（処理が下へ続かない場合は，'break' を記述）。また，一致した状態がない場合は，'default' の処理を実行します（'default'の記述は任意）。

（1）一致した状態のみの場合

・SPDチャート

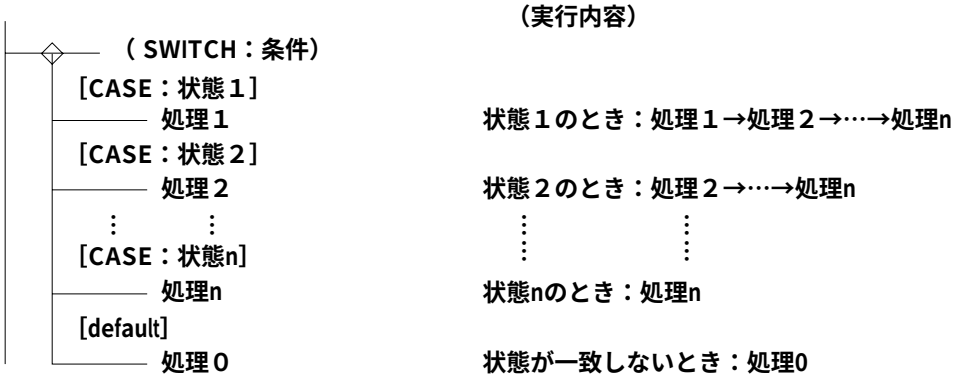


例．　入力文字により，月名を表示する

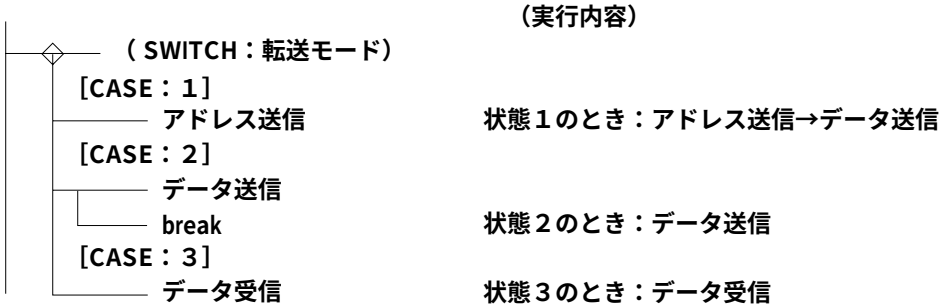


（2）一致した状態から処理が続く場合

・SPDチャート



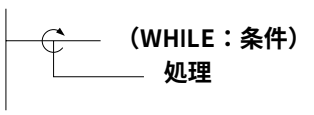
例．　シリアル・インタフェースの送受信



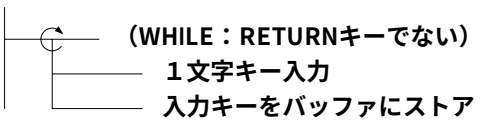
C.4　条件ループ（WHILE）

WHILEに示した条件を判定し，条件が成立している間，処理を繰り返し実行します（初めから条件が不成立の場合は，処理を実行しません）。

・SPDチャート



例．　RETURNキー入力があるまで，キーをバッファリングする

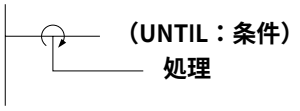


保守／廃止

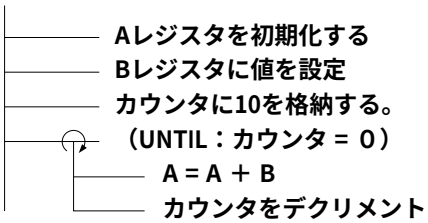
C.5　条件ループ（UNTIL）

処理を行ったあとにUNTILに示した条件を判定し，条件が成立するまで処理を繰り返し実行します（初めから条件が不成立の場合でも，処理を一度実行します）。

・SPDチャート



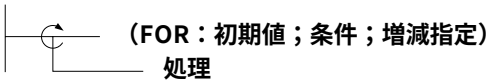
例．　Bレジスタの値を10倍してAレジスタに格納する



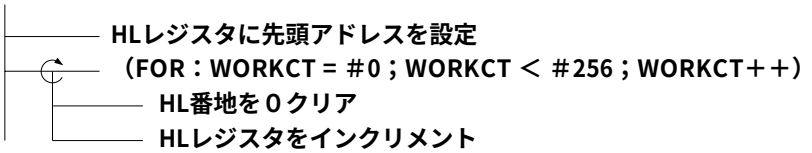
C.6　条件ループ（FOR）

FORに示されたパラメータの条件が成立している間，処理を繰り返し実行します。

・SPDチャート



例．　HL番地から256バイトを０クリアする

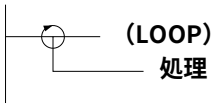


C.7　無限ループ

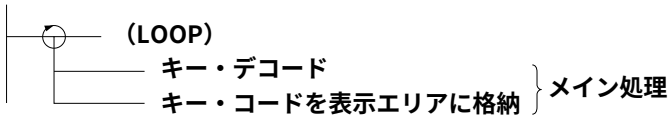
LOOPを設定すると，処理を無限に繰り返し実行します。

保守／廃止

・SPDチャート



例．　メイン処理を繰り返し実行する

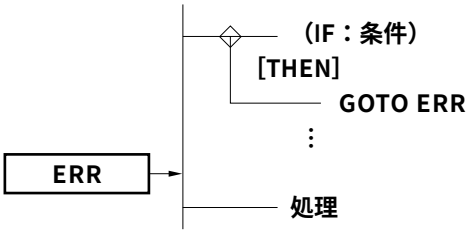


C.8　結合子（GOTO）

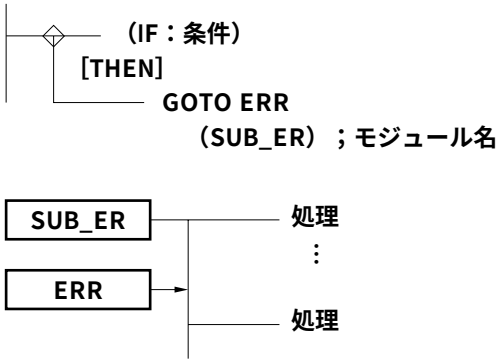
無条件に指定されたアドレスに分岐します。

・SPDチャート

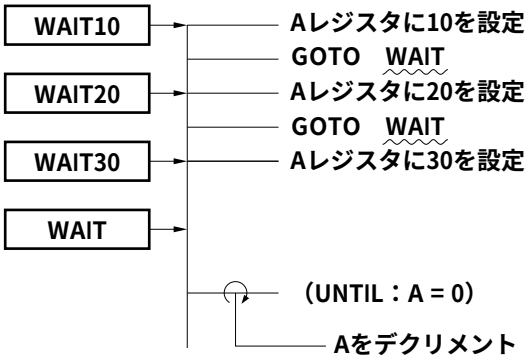
（1）同じモジュールに分岐



（2）異なるモジュールに分岐



例．　サブルーチンの開始アドレスで，パラメータを選択しウエイトを設定する



C.9　結合子（継続）

１モジュールのSPDが，複数のページにおよぶ場合に使用し，処理の流れを示します。

・SPDチャート

保守／廃止

アンケート記入のお願い

お手数ですが，このドキュメントに対するご意見をお寄せください。今後のドキュメント作成の参考にさせていただきます。

［ドキュメント名］μPD72070　アプリケーション・ノート　1 M/2 Mバイト FDD, 垂直磁化記録方式4 Mバイト FDD編
（IEA-751（第１版））

［お名前など］（さしつかえのない範囲で）

御社名（学校名，その他）　（　　）
ご住所　（　　）
お電話番号　（　　）
お仕事の内容　（　　）
お名前　（　　）

１．ご評価（各欄に○をご記入ください）

項　　目	大変良い	良　い	普　通	悪　い	大変悪い
全体の構成					
説明内容					
用語解説					
調べやすさ					
デザイン，字の大きさなど					
その他（　　）					
（　　）					

２．わかりやすい所（第　章，第　章，第　章，第　章，その他　　）
理由［　　］

３．わかりにくい所（第　章，第　章，第　章，第　章，その他　　）
理由［　　］

４．ご意見，ご要望

５．このドキュメントをお届けしたのは
NEC販売員，特約店販売員，NEC半導体ソリューション技術本部員，
その他（　　）

ご協力ありがとうございました。
下記あてにFAXで送信いただくか，最寄りの販売員にコピーをお渡しください。

NEC半導体インフォメーションセンター
FAX：（044）548-7900

キリトリ

保守／廃止

お問い合わせは，最寄りのＮＥＣへ

【営業関係お問い合わせ先】

半導体第一販売事業部 半導体第二販売事業部 半導体第三販売事業部	〒108-01　東京都港区芝五丁目７番１号（NEC本社ビル）	東　京	(03)3454−1111　(大代表)
中部支社 半導体第一販売部 半導体第二販売部	〒460　名古屋市中区錦一丁目17番１号（NEC中部ビル）	名古屋 名古屋	(052)222−2170 (052)222−2190
関西支社 半導体第一販売部 半導体第二販売部 半導体第三販売部	〒540　大阪市中央区城見一丁目４番24号（NEC関西ビル）	大　阪 大　阪 大　阪	(06) 945−3178 (06) 945−3200 (06) 945−3208

北海道支社	札　幌 (011)231−0161	太田支店	太　田 (0276)46−4011	富山支店	富　山 (0764)31−8461
東北支社	仙　台 (022)267−8740	宇都宮支店	宇都宮 (028)621−2281	三重支店	津　 (0592)25−7341
岩手支店	盛　岡 (019)651−4344	小山支店	小　山 (0285)24−5011	京都支社	京　都 (075)344−7824
山形支店	山　形 (0236)23−5511	長野支社	松　本 (0263)35−1662	神戸支社	神　戸 (078)333−3854
郡山支店	郡　山 (0249)23−5511	甲府支店	甲　府 (0552)24−4141	中国支社	広　島 (082)242−5504
いわき支店	いわき (0246)21−5511	埼玉支社	大　宮 (048)641−1411	鳥取支店	鳥　取 (0857)27−5311
長岡支店	長　岡 (0258)36−2155	立川支社	立　川 (0425)26−5981	岡山支店	岡　山 (086)225−4455
土浦支店	土　浦 (0298)23−6161	千葉支社	千　葉 (043)238−8116	四国支社	高　松 (0878)36−1200
水戸支店	水　戸 (029)226−1717	静岡支社	静　岡 (054)255−2211	新居浜支店	新居浜 (0897)32−5001
神奈川支社	横　浜 (045)324−5524	北陸支社	金　沢 (0762)23−1621	松山支店	松　山 (089)945−4149
群馬支店	高　崎 (0273)26−1255	福井支店	福　井 (0776)22−1866	九州支社	福　岡 (092)271−7700

【本資料に関する技術お問い合わせ先】

半導体ソリューション技術本部 システムマイクロ技術部	〒210　川崎市幸区塚越三丁目484番地	川　崎	(044)548−8891
半導体販売技術本部 東日本販売技術部	〒108-01　東京都港区芝五丁目7番1号（NEC本社ビル）	東　京	(03)3798−9619
半導体販売技術本部 中部販売技術部	〒460　名古屋市中区錦一丁目17番１号（NEC中部ビル）	名古屋	(052)222−2125
半導体販売技術本部 西日本販売技術部	〒540　大阪市中央区城見一丁目４番24号（NEC関西ビル）	大　阪	(06) 945−3383

半導体
インフォメーションセンター
FAX(044)548−7900
（FAXにてお願い致します）