JAERI-M
85-198

# VAX-11／780を使用したデータ収集システム (1)MBD-11のためのドライバーBDDRIVERと基本サブルーチン

１９８５年１２月

冨田　芳明・大内　　勲・菊池　士郎
池添　　博・杉本　昌義・花島　　進
丸山　倫夫

日　本　原　子　力　研　究　所
Japan Atomic Energy Research Institute

JAERI-Mレポートは、日本原子力研究所が不定期に公刊している研究報告書です。

入手の問合わせは、日本原子力研究所技術情報部情報資料課（〒319－11茨城県那珂郡東海村）あて、お申しこしください。なお、このほかに財団法人原子力弘済会資料センター（〒319-11茨城県那珂郡東海村日本原子力研究所内）で複写による実費頒布をおこなっております。

JAERI-M reports are issued irregularly.

Inquiries about availability of the reports should be addressed to Information Division, Department of Technical Information, Japan Atomic Energy Research Institute, Tokai-mura, Naka-gun, Ibaraki-ken 319-11, Japan.

© Japan Atomic Energy Research Institute, 1985

編集兼発行　日本原子力研究所
印　　刷　日立高速印刷株式会社

JAERI－M 85－198

# VAX－11／780を使用したデータ収集システム(I) MBD－11のためのドライバーBDDRIVERと基本サブルーチン

日本原子力研究所東海研究所物理部

富田 芳明・大内 勲・菊池 士郎・池添 博

杉本 昌義・花島 進・丸山 倫夫

（1985年11月12日受理）

タンデム加速器データ収集システムの改善のため，計算機をPDP－11 からVAX－11／780に切換えることになった。このための第一段階としてCAMACのブランチドライバーであるMBD－11のVAX－11／780へのインターフェース作業を行った。この報告は今回作成されたMBD－11のためのドライバーBDDRIVERとMBD－11 をFORTRANで使用するための基本サブルーチンの機能と使い方の解説である。MBD－11 のマイクロプログラムの作成法についてものべてある。

JAERI-M 85-198

# Data Acquisition System Using a VAX-11/780 Computer(I)

## BDDRIVER, a Driver Program for the MBD-11, and Basic FORTRAN subroutines

Yoshiaki TOMITA, Isao OHUCHI, Shiro KIKUCHI, Hiroshi IKEZOE,
Masayoshi SUGIMOTO, Susumu HANASHIMA and Michio MARUYAMA

Department of Physics, Tokai Research Establishment, JAERI

( Received November 12, 1985 )

In order to improve the performance of the data acquisition system for the JAERI tandem accelerator, it was decided to change the computer from PDP-11 computers to a VAX-11/780 computer. As the first step, a CAMAC branch driver MBD-11 was interfaced to the VAX-11/780. This report describes functions and usage of a driver program BDDRIVER and basic subroutines for FORTRAN. A guide to microprogramming the MBD-11 is presented also.

Keywords: Data Acquisition, Program, Computer, CAMAC, Guide, Performance

# 目　　次

CONTENTS

# 1. はじめに

原研タンデム加速器データ収集システム[1),2)]は これまで様々な核反応実験のために使用され，実験サイドから要求される多様な収集形態に速応できることが実証されてきた。しかし，設置以来すでに７年以上の年月が経過し，この間のめざましい計算機関連技術の発展により，計算機部分はかなり時代おくれとなってしまった。また老朽化による故障も多くなっており更新が必要となっている。

現システムで改善を必要とする問題点は次のとおりである。

① メモリー上にとれるデータ領域（４０ＫＢ）が小さすぎる。

② メモリーサイズが小さい（データ領域を除き，PDP－11／55 が２０８KB，PDP－11／04 が１６KB）ために高級言語が使用できず，すべてアセンブラーでプログラミングを行わなければならず，一般のユーザーがプログラムを変更することがかなり困難である。またすでに空領域がほとんどなくプログラムの大巾な改良の余地がなくなっている。

③ LIST モードでのデータ収集レートが実験によっては不十分である。（最高５０KB／秒）

④ 収集と平行してオンラインでデータ処理を行うためには演算速度が十分ではない。

これらの点を改善するために表記７名による検討が行われた。まず計算機に関しては，新しい計算機を導入することは予算上もまた必要とされる作業量の点からもさしあたりは困難であり，必然的に現在保有している，PDP－11／70 またはVAX－11／780 を使用することになった。両者を比較した場合，ほとんどの点でVAX－11／780 が優れており，多少問題となる点として，オペレーティングシステム（ＯＳ）のVAX／VMS がＰＤＰの場合のＲＳＸ－１１Ｍにくらべてずっと大きいためにＩ／Ｏ要求に対するレスポンスが遅く，また標準的でないＩ／Ｏを行うようなドライバー（Ｉ／Ｏを行うシステムプログラム）を作成することが困難であることがあげられた。前者に関してはLIST モードでのデータ収集の場合に問題となる可能性があったが，事前の検討によって実用的なデータレートでは問題とならないことが判明した。またやや特殊にならざるをえないドライバーの作成も，多少困難ではあっても可能と判断してVAX－11／780を使用することに決定した。先にのべた①－④の問題点はVAX－11／780 を使用することによってすべて解決できる。

次に問題となるデータ収集のフロントエンドについては，現在どおりCAMAC を使用することが最も現実的であり，インターフェースをどうするかという点が検討の対象となった。

はじめにのべた現システムのデータ収集形態に対する柔軟性はマイクロプログラム可能なブランチドライバーMBD－11[3),4)]の使用によることが大きく，新システムでも同様な機能を持つインターフェースを使用することが前提となった。MBD－11は１９７３年に米国ロスアラモス国立研究所で開発されたものであり，現在のＩＣ技術を用いればより高速化が可能であり，マイクロプロセッサーとしての演算種類ももっと豊富とすることができる。入手可能ないくつかのビットスライスのマイクロプロセッサーの仕様を検討すると現在の実行スピードを２

倍程度には容易に上げられることがわかる。しかし現在のCAMACからのデータの取込みの速度はマイクロプロセッサーの実行スピードよりも，CAMACの実行スピードとVAXやPDPのバスであるUNIBUS のスピードできまっており，実行スピードをあげても，現在かなり見おとりのするPHAモードのデータ取込み速度を現在の15μs から10μs程度に向上させることができるが，主要な収集モードであるLIST モードにはあまり関係がない。またMBD－11の命令は16ビット長できわめて簡明なFORMAT を持っており，新しいプロセッサーでは機能が拡張されるのと引かえにこの簡明さは失われ，プログラミングはやりにくくなる。また現在使えない乗除算命令等が使用できることはメリットであるがマイクロプログラム段階で必要とは言えない。結局のところ現在のMBD－11に対して大きな不満はなく，あまりマンパワーを投入することが不可能な現状も考慮して，あえて新しいインターフェースを開発する利点にないと判断しこれまで通りMBD－11を使用することとなった。

VAX－11／780とMBD－11を使用したデータ収集システムを作成するにあたって，まず行わなければならないのがMBD－11のためのドライバーの作成であり，リアルタイムの応用に十分耐えることができるようなドライバーを作成することができるかどうかが大きなキーポイントであった。これまでのところMBD－11のためのドライバーBDDRIVER と，FORTRANで使用可能な基本サブルーチンが完成し，これを用いたテストプログラムの実行によって期待した性能がえられることがわかった。LIST モードでのデータ取込み速度はMT へのダンプも含めて300KB／秒程度であり，ブロックサイズによって多少異なるが，現システムの約6倍となっている。またこのようなデータレートでも，ブロックサイズを極端に小さくとらない限ぎり，I／Oのために専有されるCPU時間は問題とならない。

ドライバーは今後問題点が見つかって多少変更することがあるかも知れないが基本的には現在のまま使用できると思われる。基本サブルーチンはテストプログラム用に作られたのでCOMMON ブロック等は実際のシステムを作成する際に大巾に変更されるが，使用法は現在と変える必要はないと思われる。このドライバーと基本サブルーチンを使用すれば，マイクロプログラムの作成以外はすべてFORTRANでプログラム可能である。マイクロプログラムも基本的なものはできているので，ほとんどの場合はこれらをそのまま，または多少修正して使用すればよいと思われる。特に，汎用でない個々の特殊な目的にだけ使用するような収集システムを作成しようとすれば，それほどの労力なしにできるはずである。

このレポートはドライバーBDDRIVER の基本的な機能と，QIO 要求の使用法，マイクロプログラムの作成法および使用法，基本サブルーチンの使用法についての解説である。対象としている読者は，

Ⓐ ドライバーを変更しようとする人。

Ⓑ マイクロプログラムを作成しようとする人。

Ⓒ 作成ずみのマイクロプログラムを使用して，収集システムを作成あるいは変更しようとする人。

である。Ⓐの読者を除いてドライバーの内部の動作に関する記述は完全に理解する必要はないし，このレポートの全体を読まなくてもよい。

実際にＶＡＸを使用したある程度汎用のシステムが完成するのはハードウェアの準備等もありしばらく後になる予定である。将来自分でマイクロプログラムを作成したい人はこのレポートを参考に自分なりのテストプログラムを作成して見るとよい。そうすることによって現在のドライバーの不十分な点も明らかになると思う。

なおテストの段階でＭＢＤ－１１がＶＡＸに対して割込み要求を行う際の信号のタイミングに正しくない点があり誤動作を起すことが判明し，回路の追加を行った。追加した回路はFig.1に示してある。

プログラムの作成は富田が行い，回路の変更は大内が行い，ソフト・ハードを含めたテストは富田，大内が行った。

# 2　VAX−11／780にMBD−11をインターフェースする上での基本的な考え方

## 2.1　MBD−11の概略

ここでこれからの議論のために，MBD−11についてごく簡単な説明を行っておこう。詳細は参考文献 1)，2)を参照されたい。

MBD−11は16ビットのマイクロプロセッサーであり，4kWまでの読み書き可能なメモリーを持ち，I／O ポートとしてCAMACのブランチドライバーとホスト計算機であるVAXまたはPDPのバスであるUNIBUS へのインターフェースを持っている。MBDにはRUNモードと，シングルサイクルモードと呼ばれるHALT状態があり，後者はイニシャライズ時にホストからプログラムをロードしたり，レジスターに初期値を書き込むために使用される。両者の切換えはホスト側から行われる。MBDには0－7の8個のチャンネル（VAX／VMSのI／Oチャンネルではない）があり，それぞれが独立の14個のレジスター（ファイルレジスターと呼ばれる）を持っている。14個のレジスターは更にBANK－0とBANK－1に分けられ，結局16組の7個ずつのレジスターセットがある。各チャンネルに属するプログラムは共通のメモリーに入れられ，ファイルレジスター中のCTRと呼ばれるレジスターによってポイントされる。RUNモードではCAMAC からのGL 17−24によって自動的にチャンネル0−7のプログラムが選ばれて起動され，またホスト側からチャンネルを指定してプログラムを起動することもできる。GLによる起動では0番地から，ホストからの起動では1番地からプログラムの実行が開始される。現システムでも新システムでも，GLによる起動ではBANK－0のファイルレジスターを，ホストからの起動ではBANK－1のファイルレジスターを使用するようにしており（プログラム中で再び切換えることも可能），特殊なジャンプ命令であるJVCの実行によりCTRを介して16個の独立なプログラムが独立のレジスターセットを使用して動くようになっている。CTR以外の6個のレジスターにはそれぞれILR，DAR，WCR，CCR，GP 1，GP 2 と名前がつけられているがすべて汎用のレジスターであり自由に使用できる。

これらのレジスターの他にチャンネルに共通のいくつかのレジスターがある。ATRには演算結果が入る。MBDからCAMACに対してI／Oを行うためにはブランチアドレスレジスターBARにFCNAをロードし，ブランチデータレジスターBDL，BDH を使用して，コントロールコマンドBC 0／BC 1 を実行する。またホストのメモリーにDMAでアクセスするためには，メモリーアドレスレジスターMARにUNIBUS ワードアドレスをロードし，メモリーデータレジスターMDRをデータレジスターとしてコントロールコマンドRDH／RDR／WTH／WTRを実行する。ここで，DMAと呼ぶのはすべてホスト側から見てであり，MBDから見ればプログラムI／Oである。MBDとホストのプログラムがデータをやり取りするためにはPDR（ホスト側からはPDXと呼ばれる）があり，主としてホストがMBDのプログラムを起動する際に補助データを入れるために使用される。またイニシャライズ時のメモリーやレジスター

のロードの際もこのレジスターを介して行う。

ＭＢＤはまたホストに対して２５種類の割込みを行うことができる（２５個の割込みベクターを持っている）。このうち最初の１６個はCAMACからのGL１－１６によってマイクロプログラムの関与なしに行われ，次の８個は各チャンネルに割当てられコントロールコマンドINTの実行によって行われる。残る１つはＭＢＤのエラーをホストに知らせるために用いられている。ＧＬによる割込みに対しては通常ホスト側からＧＬに対するサービスを実行するマイクロプログラムが起動される。

ホストからＭＢＤをコントロールするために，ホストからアクセスできる４個のレジスターがある。

ＣＳＲ：コントロールステータスレジスター

ＰＤＸ：プログラムデータレジスター

ＭＳＫ：割込みマスクレジスター

ＩＲ：インストラクションレジスター

ＣＳＲのビットを制御することによって，ＭＢＤのリセット，RUN／シングルサイクルモードの切換え，マイクロプログラムの起動（チャンネルイニシャライズ：ＣＩと呼ばれる），UNIBUS アドレスのビット１７のセット／リセット，割込み全体のイネィブル／ディスエイブル等を行う。ホストからのマイクロプログラムの起動はＣＩのビットとチャンネルの番号をＣＳＲにセットすることによって行われるが，これによってすぐプログラムの実行が始まるわけでなく，単に要求が行われるだけである。マイクロプログラムにはチャンネル番号の大きい方が高いプライオリティを与えられ，またＧＬによる起動とホストからの起動では，ホストからの起動に高いプライオリティが与えられている。実行中のプログラムがなくなった時点で要求のある最も高いプライオリティのプログラムが実行される。実行中のマイクロプログラムに対して割込みを行う機能は用意されていない。今後便宜上ホストから起動されるプログラムをＣＩ チャンネルのプログラム，GL１７－２４によって直接起動されるプログラムをＤＭＡチャンネル（通常ＤＭＡのデータ収集を行うので）と呼ぶことにする。ホストからの起動要求がペンディングであるかどうかはＣＳＲ中のREADYビットによって判断できる。以下で単に"ＣＩチャンネルを起動する"という表現を用いることが多いがこれは単にＣＩ ビットをセットするだけで，実際の実行には時間的な遅れがありうることに注意しなければならない。ＣＳＲにはまたブランチハイウェイとＤＭＡのエラーを示すビットもおかれている。ＰＤＸについてはすでにのべた。

ＭＳＫはGL１－１６による割込みを個別にイネィブル／ディスエイブルするためのマスクレジスターで，ビットがセットされている時対応する割込みが可能となる。

ＩＲはＭＢＤの命令レジスターで，シングルサイクルモードでＩＲにＭＢＤの命令をホストから書き込むことによって１ステップずつ命令を実行できる。主としてマイクロプログラムのロードやファイルレジスターに初期値をロードするために用いられる。

ＭＢＤの命令セットはTable１に示されている。命令の種類はきわめて少く，乗除算命令も含まれていないがデータ収集や装置のコントロールのためには十分である。以下にごく簡単にのべる。

○ MV　　データ移動。
○ INM, DEM　　インクリメント，デクリメント。
○ AD, SB　　加減算。
○ IOR, EOR, AND　　OR，イクスクルーシヴOR，AND。

これらはすべてソースとデスティネーションの2つのオペランドを持ち，同時に次のようなコントロールコマンドを実行することができる。

RDH, RDR　　DMAによるホストメモリーのREAD。（最後のH／Rはバスのホールド／リリース。）
WTH, WTR　　DMAによるホストメモリーへのWRITE。
BC0, BC1　　CAMACコマンドの実行。BC0ではF8＝0, BC1ではF8＝1。
EX1, EX2, EX4　　EXIT。
SET17, RST17　　CSR におかれているUNIBUS アドレスのビット17のセット，リセット。
INT　　ホストへの割込み。
BK0, BK1　　BANK－0, 1 のレジスターセットの選択。
BZ　　ブランチハイウェイのリセット。

○ BCT, BCF　　条件が成立／非成立によるブランチ。

判定する条件は次のようなものである。

BDF　　ブランチデマンドがある。
DCB　　DMA実行中。
BRB　　ブランチハイウェイオペレーション進行中。
INB　　割込み要求中。
NF　　ATRのビット15が1。
ZF　　ATR＝0。
ZLB　　ATRの下位バイトが0。
CF　　演算でCARRY発生。
QF, XF　　CAMACコマンドでQ，Xのレスポンス。
OF　　ATRのビット0が1。（ATRが奇数。）
GLB　　GL1－16による割込み要求中。
C12, C13, C14, C15　　CTRのビット12－15が1。

○ JVC　　CTRがポイントするアドレスへジャンプ。各マイクロプログラムへのディスパッチのために使用する。
○ LD, ST　　メモリーとATR間のロード，ストアー。
○ SLL, SRL　　ATRの論理シフト。
○ SLR, SRR　　ATRの循環。
○ LCI　　CTRへのイミーディエィトロード。

○ ＪＰ　　ジャンプ。

命令ＦＯＲＭＡＴ が単純なため，プログラミングはホストであるＰＤＰまたはＶＡＸのマクロ機能を利用してニーモニックで行うことができ，ビットスライスのマイクロプロセッサーとしてはきわめて簡単にプログラムできる。今回ＶＡＸ用にマクロ定義を書きかえるにあたりリロケーションも容易に行えるように改良した。

****** MBD INSTRUCTIONS ****

| Instruction | | Control-Condition | | | Source | | Destination | |
|---|---|---|---|---|---|---|---|---|
| MV | 0000 | ... | 000 | BDF | SWS | 00 | ATR | 0 |
| INM | 1000 | RDR | 100 | DC8 | ILR | 10 | ILR | 1 |
| DEM | 2000 | WTR | 200 | BR8 | DAR | 20 | DAR | 2 |
| AD | 3000 | RDH | 300 | INB | WCR | 30 | WCR | 3 |
| SB | 4000 | WTH | 400 | NF | CCR | 40 | CCR | 4 |
| IOR | 5000 | BC0 | 500 | ZF | CTR | 50 | CTR | 5 |
| EOR | 6000 | BC1 | 600 | ZL8 | GP1 | 60 | GP1 | 6 |
| AND | 7000 | EX4 | 700 | CF | GP2 | 70 | GP2 | 7 |
| BCT | 8000 | EX1 | 800 | QF | PCR | 80 | GLR | 8 |
| BCF | 9000 | EX2 | 900 | XF | MDR | 90 | MDR | 9 |
| JVC | A000 | SET17 | A00 | OF | MAR | A0 | MAR | A |
| ST | B000 | INT | B00 | GL8 | BDL | B0 | BDL | B |
| LD | C000 | RST17 | C00 | C12 | BDH | C0 | BDH | C |
| | | BK0 | D00 | C13 | CCL | D0 | BAR | D |
| LCI | E000 | BK1 | E00 | C14 | PDR | E0 | PDR | E |
| JP | F000 | BZ | F00 | C15 | MEM | F0 | MEM | F |

| | |
|---|---|
| SLR | D000 |
| SRR | D400 |
| SLL | D800 |
| SRL | DC00 |
| CON | 6000 |
| DEP | 6000 |

( SWS may be 0.)

< Absolute addressing >

STA=ST
LDA=LD
LCA=LCI
JPA=JP

***** < CSR Bits> ****

| | |
|---|---|
| Single | 1 |
| Reset | 2 |
| CI | 4 |
| Addr17 | 20 |
| IE | 40 |
| Ready | 80 |
| Channel | 700 |
| Branch error | 2000 |
| BUS error | 4000 |

*** < BAR Bits > ***

| | |
|---|---|
| A | F |
| N | 1F0 |
| C | E00 |
| F | F000 |

**** < Interrupt vectors(Octal) > ****

| | | | |
|---|---|---|---|
| GL 1 | 400 | GL 9 | 440 |
| GL 2 | 404 | GL10 | 444 |
| GL 3 | 410 | GL11 | 450 |
| GL 4 | 414 | GL12 | 454 |
| GL 5 | 420 | GL13 | 460 |
| GL 6 | 424 | GL14 | 464 |
| GL 7 | 430 | GL15 | 470 |
| GL 8 | 434 | GL17 | 474 |
| Chan-0 | 500 | Chan-4 | 520 |
| Chan-1 | 504 | Chan-5 | 524 |
| Chan-2 | 510 | Chan-6 | 530 |
| Chan-3 | 514 | Chan-7 | 534 |
| Error | 540 | | |

## 2.2 VAX−11／780でMBD−11 を使用する上での問題点

MBDがマイクロプログラム可能で，きわめて自由なDMAが実行できるということが，逆にOSの下でスピードと自由度をそこなわずに制御することを困難にする。現システムではMBDはPDP−11／04にインターフェースされている。PDP−11／04 はメモリーマネージメントを持たない計算機であり，アドレス指定はすべて物理アドレスで行われ，MBDからもホストのプログラムからも同じメモリーを同じアドレスで参照できる。しかもPDP−11／04がOSの下で動いていないのでMBDを制御する上でシステム上の制約はなにもなく，ユーザープログラムから直接MBDのレジスターに対してI／O を行っている。仮にマイクロプログラムが間違っていて，誤動作によってPDP−11／04 が止っても，主計算機のPDP−11／55からプログラムをロードし直して再起動するのにはほとんど時間がかからない。

VAXでMBDを使用する場合には，すべてのI／OはQIO システムサービスを要求することによって，ドライバーを通じて行うことになる。一般にI／O システムサービスのレスポンスはシステムが大きくなるにしたがって悪くなる。例えば，実際のI／O のために必要な時間が無視できる磁気テープユニットのステータスをセットあるいはセンスするQIO を実行してみると，VAX−11／780 では3.2ms，RSX−11M の下で動くPDP−11／70では1.0msかかる。VAX−11／780の実行スピードがPDP−11／70 の約2倍であることを考えるとOS が大きくなったためにI／O のレスポンスが大巾に悪くなっていることがわかる。さらに現システムと比較すれば，MBDのマイクロプログラムを起動するのにPDP−11／04で1命 令を実行するだけなのがVAX−11／780では同じ1命令を実行するために3.2ms というシステムのオーバーヘッドがついてしまうことになる。

またVAX−11／780 ではUNIBUS 上の装置からの割込みが，直接に対応するサービスルーチンを起動せず，UNIBUS 全体からの割込みを一括して受ける（ダイレクトベクターでない）ので，割込みに対するレスポンスも遅いことも考慮しなくてはならない。実際にUNIBUS上の信号を調べてみるとMBDから割込みを行うためにバス要求を行ってからバスグラントが返されるまでに約60μsもかかっている。

またアドレッシングについて考えてみると，VAXは仮想記憶を採用しているので，プログラム上のアドレスと物理アドレスはページテーブルエントリー(PTE)とよばれるテーブルを通じて関係づけられ，しかもこの関係は不変ではなく，対象となる領域やPTE自身すらもいつもメモリー上にあるとは限らない。またVAXのアドレスは32ビットであるがUNIBUSのアドレスは18ビットであり，間にUNIBUS マップレジスターが介在してアドレスの変換を行う必要がある。こういった事情からVAX−11／780では一般にDMAのためのQIO要求に対しては，まずユーザーが指定したバッファー領域がユーザーに対してアクセスが許されているかチェックした上で，領域を物理メモリー上にロックする。この際領域が物理メモリー上になければディスクから読み込まれる。次に必要なだけのUNIBUS マップレジスターを確保し，PTEを参照して領域の物理アドレス（ページアドレス）をマップレジスターにロードすることによってUNIBUS アドレスとユーザー領域を対応させる。その上でI／O を開始し，DMAが終了した後マップレジスターや物理メモリーのロックを解除してI／Oが終了する。

このような過程は先にのべたＱＩＯ のシステムオーバーヘッドを更に大きくするので，データ収集システムのようなリアルタイムシステムではできるだけ避ける必要がある。

最後に大きな問題として，マイクロプログラムの変更はシステムから見た場合に装置のハードウェアの変更に相当するということがある。実際にはマイクロプログラムはユーザー側で作成され，ドライバーはその内容をチェックすることはできない。したがってドライバーが予想していないようなマイクロプログラムが実行されるとシステムのエラーとなって，システムのクラッシュを引起したり，高いプラリオリティの部分での無限ループとなってシステムのレスポンスがなくなったりすることが容易に起りうる。このようなことは完全に防止することはできないが，マイクロプログラムの役割とプログラム作成時の規則を明確にしてできるだけシステム的なエラーが起らないようにする必要がある。

## 2.3 ＭＢＤ−１１のためのドライバーＢＤＤＲＩＶＥＲ で採用したＩ／Ｏのやり方

2.2でのべたような問題点を解決するために作成されたドライバーBDDRIVER（MBDのシステム上の装置名をＢＤとした。）で採用したＭＢＤの使い方についてここでのべる。ここでとったやり方は，これまでのＭＢＤの使用経験にてらしてＩ／Ｏのやり方に一定の制約を加えたものである。特にデータ収集の主な形態がＬＩＳＴモードであることが考慮されている。しかしこのためにCAMACに対する特定のＩ／Ｏが不便になったり遅くなったりしていることはないはずである。ただこれまでのシステムで許されてきた自由度をもっと狭くすれば，ドライバー自体の構造はもっと単純で標準的なものにすることは可能である。

ＢＤＤＲＩＶＥＲではシステム上にＭＢＤ−１１が１台しかないことを前提としている。２台以上使用するためにはＢＤＤＲＩＶＥＲ内においているいくつかのステータス情報をＭＢＤ−１１の番号でインデックスするように変更する必要がある。また異ったUNIBUS アダプターで使用する場合は問題とならないが，同じアダプターに２台以上のＭＢＤ−１１をつなぐ場合には以下にのべるＤＭＡ領域の扱いをかなり変更する必要がある。

次に，ＭＢＤ上ではいろいろなＩ／Ｏが並行して行われるが，一般にＩ／Ｏの要求はその装置に対する以前のＩ／Ｏが終了するまで待たされるので形式的にＭＢＤのチャンネルをすべて独立のユニットと見なすことにした。さらにＶＡＸからＣＩ によって起動されるＣＩ チャンネルとＧＬ１７−２４ によって直接起動されるＤＭＡチャンネルも独立のユニットとした。これによって１台のコントローラーＢＤＡ： に BDA0：～ＢＤＡ１５：の１６台のユニットが接続されている形式になっている。ユニット番号はＭＢＤのチャンネル番号ｎ（０～７）に対してＣＩ チャンネルを2n，DMAチャンネルを２ｎ＋１ とした。このように形式上は１６個のマイクロプログラムが独立の装置となったわけであるが，実際には相互に関連があって一定のデータ収集を行うわけである。このために最初のユニットＢＤＡ０：を全体を代表するユニットとして，ＢＤＡ０：を使用するプロセスだけが他のユニットも使用できるようにした。このようにすることによってドライバーのコード中にコントロールのためのテーブルを置くことができるようになり，システム上独立なユニットを関連づけて制御することが容易になる。具体的には，ＭＢＤを使用する際最初にＩＯ$＿Attach のＱＩＯをＢＤＡ０：に対して行うことにし，これによって

BDDRIVER はそのプロセスの識別番号(PID)をドライバー中のテーブルに書き込み，以後すべてのQIO 要求に対してそのPID をチェックして他のプロセスからのアクセスを排除するようにした。このMBDの専有はそのプロセスがIO$_Detach のQIO を行ってMBDの使用をやめるまで継続する。

次にマイクロプログラムからのDMAをできるだけ単純にして，システムのオーバーヘッドを小さくするとともにマイクロプログラムのエラーによるシステム全体への影響を少くするために，DMAを行える領域を固定することにした。この制限は一見厳しすぎるように思われるが，データ収集システムをマルチプロセスで構成するためにはデータ領域をグローバルセクション上にとる必要があり結局のところ半固定的にならざるをえないので大きな問題ではない。このことによってUNIBUS マップレジスターによるマッピングを固定することができ，2.2でのべたようなI/Oの都度マップレジスターをロードするといった過程をはぶくことができる。

UNIBUS マップレジスターはドライバーのロード時に256個確保しMBDのために専有することにした。このことはすべてのシステムで可能なわけではないが他に特別の装置が同じUNIBUS につながれていない限り問題ない。タンデム加速器のVAXシステムでは2台のUNIBUS アダプターが設置されており，MBDをつなぐUNIBUS-Bには他にRA-60のディスクがつながっているだけであり全く問題がない。さらにシステムのブート時にマップレジスターを確保すれば0-255番のマップレジスターを確保することができる。こうすることによってMBDはUNIBUS アドレス空間のビット17が0である下位の128KBを専有して使用することになる。これによって2.1でふれたCSR上のUNIBUS アドレスの拡張ビットは常に0でよく，DMAの都度アドレスをチェックしてセットやリセットを行う必要がなくなる。同じUNIBUS アダプターの他の装置はビット17が1のアドレスを使用することになる。こうすることによってマイクロプログラムが誤ったDMAを実行しても，SET17のコントロールコマンドによってUNBUSアドレスのビット17をセットしてない限り，ユーザーのDMA領域以外をアクセスすることはなく，システム的なトラブルを起すことはなくなる。最初のQIO であるIO$_Attach ではDMA領域のアドレスとサイズをパラメーターとして指定し，BDDRIVER はこの領域をIO$_Detachが行われるまでメモリー上にロックし，この領域をアクセスするようにUNIBUS マップレジスターをロードする。この際255番以下のレジスターだけを使用し，256番以上は使用しない。実際にはいつも0-255番のすべてのレジスターが使用できるはずである。マップされた領域のUNIBUS アドレス（0番から使用すれば0）とサイズはユーザープロセスに返されるので，マイクロプログラムはそのアドレスを使用してDMAを行う。5.にのべる基本サブルーチンでは128KBのグローバルセクションをDMA領域にとっている。この領域はマイクロプログラムが直接DMAを行う領域であり，LISTモードでとり込んだイベントデータからオンラインで作成するスペクトルの領域は別にとればよいので，予想されるタンデム加速器の実験のためには十分と思われる。

次に使用するマイクロプログラムの種類を

"LIST"，"READ"，"PHA"，"Start_Stop"，"Command"，"CI"

の6種類に分類し，それぞれに対して一定のわくをはめることにした。それぞれのタイプに対

して，例えばMPRO__type $K_LISTのように頭にMPRO_type $K__ をつけたコードが定義され，プログラムをロードする際ＱＩＯ のパラメータの１つとして指定することにしている。以下に各タイプについてのべる。実際にマイクロプログラムを作成する際に守らなければならない事項の詳細は4.で補足する。

(1) ＬＩＳＴ

ダブルバッファーを使用してＬＩＳＴモードのデータ収集を行うためのマイクロプログラムでＱＩＯ のシステムオーバーヘッドを少くするために，個々のバッファーに対するRead 要求のＱＩＯ をその都度ユーザープロセスから行う必要をなくしてある。プログラムはＧＬ１７によって直接起動されCAMAC から読んだデータをバッファーに書き込む。バッファーが一杯になった時，ＶＡＸに対して割込みを行い，これによってユーザープロセス中のＡＳＴサービスルーチンが起動される。バッファーの切換えはマイクロプログラムがＤＭＡ領域内のフラグを見ることによって行う。必ずチャンネル０を用いる。ＤＭＡチャンネルのＢＤＡ１：だけでなくＣＩ チャンネルのＢＤＡ０：も使用する。これは途中まで書き込まれた状態のバッファーをＶＡＸ側から完成させるために用いられる。詳しくは4.で述べる。

(2) ＲＥＡＤ

ＩＯ$_READLBLK のＱＩＯ によってＩ／Ｏが開始される。バッファーの領域がＤＭＡ領域に限られることを除くとＭＴ等に対する通常のＩＯ$_ＲＥＡＤＬＢＬＫのＱＩＯと同じである。ＧＬにより直接起動され，CAMAC からのデータをバッファーに書き込み，バッファーが一杯になったらＶＡＸに対して割込みを要求し，これによってＱＩＯ が完了する。ＬＩＳＴモードのデータ収集のために使用することもできる。ＬＩＳＴタイプのやり方にくらべ１つのバッファー当り約１ms システムのオーバーヘッドが増加するが，ユーザープロセスとの同期は単純でマイクロプログラムの構造は簡単である。これを用いれば２系統以上のＬＩＳＴモードのデータ収集を同時に行うことも可能である。ＬＩＳＴタイプの項でのべたのと同じ理由でＤＭＡチャンネルの他に対応するＣＩ チャンネルも使用する。チャンネルは０－６を用いる。

(3) ＰＨＡ

主としてＰＨＡモードのデータ収集のために使用される。ＧＬによって直接起動されDMA領域に対してＩ／Ｏを行う。ＶＡＸに対する割込みは行わず，ドライバーも含めてＶＡＸのプログラムとの交流は全くない。任意のＤＭＡチャンネルを使用できる。

(4) Start__ Stop

ＰＨＡモードやＬＩＳＴモード等のＤＭＡによるデータ収集のSTART／STOP を行う。具体的にはこれらのマイクロプログラムを起動するＬＡＭをイネィブル／ディスエイブルしたり，必要なゲート信号を制御したりする。ＩＯ$__Start__Stop のＱＩＯ によって起動される他，ＩＯ$__ＬＩＮＫ__ＧＬ のＱＩＯ によってあらかじめ結びつけられているＧＬ（１－１６）による割込みサービスルーチンからもＢＤＤＲＩＶＥＲによって起動される。後者の場合は通常ユーザープログラム中のＡＳＴサービスルーチンも起動される。マイクロプログラムの起動の際にＢＤＤＲＩＶＥＲ中のSTART／ＳＴＯＰのフラグも変更される。ＱＩＯ によって起動された時にだけＶＡＸに対して割込みを行いＱＩＯ を完了させる。いつもＣＩ チャンネル７（ＢＤＡ１４：）を用いる。これはプライオリティーを最も高くするためである。

(5) Command

IO$__Command のＱＩＯによって起動されＤＭＡ領域内に置かれた一連のＣＡＭＡＣコマンドを実行する。終了後ＶＡＸに対して割込みを行い，これによってＱＩＯ が終了する。形式的には次のＣＩ タイプに含めることができるが，必ず使われるマイクロプログラムなので独立させた。ＣＩ チャンネル０－６を用いる。

(6) ＣＩ

IO$__ＣＩのＱＩＯ によって起動され，最後にＶＡＸに対して割込みを行うことでＱＩＯが終了する。ＤＭＡを含め何を行うかは自由である。IO$__LINK__GLのＱＩＯ で結びつけられたGL(1－16)によっても起動させることができる。この時はＶＡＸに対して割込みは行わない。しかしIO$__SETMODEのＱＩＯでＡＳＴ ルーチンを指定しておけば，ＧＬが発生した時ユーザープロセスのＡＳＴルーチンを起動させることができる。ＣＩ チャンネル０－６を用いる。

## 3. MBD－11のためのＱＩＯ

BDDRIVER に対するＱＩＯの機能と使い方をファンクションコード毎にのべる。以下でＰ1－Ｐ6はＱＩＯで指定する装置個有のパラメークである。またＩ／Ｏステータスブロックは

```
INTEGER*2          IOST_W(4)
INTEGER*4          IOST_L(2)
EQUIVALENCE        (IOST_L, IOST_W)
```

を指定したとして記述を行う。ここでの記述はFORTRANでの使用を考えて行っているが，厳密を期するためＰ1~Ｐ6の指定の仕方はVAX／VMSシステムサービスリファレンスマニュアルに準じて記述する。したがって例えば

Ｐ１：バッファーのアドレス

Ｐ２：バッファーのサイズ

を指定するには，バッファーの配列名がBuffer でありそのバイトサイズがlength に入れられているとして

SYS$QIO(…… , Buffer , %val(length),,,)

のように書かなければならない。

以下で用いられるファンクションコードやすでにのべたマイクロプログラムのタイプを表すコード等標準的でないコードは，MACROではマクロライブラリー

〔DATACQ.MBD〕 MBDMAC.OLB

中の$MBDDEF によって

FORTRAN ではテキストライブラリー

〔DATACQ.TLB〕DATACQ.TLB

中のMBDDEFによって定義されている。

(5) Command

IO$__Command のＱＩＯによって起動されＤＭＡ領域内に置かれた一連のＣＡＭＡＣコマンドを実行する。終了後ＶＡＸに対して割込みを行い，これによってＱＩＯ が終了する。形式的には次のＣＩ タイプに含めることができるが，必ず使われるマイクロプログラムなので独立させた。ＣＩ チャンネル０－６を用いる。

(6) ＣＩ

IO$__ＣＩのＱＩＯ によって起動され，最後にＶＡＸに対して割込みを行うことでＱＩＯが終了する。ＤＭＡを含め何を行うかは自由である。IO$__LINK__GLのＱＩＯ で結びつけられたGL(1－16)によっても起動させることができる。この時はＶＡＸに対して割込みは行わない。しかしIO$__SETMODEのＱＩＯでＡＳＴ ルーチンを指定しておけば，ＧＬが発生した時ユーザープロセスのＡＳＴルーチンを起動させることができる。ＣＩ チャンネル０－６を用いる。

## 3. MBD－11のためのＱＩＯ

BDDRIVER に対するＱＩＯの機能と使い方をファンクションコード毎にのべる。以下でＰ1－Ｐ6はＱＩＯで指定する装置個有のパラメータである。またＩ／Ｏステータスブロックは

```
INTEGER*2          IOST_W(4)
INTEGER*4          IOST_L(2)
EQUIVALENCE        (IOST_L, IOST_W)
```

を指定したとして記述を行う。ここでの記述はFORTRANでの使用を考えて行っているが，厳密を期するためＰ1～Ｐ6の指定の仕方はＶＡＸ／ＶＭＳシステムサービスリファレンスマニュアルに準じて記述する。したがって例えば

Ｐ１：バッファーのアドレス

Ｐ２：バッファーのサイズ

を指定するには，バッファーの配列名がBuffer でありそのバイトサイズがlength に入れられているとして

```
SYS$QIO(…… ,Buffer , %val(length),,,,)
```

のように書かなければならない。

以下で用いられるファンクションコードやすでにのべたマイクロプログラムのタイプを表すコード等標準的でないコードは，ＭＡＣＲＯではマクロライブラリー

〔DATACQ.MBD〕 MBDMAC.OLB

中の$MBDDEF によって

FORTRAN ではテキストライブラリー

〔DATACQ.TLB〕DATACQ.TLB

中のMBDDEFによって定義されている。

なお5でのべる基本サブルーチンを使用すればほとんどの場合直接ＱＩＯ を要求する必要はない。

IO$__Attach

ＭＢＤを専有して使用することを宣言するＱＩＯですべてのＱＩＯに先だってBDA0：に対して行わなければならない。IO$__AttachによってＭＢＤを専有していないプロセスがMBDに対してＱＩＯを行っても要求は受けつけられず，SS$__NOPRIV のリターンステータスが返される。同一のプロセスから，くり返しIO$__Attach を行うためには，一旦ＩＯ$__Detach のＱＩＯを行ってＭＢＤの使用をやめてから行わなければならない。

Ｐ１：ＤＭＡ領域の先頭アドレス。ページ境界にアラインされていなければならない。

Ｐ２：ＤＭＡ領域のページ数。最大２５６ページ。

Ｐ３：BDDRIVER から個々のＱＩＯに直接関連しないメッセージを受けとるためのメールボックスのユニット番号。このメールボックスは永久メールボックスでなくてはならない。使用しない場合は指定しなくてよい。通常はブート時に作成される永久メールボックスMBD$__Mailboxのユニット番号を指定する。ユニット番号は$GETDVIのシステムサービスを用いて知ることができる。指定すると，BDDRIVER はMBDがエラーを起したり，CAMACから予想していない割込みが込ったりした場合，このメールボックスにメッセージを送る。ユーザープロセス（別の独立のプロセスでもよい）はＡＳＴ付でこのメールボックスに対してRead を行い，ＡＳＴルーチンでメッセージをターミナル等に出力するとよい。

正常終了時のＩ／Ｏステータスブロック

IOST__W(2)：最初のUNIBUS マップレジスターの番号（通常は０）。

IOST__L(2)：UNIBUS マップされたＤＭＡ領域のバイト数。

UNIBUS マップレジスターの番号はUNIBUS アドレスのビット９－１７ に対応する。BDDRIVER では２５５番以下を使用するのでビット１７は常に０である。したがって例えばＤＭＡ領域が

INTEGER*2 DMA(64*1024)

のようにとられているとすると，ＤＭＡ(n)に対するUNIBUS アドレスは

IOST__W(2)*512+2*(n－1)

となる。ＭＢＤはワード単位でUNIBUS にアクセスするので，マイクロプログラムはこの値を２で割ったワードアドレスをＭＡＲにロードしてＤＭＡを実行する。

エラー時のリターンステータス

SS$__PROTOCOL　　：すでにＩＯ$__Attachが行われている。

SS$__BUFNOTALIGN：ＤＭＡ領域がページ境界にアラインされていない。

SS$__ILLPAGCNT　　：ページ数の指定がない。

SS$__INSFMAPREG　：ページ数が確保したマップレジスター数より多い。

SS$__IVCHAN　　：BDA0：以外のユニットを指定した。

IO$__Detach

MBDの使用をやめるためのQIOでBDA0：に対して行わなければならない。これによりIO$__Attach のQIOが行われる以前の状態になり他のプロセスがMBDを使用することが可能となる。またMBDはリセットされ，メモリー上にロックされていたDMA領域はロックを解除される。なおこのQIOが要求される以前に，MBDに対するすべてのQIOは完了しており，DMAはSTOP状態（BDDRIVER 内のSTART／STOPのフラグがSTOP状態）でなければならない。ペンディングとなっているQIOに対しては$CANCEL のシステムサービスを実行しなければならない。

エラー時のリターンステータス

SS$__PROTOCOL ：IO$__Attach が行われていない。
SS$__DEVACTIVE：DMAがSTART状態か，終了していないQIOがある。
SS$__IVCHAN　　：BDA0：以外のユニットを指定した。

IO$__INIT

3つのファンクションモディファイアー

IO$M_INIT_Begin
IO$M_INIT_Modify
IO$M_INIT_End

をつけて行われる。ユニットに共通なQIOであり，BDA0：に対して実行する。便宜上IO$__INIT__Begin で開始される期間を" INIT中" と呼び，IO$__INIT__Modify で開始される期間を" INIT—Modify 中" と呼ぶことにする。後者は収集モードの一部変更のために用いられる。どちらの期間もIO$M_INIT__End のQIOによって終る。この期間はイニシャライズのためのCAMACコマンドの実行の間を除きMBDはシングルサイクルモードにある。この期間以外では収集モードの設定や変更は許されず，MBDのメモリーやファイルレジスターのRead／Write ，GLとマイクロプログラムのリンクはすべてこの期間内に行わなくてはならない。

IO$__INIT のQIOを出す時は，DMAはSTOP状態でなくてはならず，また終了していないMBDに対するQIOがあってはならない。

＜IO$M__INIT__Begin＞

このQIOによってINIT中となる。すべてをリセットして収集モードの設定をはじめからやり直す。まずMBDがリセットされ，シングルサイクルモードにされる。ブランチハイウェイにもBZが送られ，CAMACもイニシャライズされる。各ユニットのマイクロプログラムのタイプもすべてリセットされ，GLとマイクロプログラムのリンクもリセットされ，すべてのASTルーチンがとり消される。次にMBDのメモリーの0－7番地が次のようにロードされ

る。

```
0 :   JVC
1 :   MV      PDR, ATR
2 :   CON     BK1
3 :   JVC
4 :   BCT     $, INB
5 :   CON     INT
6 :   BCT     $, INB
7 :   CON     EX4
```

これらはマイクロプログラム起動のためのコードである。ＤＭＡチャンネルがＧＬ１７－２４により起動されるとＢＡＮＫ－０のファイルレジスターが選択され，０番地から実行がはじまる。ＪＶＣ命令によってそのチャンネルのＣＴＲに入っているアドレスへジャンプし，ロードされているプログラムが実行される。ＢＤＤＲＩＶＥＲ によってＣＩチャンネルが起動されると実行は１番地からはじまる。まずＢＤＤＲＩＶＥＲ によってＰＤＲに入れられているデータをＡＴＲに移す。これは次のチャンネルを起動するためにＢＤＤＲＩＶＥＲがＰＤＸ（＝ＰＤＲ）に新しいデータをロードする可能性があるからである。続いてＢＡＮＫ－１のレジスターセットを選び，３番地のＪＶＣによってＢＡＮＫ－１のＣＴＲに入っているアドレスへジャンプする。４番地以降はマイクロプログラムがロードされていないＤＭＡチャンネルがまちがって起動された時の対策である。ＩＮＩＴ終了時に使用されていないＤＭＡチャンネルのＣＴＲにはすべて４がロードされる。これによって正しくないチャンネルがＧＬにより起動された時ＶＡＸに対して割込みを行い（ＢＤＤＲＩＶＥＲが予想していない割込み）エラーのあったことを知らせるとともに最後のＥＸ４によって連続して同じチャンネルが起動されることを防いでいる。したがって特別の理由がない限り，マイクロプログラムは８番地以降にロードしなければならない。

エラー時のリターンステータス

ＳＳＳ＿ＰＲＯＴＯＣＯＬ ：すでにＩＮＩＴ中あるいはＩＮＩＴ—Ｍｏｄｉｆｙ 中である。

ＳＳＳ＿ＤＥＶＡＣＴＩＶＥ：ＤＭＡがＳＴＯＰ状態でないか，終了していないＱＩＯがある。

＜ＩＯＳＭ＿ＩＮＩＴ＿Ｍｏｄｉｆｙ ＞

これによってＩＮＩＴ— Ｍｏｄｉｆｙ 中となる。この期間は主としてファイルレジスターの書きかえのために用いられ，マイクロプログラムのロードやＧＬとマイクロプログラムのリンクは許されない。ＭＢＤはリセットされるが，ＣＡＭＡＣはイニシャライズされない。リセットに先だってＭＳＫレジスターはセーヴされ，ＩＮＩＴ終了時に再びロードされる。収集モードはすべてそのままで，何もリセットされない。

エラー時のリターンステータスはＩＯＳＭ＿ＩＮＩＴ＿Ｂｅｇｉｎ と同じである。

＜ＩＯＳＭ＿ＩＮＩＴ＿Ｅｎｄ ＞

収集モードの設定を終了するＱＩＯでこれによってＩＮＩＴ中あるいはＩＮＩＴ—Ｍｏｄｉｆｙ 中

が終了する。まず各ＤＭＡチャンネルにマイクロプログラムがロードされているかどうかチェックし，ロードされていればシングルサイクルで

CON　　EX 2

を実行してチャンネルイネィブルラッチ（CEL）をセットしＧＬによってマイクロプログラムが起動されるようにする。マイクロプログラムがロードされていないチャンネルに対しては

CON　　EX 4

によってＣＥＬをリセットすると共にＢＡＮＫ－０のＣＴＲに４をロードし，先にのべたように誤って起動されても予想していないアドレスから実行がはじまることがないようにする。
（ＣＥＬは対応するＣＩ チャンネルのプログラムのEXIT時にセットされてしまう。）

次にＩＮＩＴ中の場合はＩＯ$__ＬＩＮＫ__ＧＬ のＱＩＯによってマイクロプログラムと結びつけられたＧＬに対応するＭＳＫレジスターのビットをセットし，これらのＧＬによる割込みを可能にする。ＩＮＩＴ－Modify中の場合は最初にセーヴしておいた値をＭＳＫレジスターにロードし直す。

最後にＭＢＤ をＲＵＮモードにし，割込みも可能にする。

エラー時のリターンステータス

ＳＳ$__ＰＲＯＴＯＣＯＬ　：ＩＮＩＴ中でもＩＮＩＴ－Modify 中でもない。

ＳＳ$__ＤＥＶＡＣＴＩＶＥ：終了してないＱＩＯがある。

IO$__WRITEMBD

２つのモディファイアー

IO$M__MEMORY

IO$M__FREG

のどちらかをつけて実行される。

＜IO$M__MEMORY＞

ＭＢＤの各ユニットに対してマイクロプログラムをロードするために使用する。ＩＮＩＴ中でのみ許される。2.3でのべたようにＬＩＳＴタイプとＲＥＡＤタイプのマイクロプログラムはＤＭＡチャンネルと共に対応するＣＩ チャンネルも使用するが，プログラムは一連のものとしてＤＭＡチャンネルのユニットに対してロードしなければならない。

Ｐ１：マイクロプログラムが入っている領域のＶＡＸのアドレス。

Ｐ２：マイクロプログラムのワードサイズ。

Ｐ３：ロードするＭＢＤのアドレス。

Ｐ４：マイクロプログラムのタイプ。

エラー時のリターンステータス

ＳＳ$__ＰＲＯＴＯＣＯＬ：ＩＮＩＴ 中でない。

ＳＳ$__ＢＡＤＰＡＲＡＭ：ワードサイズが０。

SS$__IVCHAN ：MBDのユニット番号がそのタイプに許されているユニットでない。
SS$__DEVREQERR ：すでにプログラムがロードされているユニットに対してロード要求が出された。
SS$__ENDOFVOLUME ：マイクロプログラムが大きすぎてメモリーに入りきらないか，ロードアドレスが正しくない。

＜IO$M__FREG＞

各ユニットのファイルレジスターをロードする。INIT中またはINIT－Modify中でのみ許される。

P 1：ILR，DAR，WCR，CCR，CTR，GP 1，GP 2の順にならべられた7ワードの配列の先頭アドレス。

P 2：レジスターを書きかえるためのマスク。ビット0がILR，ビット6がGP 2のように上にあげた順にビットが割当てられる。対応するマスクビットが1である場合にだけレジスターは書きかえられる。

上にあげた順序はMBDDEF中で定義されており，必ずしも意識しなくてもよい。5.2のLoad__File__Registerの項にくわしくのべる。

エラー時のリターンステータス

SS$__PROTOCOL：INIT中でもINIT－Modify中でもない。

IO$__READMBD

2つのモディファイアー

IO$M__MEMORY

IO$M__FREG

のどちらかをつけて実行される。INIT中またはINIT－Modify中でのみ許される。

＜IO$M__MEMORY＞

MBDのメモリーを読むためのQIOである。アドレス指定で読むのでユニット番号はどれでもよい。

P 1：読んだデータを入れる領域のアドレス。

P 2：読むワード数。

P 3：読みとるMBDの先頭アドレス。

実際に読んだワード数はIOST__W(2)に入れられる。

エラー時のリターンステータス

SS$__PROTOCOL：INIT中でもINIT－Modify中でもない。

SS$__BADPARAM：ワード数が0。

SS$__IVADDR ：MBDのアドレスが存在しないアドレスである。

＜IO$M__FREG＞

各ユニットの7個のファイルレジスターを読む。

P1：ファイルレジスターの値を入れる，7ワードの配列のアドレス。レジスターの順はロードの時と同じである。

エラー時のリターンステータス

SS$__PROTOCOL：INIT中でもINIT－Modify中でもない。

IO$__LINK__GL

GL（1－16）による割込みを使用するためのQIOであり，GLとCIチャンネルのマイクロプログラムをリンクする。INIT中でのみ使用できる。すべてのGLに対するリンクを1つのQIOで行うのでBDA0：に対して実行すべきである。1つのマイクロプログラムにいくつものGLをリンクしてもよい。CIチャンネルを起動する際PDXにロードするコードもここで指定する。マイクロプログラムはこのコードによってのみ，QIOで起動されたかあるいはどのGLで起動されたかの判別ができる。

これらのデータは次のように指定する。

INTEGER＊2　Channel__code（2，16）

Channel__code（1，n）：GLnの割込みが起きた時起動されるCIチャンネルの番号（0－7）。このチャンネルのプログラムのタイプはCIまたはStart__Stopでなくていなければならない。使用しないGLに対しては0－7以外の数（－1がよい）を入れる。

Channel__code（2，n）：GLnによる起動の際PDXにロードするコード。

このQIOを出す前にマイクロプログラムはロードされていなければならない。

P1：Channel__codeのアドレス。

エラー時のリターンステータス

SS$__PROTOCOL：INIT中でない。

SS$__BADPARAM：指定されたCIチャンネルのマイクロプログラムのタイプがCIでもStart__Stopでもない。あるいはマイクロプログラムがロードされていない。

IO$__Command

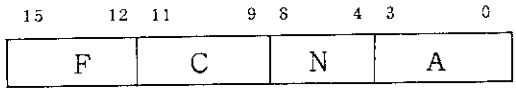
Commandタイプのマイクロプログラムを起動して一連のCAMACコマンドを実行する。このQIOはINIT中もINIT－Modify中もINITの終了後も実行できる。

1個のCAMACコマンドは次のような4ワードからなっている。

| FCNA | |
|---|---|
| BC0／BC1 | データの上位8ビット |
| データの下位16ビット | |
| Q | X |

ここで第1ワードのFCNAはMBDのBARにロードするデータで次のようなビット配置にな

っている。




なおFにはF(8)のビットを除いたものがつめて入れられ，F(8)が0／1にしたがって第2ワードにコントロールコマンドBC0／BC1の上位バイトを入れる。データはWriteコマンドの時は出力データを入れ，Read コマンドの時はCAMACからのデータがマイクロプログラムによって入れられる。Q／Xのフィールドにはコマンド実行結果のQ／Xがセットされた時には1が，セットされなかった時は0が入れられる。

このコマンドブロックはMACROの場合は6.2にのべるマクロ命令PFCNAによって，FORTRAN の場合は5.2にのべるサブルーチンSet__PFCNA によって作られ，結果はサブルーチンGet__QX によってえられるので4ワードのブロックであること以外にブロック内の構造は知らなくてもよい。一連のコマンドブロックの後にはデリミッターとして0のワードをつけなければならない。

P1：コマンドブロックの先頭アドレス。DMA領域内でワード境界にアラインされていなければならない。

P2：コマンドの個数

P6：タイムアウトカウント(秒)。マイクロプログラムは全コマンド実行後VAXに対して割込みを行うが，BDDRIVER がチャンネルを起動(CSRにCI ビットをセット)してから指定した時間内に割込みが入らないと，QIOを打切りSS$__TIMEOUTのエラーステータスをI／Oステータスブロックの第1ワードに返す。指定しないとBDDRIVER 中のデフォールト値が用いられる。

エラー時のリターンステータス

SS$__IVCHAN：マイクロプログラムがCommandタイプではない。

SS$__IVADDR：コマンドがDMA領域内にない。

SS$__BUFBYTALI：コマンドがワード境界にアラインされていない。

SS$__BADPARAM：コマンドの個数が0。

SS$__IVBUFLEN ：コマンドの後にデリミッターの0がつけられていない。

### IO$__SETMODE

通常I／Oに関連するASTはQIOのパラメーターとして指定されI／Oの終了時に伝達される。このQIOはI／Oの終了に関係のないイベントの発生をユーザープロセスに伝達するために使用される。QIO自身はイベントの発生と関係なく直ちに終了する。

ファンクションモディファイアー

IO$M_ATTNAST

IO$M_OUTBAND

のどちらかをつけて実行される。

＜IO$M_ATTNAST＞

ＧＬによる割込みが生じた時，マイクロプログラムを起動する際にユーザープロセスのＡＳＴルーチンも起動することを可能にするためのＱＩＯである。ＧＬとマイクロプログラムはIO$_LINK_GLのＱＩＯによってリンクされていなければならない。またプログラムはCIタイプでなくてはならない。ＱＩＯはそのマイクロプログラムがロードされているユニットに対して行う。したがってそのマイクロプログラムにリンクされているすべてのＧＬに対して同じＡＳＴルーチンが起動される。この際ＡＳＴルーチンに渡されるＡＳＴパラメーターの上位１６ビットにはＧＬの番号が入っているので識別が可能である。下位１６ビットにはBDDRIVER 内のステータスワードが入れられるが通常は使用しない。このＡＳＴはターミナルドライバーのアテンションＡＳＴに相当するので同じくアテンションＡＳＴと呼ぶことにする。ただしIO$_SETMODEでＡＳＴを要求しておけば，別のIO$_SETMODEのＱＩＯで取消されるまでＧＬが発生するまで何回でもＡＳＴルーチンが起動される点で異っている。ＡＳＴ伝達のためのコントロールブロック（ＡＣＢ）は各ユニットに対して３個用意してある。したがってＡＳＴが３個未処理でたまった場合はマイクロプログラムは起動されるがＡＳＴの伝達を重ねて行うことはしない。

Ｐ１：ＡＳＴサービスルーチンのアドレス。または０。０を指定するとすでに要求されているＡＳＴルーチンが取消される。

Ｐ３：ＡＳＴ実行のアクセスモード。通常は指定する必要はない。

エラー時のリターンステータス

SS$_IVCHAN：そのユニットにロードされているマイクロプログラムがＣＩタイプでない。

SS$_ILLIOFUNC　：すでにＡＳＴの伝達が要求されている。（新たに要求する場合は一旦取消さなければならない。）

＜IO$M_OUTBAND＞

LISTタイプのマイクロプログラムを使用してLISTモードのデータ収集を行う時に用いられる。この場合GL１７によってマイクロプログラムが起動されるが，個々のバッファーのRead 要求のＱＩＯをだすわけでないので，マイクロプログラムからバッファーが一杯になったことを知らせる割込みが入った時ユーザープロセスのＡＳＴサービスルーチンが起動されるようにする。ターミナルドライバーのOUT－of －BAND　ASTと同様な機構を用いているので同じファンクションモディファイアーを使用している。（先にのべたアテンションＡＳＴもドライバー内の機構は同じになっている。）

ＡＣＢは３個用意されているのでダブルバッファーを使用したLISTモードの収集では，先にのべたアテンションＡＳＴの場合のようにＡＳＴが伝達されないことはない。ＡＳＴサービスルーチンにはＡＳＴパラメーターとしてバッファーの番号（０または１）が渡される。ＱＩＯはBDA１：に対して行わなければならない。

Ｐ１：ＡＳＴルーチンのアドレスまたは０。０の時はＡＳＴルーチンが取消される。

P 2 ：最初に書かれるバッファーの番号（0または1）。これによってドライバーのデータベース中のUCB$W_BOFFに保持されるバッファーの番号がイニシャライズされる。この番号はマイクロプログラムから割込みが入る度に0／1の切換えが行われる。バッファーのポインターはマイクロプログラム中にも別にあるので矛盾しないようにイニシャライズしなければならない。

P 3 ：AST実行時のアクセスモード。通常は指定する必要はない。

エラー時のリターンステータス

SS$_IVCHAN：ユニットがBDA1：でないか，ユニットにロードされているマイクロプログラムがLISTタイプではない。

SS$_BADPARAM：バッファーの番号が0／1以外である。

SS$_ILLIOFUNC：すでにASTが要求されている。

IO$_Start_Stop

DMAのSTART／STOPを行うためのQIOで，Start_Stopタイプのマイクロプログラムを起動する。STARTのためにはIO$M_STARTUP のモディファイアーをつける。常にCIチャンネル7（BDA14：）に対して実行する。マイクロプログラムの起動の際PDXには，STARTの場合は1がSTOP の場合は0がロードされる。

＜IO$M_STARTUP＞

DMAをSTART状態にする。このQIOはまた先にIO$_SETMODE．OR．IO$M_ATTNAST のQIOの項でのべたのと同様のASTの伝達を可能にする。あらかじめIO$_LINK_GL のQIOによってStart_Stop タイプのマイクロプログラムとリンクされているGLによる割込みが発生した時，マイクロプログラムを起動してSTOP 動作を行うと共に，このQIOで指定したユーザープロセスのASTルーチンが起動される。ASTルーチンに渡されるASTパラメーターはアテンションASTの項でのべたのと同じである。なおIO$_LINK_GL で指定する，PDXにロードするコードは，QIOによる起動と区別するために0／1以外でなくてはならない。

P 1 ：ASTルーチンのアドレス。

エラー時のリターンステータス

SS$_IVCHAN：ユニットのマイクロプログラムがStart_Stop タイプでない。

SS$_BADPARAM：ASTルーチンのアドレスに誤りがある。

SS$_PROTOCOL ：データ収集モードの設定が行われていないか，終了していない。

＜モディファイアーなし＞

Start_Stop タイプのマイクロプログラムを起動してDMAをSTOP 状態にする。IO$M_STARTUP で要求されたASTルーチンを取消す。

エラー時のリターンステータス

SS$__IVCHAN：ＱＩＯを行ったユニットのマイクロプログラムのタイプがStart__Stop でない。

S$S__PROTOCOL ：収集モードの設定が行われていないか，設定が終了していない。

IO$__READLBLK

READタイプのマイクロプログラムを用いて，ＤＭＡによって指定した領域にデータを読み込む（実際には出力があってもかまわない）ためのＱＩＯである。

2.3でのべたようにこのタイプのマイクロプログラムは一般にＣＩチャンネルとＤＭＡチャンネルの両方を使用する。このＱＩＯによって起動されるのはＣＩチャンネルのマイクロプログラムであるが，ＱＩＯはＤＭＡチャンネルのユニットに対して行わなければならない。これはあとでのべるIO$__Complete__BufferのＱＩＯによって途中まで書き込まれてペンディングになっているバッファーを形式的に完成させる場合に，ＱＩＯをＣＩチャンネルに対して行えるようにするためである。

IO$__READLBLKによって起動されたマイクロプログラムは一般にはＬＡＭをイネィブルする等のデータ取込みの準備を行う。実際のデータの取込みはイネィブルされたＬＡＭによるＧＬが発生して起動されるＤＭＡチャンネルのマイクロプログラムによって行われ，バッファーが一杯になった時マイクロプログラムから行われる割込みによってIO$__READLBLKのＱＩＯが終了する。ＱＩＯが終了するまでに一般には何回かＤＭＡチャンネルのマイクロプログラムが起動される。

Ｐ１：データを読み込む領域の先頭アドレス。ＤＭＡ領域内でワード境界にアラインされていなければならない。このアドレスはUNIBUS ワードアドレスに変換されてＰＤＸにロードされてマイクロプログラムに渡される。バッファーのサイズはマイクロプログラムに対して事前に直接与えられており，通常のIO$__READLBLKの場合のようにＰ２で指定されない。

エラー時のリターンステータス

SS$__PROTOCOL ：収集モードの設定が行われていないか，終了していない。

SS$__IVCHAN：ユニットがＤＭＡチャンネルでないか，マイクロプログラムのタイプがＲＥＡＤでない。

SS$__ACCVIO：ＤＭＡ領域外の領域を指定した。

SS$__BUFBYTALI ： 領域がワード境界にとられていない。

IO$__Complete__Buffer

LISTタイプまたはＲＥＡＤタイプのマイクロプログラムを用いてＤＭＡのデータ収集を行っている場合に，ＤＭＡをSTOP状態にした時一般にはバッファーには途中までデータが入った状態になっている。このＱＩＯの実行によってマイクロプログラムが起動され，指定されたＤＭＡ領域のワード（複数でもよい）にバッファー中のイベント数等のデータを書き込み，

ＶＡＸに対して割込みを行う。IO$_Complete_Buffer のＱＩＯはこの割込みによってではなく，マイクロプログラムを起動した段階で終了している。この割込みによってＬＩＳＴタイプの場合は要求されているＡＳＴルーチンが起動され，ＲＥＡＤタイプの場合はIO$_READLBLKのＱＩＯが終了する。したがってユーザープロセスはマイクロプログラムから返されるイベント数等の情報をIO$_Complete_Buffer のＱＩＯの完了ではなく，ＡＳＴルーチンの起動あるいはIO$_READLBLKのＱＩＯの完了を待って取込まなければならない。ＱＩＯはＣＩチャンネルに対して行う。

Ｐ１：バッファー中のイベント数等，ペンディングとなっているバッファーの状態を示すデータを受けとる領域のアドレス。ＤＭＡ領域内でワード境界にアラインされていなければならない。このアドレスはBDDRIVERによってUNIBUS ワードアドレスに変換されて，ＰＤＸにロードされマイクロプログラムにわたされる。ＲＥＡＤタイプのマイクロプログラムは通常このデータによってIO$_Complete_BufferとIO$_READLBLKを区別する。

エラー時のリターンステータス

SS$_PROTOCOL ：収集モードの設定が行われていないか，終了していない。

SS$_IVCHAN：ユニットがＣＩチャンネルでないか，マイクロプログラムがＬＩＳＴタイプでもＲＥＡＤタイプでもない。

SS$_DEVACTIVE ：ＤＭＡがSTOP状態でない。

SS$_IVADDR：Ｐ１で指定したアドレスがＤＭＡ領域内にない。

SS$_BUFBYTALI ：Ｐ１で指定したアドレスがワード境界にない。

### IO$_CI

ＣＩタイプのマイクロプログラムを起動するためのＱＩＯである。ＱＩＯはマイクロプログラムがＶＡＸに対して行う割込みによって終了する。

Ｐ１：起動時にＰＤＸにロードするデータ。

Ｐ６：タイムアウトカウント秒。起動後指定した時間が過ぎてもマイクロプログラムから割込みが入らない場合，Ｉ／Ｏステータスブロックの第１ワードにSS$_TIMEＯＵＴのエラーステータスを入れてＱＩＯを打切る。指定しないとBDDRIVER内のデフォールト値が用いられる。

エラー時のリターンステータス

SS$_PROTOCOL ：収集モードの設定が行われていないか，終了していない。

SS$_IVCHAN：ユニットがＣＩチャンネルでないか，マイクロプログラムがＣＩタイプでない。

### IO$_DIAGNOSE

このＱＩＯは他のＱＩＯと異り，ＭＢＤの診断のためにのみ用いられる。DIAGNOSE特権

がないプロセスは使用できない。

IO$M_WRITEDEV

IO$M_READDEV

のどちらかのモディファイアーをつけて実行される。ユニットは関係ないのでBDA0:に対して行うべきである。MBDの4つのレジスター，CSR，PDX，MSK，IR（Writeのみ）の1つに対してRead／Writeを行う。

P1：データのアドレス（DMA領域でなくてもよい）。

P2：Read／Write を行うレジスターのアドレスのCSRに対するオフセット(0－6)。

エラー時のリターンステータス

SS$_NOPRIV：DIAGNOSE 特権がない。

SS$_ACCVIO：P1で指定したアドレスがアクセスできない。

SS$_BADPARAM：P2のレジスターオフセットが誤り。

## 4. マイクロプログラムの役割とユーザープロセスとの関係

各タイプのマイクロプログラムについては，すでに2.3でのべ，3.でもQIOに関連してのべている。多少これまでの記述と重複するが，ここで各マイクロプログラムが守らなければならない事項をくわしくのべる。また，ユーザープロセスとの関連についてものべる。

(1) LISTタイプ

ダブルバッファーを使用したLISTモードのデータ収集を，個々のバッファーに対するRead要求のQIOを行うことなく，連続して行うため，マイクロプログラムが守らなければならない約束が多い。

まずこのプログラムはBDA1：（DMAチャンネル0）にロードされるが，同時にCIチャンネルのBDA0：も使用する。したがってプログラム内には両方のチャンネルに対する別々のエントリーポイントがある。したがってファイルレジスターのロードは両方のユニットに対して行い，それぞれのCTRがそれぞれのエントリーポイントを指すようにしなければならない。DMAチャンネルを起動するGL17は基本的にはStart_Stop タイプのマイクロプログラムによってイネィブル／ディスエイブルされる。DMAチャンネルのプログラムはCAMACからデータを読みDMAでLISTバッファーに書き込む。バッファーが一杯になるまでは書き込み後単にEXITするだけである。バッファーが一杯になった時はVAXに対して割込みを行いユーザープロセスにASTルーチンを通じて知らせると共にバッファーの切換えを行う。バッファーを切換えるためにはユーザープロセスが次のバッファーの処理（通常はMTへのダンプ）を終えているかどうかチェックしなければならない。このためにはDMA領域中にフラグを置く必要がある。バッファーが一杯になるとマイクロプログラムがフラグをセットし，ユーザー

がないプロセスは使用できない。

IO$M_WRITEDEV

IO$M_READDEV

のどちらかのモディファイアーをつけて実行される。ユニットは関係ないのでBDA0:に対して行うべきである。MBDの4つのレジスター，CSR，PDX，MSK，IR（Writeのみ）の1つに対してRead／Writeを行う。

P1：データのアドレス（DMA領域でなくてもよい）。

P2：Read／Write を行うレジスターのアドレスのCSRに対するオフセット(0－6)。

エラー時のリターンステータス

SS$_NOPRIV：DIAGNOSE 特権がない。

SS$_ACCVIO：P1で指定したアドレスがアクセスできない。

SS$_BADPARAM：P2のレジスターオフセットが誤り。

## 4. マイクロプログラムの役割とユーザープロセスとの関係

各タイプのマイクロプログラムについては，すでに2.3でのべ，3.でもQIOに関連してのべている。多少これまでの記述と重複するが，ここで各マイクロプログラムが守らなければならない事項をくわしくのべる。また，ユーザープロセスとの関連についてものべる。

(1) LISTタイプ

ダブルバッファーを使用したLISTモードのデータ収集を，個々のバッファーに対するRead要求のQIOを行うことなく，連続して行うため，マイクロプログラムが守らなければならない約束が多い。

まずこのプログラムはBDA1：（DMAチャンネル0）にロードされるが，同時にCIチャンネルのBDA0：も使用する。したがってプログラム内には両方のチャンネルに対する別々のエントリーポイントがある。したがってファイルレジスターのロードは両方のユニットに対して行い，それぞれのCTRがそれぞれのエントリーポイントを指すようにしなければならない。DMAチャンネルを起動するGL17は基本的にはStart_Stop タイプのマイクロプログラムによってイネィブル／ディスエイブルされる。DMAチャンネルのプログラムはCAMACからデータを読みDMAでLISTバッファーに書き込む。バッファーが一杯になるまでは書き込み後単にEXITするだけである。バッファーが一杯になった時はVAXに対して割込みを行いユーザープロセスにASTルーチンを通じて知らせると共にバッファーの切換えを行う。バッファーを切換えるためにはユーザープロセスが次のバッファーの処理（通常はMTへのダンプ）を終えているかどうかチェックしなければならない。このためにはDMA領域中にフラグを置く必要がある。バッファーが一杯になるとマイクロプログラムがフラグをセットし，ユーザー

プロセスはそのバッファーの処理が終るとフラグをリセットする。マイクロプログラムは次のバッファーのフラグがリセットされてからＶＡＸに対して割込みを行う。フラグがリセットされていない時はマイクロプログラムはフラグをチェックしながら待つ。しかしこれによって他のチャンネルの起動がブロックされてしまうので，マイクロプログラムはブランチデマンド（ＢＤ）があるかどうかをチェックし，ある場合には

CON　EX1

のＥＸＩＴコマンドを実行して，他のチャンネルに実行権を一時的に明け渡す。またＢＤがない場合でも一定の時間が経過したら同じＥＸＩＴを行う。これによって待たされているＣＩチャンネルの起動が行われる。他のチャンネルの要求がなくなれば，ＣＴＲにセーヴしておいたアドレスからＬＩＳＴタイプのプログラムが再開される。この間，このプログラムを起動するＬＡＭはディスエィブルしておく。最終的にフラグがリセットされた時マイクロプログラムは再びＬＡＭをイネィブルしてＶＡＸに対して割込みを行う。このＬＡＭのイネィブルはStart__Stop のマイクロプログラムが行う，ＳＴＯＰ動作を破るものであってはならない。したがってＳＴＯＰ動作がこのＬＡＭをディスエィブルするCAMAC コマンドである場合には，ＭＢＤのメモリー中にもＤＭＡのＳＴＡＲＴ／ＳＴＯＰ状態を示すフラグを置かなければならない。現在のシステムではＳＴＡＲＴ／ＳＴＯＰの動作は別のGATE 信号をコントロールすることによって行っているのでこの必要はない。ＬＩＳＴタイプのマイクロプログラムがチャンネル０を用いるのは，プライオリティーを最も低くして，必要に応じて他のチャンネルの実行を許すためである。

測定のなんらかの区切でＤＭＡをＳＴＯＰした時，一般にはバッファー中には途中までデータが入っている。ユーザープロセスはこのバッファーの状態を知るためにＩＯ$__Complete__Buffer のＱＩＯ を行う。これによってＬＩＳＴタイプのマイクロプログラムがＣＩ チャンネルのエントリーポイントから開始される。マイクロプログラムは指定されたＤＭＡ領域（UNI－BUS ワードアドレスとしてＰＤＸに入れられる）にバッファー中のイベント数等の情報を書き込んでから，バッファーが一杯になった時の処理を行いＶＡＸに対して割込みを行う。この際必要なら実際のデータと区別できるようなダミーデータをバッファーの残りに書き込む。

テスト用に作成されたリストタイプのマイクロプログラムは 6.4 にあげてある。

(2)　ＲＥＡＤタイプ

ＩＯ$__ＲＥＡＤＬＢＬＫ のＱＩＯによって指定されたＤＭＡ領域にCAMACからデータをとり込むためのプログラムであり，ＬＩＳＴモードのデータ収集にも使用できるようになっている。このためＬＩＳＴタイプの場合と同様ＣＩ チャンネルとＤＭＡチャンネルの両方を使用するようになっている。しかしＬＩＳＴタイプと異なりチャンネルは０である必要はない。ＤＭＡチャンネルがＧＬによって直接起動される点とＩＯ$__Complete__Buffer のＱＩＯに対するレスポンスは同様であるが，まず最初にＩＯ$__ＲＥＡＤＬＢＬＫ のＱＩＯによってＣＩ チャンネルが起動され，バッファーアドレスの指定を行いＧＬによるＤＭＡチャンネルの起動を可能にする点が異なる。また次のバッファーへの切換はマイクロプログラムが行わず，次のＩＯ$__ＲＥＡＤ－ＬＢＬＫ のＱＩＯによる点が大きく異なっている。このためユーザープロセスとの同期は完全にとることができ，バッファー個数も２個以上とることが可能である。バッファーのアドレス

はUNIBUS ワードアドレスに変換されてＰＤＸを通じてマイクロプログラムに渡されるが，マイクロプログラムはこのアドレスあるいは何らかのフラグによってIO$__READLBLK とIO$__Complete__Buffer のＱＩＯを区別しなければならない。

IO$__READLBLKのＱＩＯ によってイネィブルされたＧＬが発生した時，ＤＭＡチャンネルのマイクロプログラムが起動され指定されたバッファーにデータを書き込む（逆に出力であってもかまわない）。バッファーが一杯になるまではプログラムはただEXIT するだけである。バッファーが一杯になった時はＬＡＭをディスエィブルすると共にＶＡＸに対して割込みを行う。これによってIO$__READLBLK のＱＩＯが終了する。上にのべたＬＡＭのイネィブル／ディスエイブルはＤＭＡのSTART／STOP 状態を変えるものであってはならない。したがって場合によってはＭＢＤのメモリー内にもSTART／STOP状態のフラグを置く必要がある。

IO$__Complete__Bufferについては，これによってペンディングになっているIO$__READLBLK のＱＩＯが終了することを除き，ＬＩＳＴタイプの場合と同様である。

(3) ＰＨＡタイプ

2.3で述べた通り。

(4) Start__Stopタイプ

プライオリティを最も高くするためいつもＣＩ チャンネル７を用いる。IO$__Start__StopのＱＩＯによって起動されＤＭＡのSTART／STOPのコントロールを行うが，IO$__LINK__ＧＬによってリンクされているＧＬによる割込みルーチンの中からも起動されSTOP 動作を行う。ＱＩＯによる場合はSTART／STOPにしたがい１／０が，ＧＬによる場合はIO$__LINK__ＧＬで指定されたコードがＰＤＸにロードされるのでマイクロプログラムはこのコードによって必要な処理を識別する。ＱＩＯ によって起動された時はＶＡＸに対して割込みを行うが，ＧＬによる場合は割込みを行ってはならない。ＧＬの発生はSTART のためのＱＩＯ で指定されたＡＳＴルーチンを起動することによってユーザープロセスに知らされる。このマイクロプログラムによるSTART／STOPのコントロールがＧＬの発生源のＬＡＭのイネィブル／ディスエイブルのCAMACコマンドによる場合は，ＬＩＳＴタイプやREADタイプのマイクロプログラムによって出される同じコマンドと矛盾が起らないように，ＭＢＤのメモリー中に共通のSTART／STOP状態を示すフラグを置く必要がある。現在のシステムではSTART／STOPのコントロールを各CAMAC モジュールに対するＧＡＴＥ 信号をコントロールすることによって行っているのでこの必要はない。

(5) Commandタイプ

すでに作成されているマイクロプログラムを使用すればよい。特別の理由がない限り作り直す必要はないと思われる。

(6) CI タイプ

IO$__CI のQIOによって起動されるマイクロプログラムであり，QIOの第1パラメーターとして指定された値をPDXを通じて受けとり対応する処理を行い，終了後VAXに対して割込みを行う。この割込みによってQIOは終了する。他のマイクロプログラムと矛盾しない限りDMAを含め何を行うかは自由である。例えばあらかじめきめられた一連のスケーラーのデータを読んで指定された領域に書き込むために用いられる。

またIO$__LINK_GLのQIOによってリンクされたGLによる割込みサービスルーチンの中からも起動される。この際は一般にはVAXに対して割込みを行わないが，QIOで起動された際には割込みを行わず，何回かのGLの後VAXに対して割込みを行ってQIOを終了させるというやり方も可能である。1個のQIOに対して1回の割込みが行われるようにすればよい。

# 5. 基本サブルーチン

MBDを使用するためには，3.にのべたQIOと他の必要なシステムサービス要求を行えばよいが，きまった手続きを行いやすくするためにFORTRANから使用できるサブルーチンを作成した。これらはライブラリー

〔DATACQ. LIB〕DATACQ. OLB

に入れられている。これらのサブルーチンは実際にデータ収集システムを作成するにあたって多少手直しされるものもあるかも知れないが，ほとんどはこのまま変更せずに使用できると思われる。

## 5.1 基本サブルーチンが使用するCOMMONブロック

基本サブルーチンは2つのCOMMON ブロック

DMA area と Control___data

を使用している。これら（特に後者）は実際のシステムによって大きく変るものであるが，MBDのコントロールに関する部分は多分変える必要がないと思われる。変更する場合もテキストライブラリー

〔DATACQ. TLB〕DATACQ. TLB

中のモジュールACQCOMMON を変更すれば，すべてこのモジュールをINCLUDE して使用しているので容易に変更できる。

(1) DMA area

次のように定義されている。

```
INTEGER*2 DMA__page, DMA
```

(6) CI タイプ

IO$__CI のQIOによって起動されるマイクロプログラムであり，QIOの第1パラメーターとして指定された値をPDXを通じて受けとり対応する処理を行い，終了後VAXに対して割込みを行う。この割込みによってQIOは終了する。他のマイクロプログラムと矛盾しない限りDMAを含め何を行うかは自由である。例えばあらかじめきめられた一連のスケーラーのデータを読んで指定された領域に書き込むために用いられる。

またIO$__LINK__GLのQIOによってリンクされたGLによる割込みサービスルーチンの中からも起動される。この際は一般にはVAXに対して割込みを行わないが，QIOで起動された際には割込みを行わず，何回かのGLの後VAXに対して割込みを行ってQIOを終了させるというやり方も可能である。1個のQIOに対して1回の割込みが行われるようにすればよい。

# 5. 基本サブルーチン

MBDを使用するためには，3.にのべたQIOと他の必要なシステムサービス要求を行えばよいが，きまった手続きを行いやすくするためにFORTRANから使用できるサブルーチンを作成した。これらはライブラリー

〔DATACQ.LIB〕DATACQ.OLB

に入れられている。これらのサブルーチンは実際にデータ収集システムを作成するにあたって多少手直しされるものもあるかも知れないが，ほとんどはこのまま変更せずに使用できると思われる。

## 5.1 基本サブルーチンが使用するCOMMONブロック

基本サブルーチンは2つのCOMMON ブロック

DMA area と Control___data

を使用している。これら（特に後者）は実際のシステムによって大きく変るものであるが，MBDのコントロールに関する部分は多分変える必要がないと思われる。変更する場合もテキストライブラリー

〔DATACQ.TLB〕DATACQ.TLB

中のモジュールACQCOMMON を変更すれば，すべてこのモジュールをINCLUDE して使用しているので容易に変更できる。

(1) DMA area

次のように定義されている。

```
INTEGER*2 DMA__page, DMA
```

```
PARAMETER  (DMA__page=256)
COMMON  /DMAarea/DMA(0:255, 0:DMA__page-1)
```

これによって256ページのDMA領域を定義している。基本サブルーチンではこの領域にだけMBDからのDMAを可能にしている。あとにのべるサブルーチンMBD__INITを呼ぶことによって，この領域はグローバルセクションDMA__area 上にとられる。

プログラムのリンクの際には，この領域をページ境界にアラインするために例えば次のようなリンクコマンドを実行する。

```
$LINK  TESTMBD, . . . , 〔DATACQ.LIB〕DATACQ/LIB, -
        SYS$INPUT:/OPTIONS
CLUSTER=DMAarea
COLLECT=DMAarea, DMAarea
PSECT__ATTR=DMAarea, PAGE
```

(2) Control__data

COMMON 中の将来も変える必要のない変数についてだけのべる。

INTEGER*4  Start__UNIBUS__address

サブルーチンMBD__INIT によって，DMA領域の先頭に対応するUNIBUS アドレスが入れられる。

INTEGER*4  Mapped__size

MBD__INIT によって実際にUNIBUS にマップされた領域のバイト数が入れられる。必要なUNIBUS マップレジスターが確保されていればこの値は

DMA__page *512

となっているはずである。このサイズを越えてDMAを行ってはならない。

INTEGER*2  MBD__channel (0:15)

サブルーチンMBD__INITでSYS$ASSIGN が，BDA0:-BDA15:に対して行われ，割当てられたI/O チャンネル番号（MBDのチャンネルではない）がここに入れられる。インデックスはMBDのユニット番号である。以後すべてのQIOはこのI/Oチャンネル番号を指定して行われる。

INTEGER*2  Command__channel

Command タイプのマイクロプログラムがロードされているユニットのI/Oチャンネル番号。Command タイプのプログラムをロードした後対応するI/O チャンネル番号を入れなければならない。

INTEGER*2  FREG( 7, 0:15)

ファイルレジスターにロードするデータのテーブルである。第1インデックスがレジスターの番号でILR, DAR, WCR, CCR, CTR, GP1, GP2の順であるがサブルーチンLoad__file__register の項にのべるように通常この順序を意識する必要はない。第2インデックスはMBDのユニット番号である。

## 5.2 各サブルーチンの使用法

以下にのべるサブルーチンの多くは3.でのべたＱＩＯを実行する。ＱＩＯは特にのべられていない限りすべてWait 付のＳＹＳ$ＱＩＯＷ によって行われる。この際イベントフラグは０が，Ｉ／Ｏ ステータスブロックはＩＯＳＴ_Ｌ が指定される。システムサービスのリターンステータスはＳＹＳ_ＳＴＡＴＵＳに入れられる。これらはCOMMONブロックとして

```
integer *4    SYS_STATUS, IOST_L(2)
integer *2    IOST_W(4)
common /COMSS/SYS_STATUS, IOST_L
equivalence   (IOST_L, IOST_W)
```

のように定義されている。

一部のサブルーチンはWait なしのＳＹＳ$ＱＩＯ を行う。この場合はサブルーチンのアーギュメントとしてイベントフラグの番号と，Ｉ／Ｏ ステータスブロックを指定するようになっている。もしどちらかを指定しない場合はイベントフラグ０またはＩＯＳＴ_Ｌ が使用されＱＩＯはWait 付のＳＹＳ$ＱＩＯＷ によって行われる。リターンステータスは上と同様にＳＹＳ_STATUS に入れられ，エラーはターミナルに表示される。

なお以下で示されるサブルーチンのアーギュメントは特にのべないかぎりＩＮＴＥＧＥＲ＊２のタイプである。

### MBD_INIT

呼び方

```
CALL MBD_INIT
```

最初に呼ばれるサブルーチンで次のステップを実行する。

1. 永久メールボックスMBD$_Mailbox をASSIGNし，ASTルーチン付でWait なしのRead 要求を行う。以後ＢＤＤＲＩＶＥＲ がこのメールボックスにメッセージを送るたびに自動的にターミナルにメッセージが出力され，再びRead 要求が行われる。メッセージは次のような形で示される。
   ＜ＢＤＤＲＩＶＥＲ＞：＊＊＊Unexpected interrupt ....index＝１７＊＊＊
2. ＢＤＡ０：－ＢＤＡ１５：をＡＳＳＩＧＮし，割当てられたＩ／Ｏチャンネル番号をＭＢＤ_channel に入れる。
3. グローバルセクションDMA_area を作成しＤＭＡ領域にＭＡＰする。
4. ＩＯ$_Attach のＱＩＯを実行する。この際ＤＭＡ領域のアドレスとサイズ，ＭＢＤ$_Mailbox のユニット番号を指定する。ＱＩＯ によって返される，ＤＭＡ領域の先頭UNIBUS アドレスとマップされたバイト数をStart_UNIBUS_address とMapped_size に入れる。

Detach__MBD

呼び方

CALL Detach__MBD

ＭＢＤの使用をやめるために最後に呼ばれる。これによってIO$__DetachのＱＩＯが実行され，またMBDのすべてのユニットのASSIGNが解除される。このサブルーチンを呼ぶ時にはＤＭＡはSTOP状態でなくてはならず，また終了していないＱＩＯがあってはならない。

INIT__Begin

呼び方

CALL INIT__Begin

IO$__INIT.OR.IO$M_INIT_BeginのＱＩＯを行って，収集モードをすべてリセットして，新たな収集モードの設定を開始する（ＩＮＩＴ中となる）。この設定はサブルーチンINIT__Endが呼ばれるまでの間に行われる。

INIT__Modify

呼び方

CALL INIT__Modify

IO$__INIT.OR.IO$M_INIT_ModifyのＱＩＯを行って，すでに設定がすんでいる収集モードの変更を行う（INIT−Modify 中となる）。この変更はサブルーチンINIT__Endが呼ばれるまでの間に行われる。マイクロプログラムのロードやＧＬとマイクロプログラムのリンクは行えない。

Load__MBD__memory

呼び方

CALL Load__MBD__memory（MBD__Unit，Program， size， MBD__address，
MPRO__type ）

マイクロプログラムをＭＢＤのユニットにロードするサブルーチンである。ＩＮＩＴ中でのみ使用できる。

MBD__UNIT ： ＭＢＤのユニット番号（０−１５）。

Program ：マイクロプログラムが入れられている配列。通常はMACROで書かれたサブルーチンからこの配列のアドレスを受けとるので，受けとったロングワードに%valをつけて指定する。

SIZE：マイクロプログラムのワードサイズ。

MBD__address ：ロードするＭＢＤのアドレス。

MPRO__type ：マイクロプログラムのタイプ。MPRO__type $K__LIST のようにコードで指定する。

### Load__file__register

呼び方

CALL Load__file__register (MBD__Unit , mask )

MBDの1つのユニットのファイルレジスターをロードする。INIT中またはINIT−Modify中でのみ使用できる。

MBD__UNIT ：MBDのユニット番号（0−15）。

mask ：各レジスターに対するビットマスクで対応するビットが1のレジスターだけが書きかえられる。

ロードするデータはCOMMON中のFREGに入れておく。ILR−GP 2 の7個のレジスターの名前はテキストライブラリーDATACQ中のMBDDEFでPARAMETER 文によって1−7の値が割当てられている。したがってレジスターの順序は意識する必要がなく，例えばユニットBDA5 :のDAR とCTRに100と200をロードするためには次のように書けばよい。

```
INCLUDE  '〔DATACQ. TLB〕DATACQ(MBDDEF)'
INCLUDE  '〔DATACQ. TLB〕DATACQ (ACQCOMMON)'
FREG (DAR, 5) =100
FREG (CTR, 5) =200
mask =2**(DAR−1) +2** (CTR−1)
CALL Load__file__register (5, mask)
```

### Read__MBD__memory

呼び方

CALL READ__MBD__memory ( Buffer , size , Start__address )

MBDのメモリーを読むために使用される。INIT中またはINIT−Modify 中でのみ使用できる。

Buffer ：読んだデータを入れる配列。

size ：読むワード数。

Start__address ：読むMBDメモリーの先頭のアドレス。

### Read__file__register

呼び方

CALL Read__file__register (MBD__Unit , Buffer )

MBDの1つのユニットの7個のレジスターを読む。INIT中またはINIT−Modify中でのみ使用できる。

MBD__Unit ：MBDのユニット番号（0−15）。

Buffer ：読んだデータを入れる7ワードの配列。

Link__GL

呼び方

CALL Link__GL (Channel__code )

IO$__LINK_GL のQIOによってGL (1−16) とマイクロプログラムをリンクする。INIT中でのみ使用できる。

Channel__code (2, 16) :

Channel code (1, n)——GL nの割込みが発生した時起動されるCI チャンネルのマイクロプログラムがロードされているチャンネルの番号 (0−7)。使用しないGLに対しては−1を入れる。マイクロプログラムはCI タイプまたはStart__Stop タイプでなくてはならず,またすでにロードされていなければならない。

Channel__code (2, n )——CIチャンネルを起動する際PDXにロードされるデータ。

Prep__Command__pro

呼び方

CALL Prep__Command__pro (Addr__MPRO, size, address , Out__address, MPRO__status )

MBDにロードするためにCommandタイプのマイクロプログラムを用意する。

INTEGER*4 Addr__MPRO :マイクロプログラムが入っている領域のアドレスが入れられる。

size :マイクロプログラムのワード数が入れられる。

address:ロードするMBDのアドレス。

Out__address:指定したaddressに対して,プログラムがMBDメモリーのページ境界にまたがる時はロードアドレスを次のページの先頭に変えてロケーションが行われる。変更されたロードアドレスがこのワードに返される。変更されなかった時は指定したaddressが返される。プログラムをMBDにロードする時はこのOut__addressを指定する。

INTEGER*4 MPRO__status :サブルーチン中でマイクロプログラムが指定したアドレスまたは変更されたロードアドレスからはじまるようにリロケーションが行われるが,この際BCTまたはBCFのブラン命令がページ境界を越えて行われる時は正しくブランチが行われないので,エラーコードSS$__LENVIOがここに返される。正しい時はSS$__NORMALが返される。(現在作成されているCommandタイプのプログラムではSS$__LENVIO が返されることはないが,一般にはエラーコードが返された時はJPコマンドを用いる等プログラムの修正が必要である。)

なおこのサブルーチンによって用意されるCommand タイプのマイクロプログラムは,上のOut__addressをCTRに入れる以外に,他のファイルレジスターに初期値をロードする必要はない。

INIT__End

呼び方

CALL INIT__End

収集モードの設定を終了する。

Link__ Attension __AST

呼び方

CALL Link__Attension__AST (MBD__Unit, AST__routine )

GL(1−16)の割込みが発生して，CIチャンネルのマイクロプログラムが起動される際，ユーザープロセスのASTルーチンも起動されるようにする。

MBD__Unit：マイクロプログラムがロードされているCIチャンネルのユニット番号。GLはサブルーチンLink__GLによってこのマイクロプログラムとリンクされていなければならない。マイクロプログラムはCIタイプでなくてはならない。

AST__routine ：ASTルーチンのサブルーチン名。%val(0)を指定するとASTルーチンが取消される。

ASTルーチンにはロングワードのASTパラメーターが渡される。このパラメーターの上位16ビットにはGLの番号が入れられているので何によってASTルーチンが起動されたか判別することができる。下位16ビットにはBDDRIVER 中のMBDのステータスが入れられているが通常用いられない。ASTパラメーターは%val に相当する渡し方をされるのでFORTRAN では例えば次のようにして受けとる。

```
SUBROUTINE AST ( AST__val )
INTEGER*4 AST__val, AST__PRM
INTEGER*2 AST__PRM__W(2), GL
EQUIVALENCE ( AST__PRM, AST__PRM__W )
AST__PRM=%LOC ( AST__val )
GL=AST__PRM__W(2)
```

Link__Buffer__full__AST

呼び方

CALL Link__Buffer__full__AST (AST__routine, Buffer__no )

LISTタイプのマイクロプログラムを用いてLISTモードのデータ収集を行う場合に，バッファーが一杯になった時に起動されるASTルーチンを指定する。

AST__routine ：ASTルーチンのサブルーチン名。

Buffer__no：最初にデータを入れるLISTバッファーの番号（0または1）。バッファーの番号はマイクロプログラム中とBDDRIVER 中に別にある。マイクロプログラム中の番号はINIT中に通常はファイルレジスターに初期値を与えることで指定する。このサブルーチンによってBDDRIVER中の番号がマイクロプログラム中の番号と一致するようにイニシャライズする（通常は0）。サブルーチンStart__DMAによってデータ

収集が開始される前に呼ばなければならない。

ＡＳＴルーチンにはＡＳＴパラメーターとしてバッファーの番号が渡される。受けとり方はサブルーチンLink__Attension__ＡＳＴの項でのべたやり方に準じて行えばよいが，３２ビットのデータとして受けとる。

### Start__DMA

呼び方

CALL Start__DMA(ＡＳＴ_routine)

Start__Stop タイプのマイクロプログラムを起動してＤＭＡをSTART 状態にする。サブルーチン Link__ＧＬによってこのマイクロプログラムとリンクされているＧＬが発生した時，このマイクロプログラムが起動されてＤＭＡをSTOP 状態にするが，この際ユーザープロセス中のＡＳＴルーチンを起動させることができる。

ＡＳＴ__routine ：ＡＳＴルーチンのサブルーチン名。

ＡＳＴルーチンにはＡＳＴパラメーターの上位１６ビットにＧＬの番号が渡される。受けとり方はサブルーチンLink__Attension__ＡＳＴの項でのべた通りである。

### Stop__DMA

呼び方

CALL Stop__DMA

Start__Stopタイプのマイクロプログラムを起動してＤＭＡをSTOP 状態にする。指定したＡＳＴルーチンは取消される。

### Complete___Buffer

呼び方

CALL Complete___Buffer（MBD__Unit，N__event）

IO$__Complete__BufferのＱＩＯ によって，途中までデータが入っている状態のバッファーに対して，バッファーを形式的に一杯にする処理を行わせる。バッファーの状態を返してもらうためのワードを指定する。

MBD__Unit ：LISTタイプまたはREADタイプのマイクロプログラムがロードされているＭＢＤのユニット番号。ＣＩ チャンネルのユニットを指定する。

N__event ：バッファーの状態が入れられるワード。6.4にあげたマイクロプログラムの場合はバッファー中のイベント数が返される。この情報はこのサブルーチンからのリターン時には入れられていることが保証されない。このサブルーチンによって起動されたマイクロプログラムからの割込みによって，バッファーが一杯になったことを示すＡＳＴルーチンが起動されるがIO$__READLBLKのＱＩＯが終了した時点でこの情報を読まなければならない。

### CAMAC__Command

呼び方

CALL CAMAC__Command (Command , N[, [evf][, IOSB ]])

Commandタイプのマイクロプログラムを起動して一連のCAMACコマンドを実行する。QIOはWait 付でないので必要ならSYS$WAITFRによって同期をとる。

Command( 4, * ) : 1コマンドあたり4ワードのブロックからなるCAMACコマンドの配列。最後のコマンドの次のワードにはデリミッターとして0を入れなければならない。ブロックのFORMATは3.のIO$__Command の項にのべてあるが，以下にのべるサブルーチンSet__PFCNAとGet__QXを用いれば知らなくてもよい。この配列はDMA領域内でワード境界にアラインされていなければならない。

N : コマンドの個数。この数はQIOでチェックのため使われるだけで，マイクロプログラムには渡されない。

evf : QIOで使用されるイベントフラグの番号。省略するとイベントフラグ0が用いられ，QIOはWait 付で実行される。0を指定しても同じ。

IOSB : QIOで使用される4ワードのI/Oステータスブロック。省略するとCOMMON中のIOST-Lが用いられ，QIOはWait 付で実行される。

### Set__PFCNA

呼び方

CALL Set__PFCNA( F, C, N, A, data, Command )

サブルーチンCAMAC__Commandで使用するコマンドブロックを作成する。

F, C, N, A : コマンドのFCNA。

INTEGER*4 data : Writeコマンドの時に，データを入れる。

Command : 4ワードのコマンドブロック。

### Set__FCNA

呼び方

CALL Set__FCNA ( F, C, N, A, Command )

MBDのBARにロードするワードを作成する。

F, C, N, A : CAMACコマンドのFCNA。

Command : BARにロードするワード。

### Get__QX

呼び方

CALL Get__QX (Command, data , Q, X)

CAMACコマンド実行後，コマンドブロックからデータとQ，Xをとり出す。

Command : 4ワードのコマンドブロック。

INTEGER*4 data : データ。

LOGICAL*1 Q, X ：コマンド実行時にQ，Xがセットされていると，それぞれ .TRUE. になる。セットされていないと.FALSE.。

Trigger__channel

呼び方

CALL Trigger__channel(MBD__Unit, code[,[evf][,[IOSB][,timeout]]])

IO$__CIのQIOによってCIタイプのマイクロプログラムを起動する。QIOはWait付でないのでマイクロプログラムの終了はSYS$WAITFRによって同期をとらなければならない。

MBD__Unit：マイクロプログラムがロードされているMBDのユニット番号（0－14の偶数）。

code ：マイクロプログラム起動時にPDXにロードするデータ。

evf, IOSB ：CAMAC__Commandの項でのべた通り。どちらかが省略されるとQIOはWait 付で実行される。

timeout ：QIOを打切るタイムアウトカウント。秒で指定する。省略するとBDDRIVER のデフォールト値が用いられる。

INTEGER*4 FUNCTION Create__Global__Section

呼び方

STATUS = Create__Global__Section (Name, Start , size )

グローバルセクションを作成しMAPする。

INTEGER*4 STATUS ：システムサービスのリターンステータスが入れられる。

CHARACTER Name：グローバルセクションの名前。

Start：グローバルセクションをMAPするCOMMONブロックの先頭の変数。
COMMONブロックはLINK時にページ境界にアラインしなければならない。

size ：グローバルセクションのバイト数。

INTEGER*4 FUNCTION MAP__Global__Section

呼び方

STATUS =MAP__Global__Section ( Name, Start, size )

変数の意味はCreate__Global__Section と同じである。作成ずみのセクションをMAPする。

## 6. マイクロプログラムの作り方

マイクロプログラムの作成だけは高級言語では行えず，VAXのアセンブラー言語である

LOGICAL*1 Q,X ：コマンド実行時にQ，Xがセットされていると，それぞれ.TRUE.になる。セットされていないと.FALSE.。

Trigger__channel

呼び方

CALL Trigger__channel(MBD__Unit, code[,[evf][,[IOSB][,timeout]]])

IO$__CIのQIOによってCIタイプのマイクロプログラムを起動する。QIOはWait付でないのでマイクロプログラムの終了はSYS$WAITFRによって同期をとらなければならない。

MBD__Unit：マイクロプログラムがロードされているMBDのユニット番号（0－14の偶数）。

code ：マイクロプログラム起動時にPDXにロードするデータ。

evf,IOSB ：CAMAC__Commandの項でのべた通り。どちらかが省略されるとQIOはWait 付で実行される。

timeout ：QIOを打切るタイムアウトカウント。秒で指定する。省略するとBDDRIVER のデフォールト値が用いられる。

INTEGER*4 FUNCTION Create__Global__Section

呼び方

STATUS = Create__Global__Section (Name, Start , size )

グローバルセクションを作成しMAPする。

INTEGER*4 STATUS ：システムサービスのリターンステータスが入れられる。

CHARACTER Name：グローバルセクションの名前。

Start：グローバルセクションをMAPするCOMMONブロックの先頭の変数。
COMMONブロックはLINK時にページ境界にアラインしなければならない。

size ：グローバルセクションのバイト数。

INTEGER*4 FUNCTION MAP__Global__Section

呼び方

STATUS =MAP__Global__Section ( Name, Start, size )

変数の意味はCreate__Global__Section と同じである。作成ずみのセクションをMAPする。

## 6. マイクロプログラムの作り方

マイクロプログラムの作成だけは高級言語では行えず，VAXのアセンブラー言語である

MACRO を使用しなければならない。MBDのインストラクションのアセンブルはMACROのマクロ定義によって行われる。必要なすべてのマクロ定義はマクロライブラリー

〔DATACQ. MBD〕MBDMAC. MLB

中で行われている。マイクロプログラムはMACRO で書かれたサブルーチン中に置かれるが，その前にINSTBD のマクロを置くだけですべてのマクロ定義が行われる。

## 6.1 マイクロプログラムの形式

マイクロプログラムを含むMACROのサブルーチンはVAX／VMS のプロシージャーの形式をとっておりFORTRAN からは

```
CALL Prep__micro__pro ( Addr -MPRO, size, address, Out__address,
    MPRO__status, ・・・・)
```

の形式で呼ばれるように書かなければならない。サブルーチンのはじめの5個のアーギュメントはいつもこの順でなければならない。これらは5.2のサブルーチンPrep__Command__Proの項でのべた通り，マイクロプログラムが入っている配列のアドレス，ワードサイズ，指定するロードアドレス，リロケーションの際変更されたロードアドレスおよびリロケーションのステータスである。6番目以降はマイクロプログラム個有のもので主としてマイクロプログラムに初期値を設定するために用いられる。

サブルーチンは次のような形になっていなければならない。

```
.TITLE Prep__micro__pro
.ENTRY Prep__micro__pro, ^M <レジスターマスク>
INSTBD
RELOCATE__MBD
マイクロプログラム中へのデータの設定等のＶＡＸのインストラクション。
この間に用いられるレジスターでＲ0，Ｒ1以外のものははじめのレジスター
マスクに指定して保存するようにしなければならない。
RET
Start__of__microprogram
マイクロプログラムのコード
End__of__microprogram
.END
```

はじめのINSTBD によって他のマクロ

```
RELOCATE__MBD
Start__of__microprogram
End__of__microprogram
```

も定義される。これらは6.2を説明する。

このプログラムをアセンブルするためには，ファイル名がPrep__micro__pro.MPRとして

$MACRO Prep_micro_pro.MPR+〔DATACQ.MBD〕MBDMAC／LIB
のコマンドを実行すればよい。

## 6.2 MBD−11のためのマクロ

マイクロプログラムを作成する際使用するマクロは基本的にはBiRa社のPDP用のものであり参考文献4にのべられている。マイクロプログラムを作成する人はこの文献を十分に理解しておかなければならない。このマクロをVAXで使用するために多少追加を行った。特にリロケーション（MBDのアドレスでの）が容易に行えるように考慮した。またBCT，BCFのブランチ命令がページ境界を越えている時は，後方へのブランチはアセンブル時にエラーメッセージを出し，前方へのブランチのエラーはリロケーション時にMPRO_status にエラーコードを返すようにした。

ここでは主として追加されたマクロについてのべる。

STA，LDA，LCA，JPA

これらのオペレーションコードはリロケートされるインストラクションST，LD，LCI，JPを絶対アドレスで使用するために用いられる。例えば3000番地のメモリーを読む場合に

LD 3000

と書くと，100番地からロードするようにリロケーションが行われると

LD 3100

になってしまう。この場合に

LDA 3000

と書くと3000はリロケートされない。

OCT octal 1, octal 2, ・・・・

HEX hex 1, hex 2, ・・・・

PDPでは8進数がデフォールトであったが，VAXでは10進がデフォールトになっている。8進や16進でワードのデータを指定するためにOCTとHEXを用意した。それぞれ20個までのデータを指定できる。^O や ^XのRADIXはつけてはならない。

なおMBDのインストラクションはVAXのマクロ定義によってバイナリーに変換されるので，VAXのマクロの規則にしたがって ^Oや ^X等のRADIXをつける時は必ず＜^XABCD＞のように＜＞でくくらなければならない。

RELOC address

マイクロプログラム中のラベル等リロケートされるアドレスをデータとして入れるために用いる。

Start_of_microprogram

必すマイクロプログラムの先頭におかなければならない。このマクロによって次のようなステップが行われる。

① マイクロプログラムのロケーションカウンター$ が0にリセットされる。

② マイクロプログラムの先頭にＶＡＸのラベル

MPRO__HEAD

がつけられる。

③ リロケーションのためのテーブルを置くプログラムセクション

RELOC__BLOCK

が開始され，その先頭にＶＡＸのラベル

RELOC__TABLE：

が置かれる。またリロケーションテーブルのカウンター$REL が0にイニシャライズされる。以後ＳＴ，ＬＤ，ＬＣＩ，ＪＰ，ＢＣＴ，ＢＣＦ，ＲＥＬＯＣ等のリロケーションが必要なインストラクションが現れるたびにこのテーブルに２ワードずつのリロケーションに必要なデータが入れられ，$RELがインクリメントされる。この２ワードの第１ワードにはロケーションカウンター$ が入れられ，第２ワードにはディスティネーションアドレスが入れられる。ＢＣＴ，ＢＣＦ の場合にはフラグとして第２ワードの最上位ビットがセットされる。

End__of__microprogram

マイクロプログラムの最後に必ず置かなければならない。これによって次のステップが行われる。

① 次のロケーションにＶＡＸのラベル

MPRO__END：

が置かれ

Size__of__microprogram ＝$

によってマイクロプログラムのワードサイズが定義される。

② Number__of__RELOC＝$REL

によってリロケートする必要があるインストラクションの個数が定義される。

なおStart__of__microprogram，およびEnd__of__microprogramによって定義されるいろいろな名前はすべて次のRELOCATE__MBD のマクロ内で用いられ，6.1でのべた形式でマイクロプログラムを用意すれば，同じ名前を別の目的で用いない限り全く意識する必要がない。

RELOCATE__MBD

0番地から始まるようにアセンブルされたマイクロプログラムをリロケートして，ロードのために必要なデータを主プログラムに返すために用いられる。6.1で示したようにサブルーチンのはじめにINSTBD の次に置かなければならない。このマクロではＲ0，Ｒ1以外のすべてのレジスターは保存される。次のようなステップを行う。

① 指定されたロードアドレスに対して，プログラムサイズ（Size__of__microprogram）が１つのページ（２５６ワード）内に入らない時はロードアドレスを次のページの先頭

に変更する。最終的なロードアドレスは

offset :. BLKL 1

に入れられる。

② サブルーチン

Relocate__MPRO

をCALL してリロケーションを行う。この際リロケーションのためのテーブル

RELOC__TABLE

が用いられる。またBCT, BCF のブランチ先がページ境界を越えていないかチェックを行う。

③ 6.1にのべた

Addr__MPRO

size

Out__address

MPRO__status

に必要なデータを返す。

PFCNA F, C, N, A, data

IO$__Commandで用いられる4ワードのコマンドブロックを作成する。

ソースオペランドコードの定義

MV, AD等2オペランドのインストラクションで用いられるソースオペランドに対してはオペランドの名前にSをつけた名前を別に定義しソースオペランドとして必要なデータを与えた。したがって例えばVAXのプログラム中で

MV PDR, MAR

のMBDのインストラクションをR0に入れたい時は

MOVW #MV+PDRS+MAR, R0

のように書けばよい。ただしSWS はいつもソースオペランドとしてしか用いられないのでこのまま用いる。SWS はMBDのフロントパネルのスウィッチがManualの時，スウィッチをソースとして用いるという意味である。OFFの場合あるいはスウィッチオプションがついていない場合には0がソースとなる。SWSの代りに0を書いてもよい。

## 6.3 プログラミング上の注意

2.1でのべたようにMBDがDMAやVAXに対する割込みを自由に行えるために，マイクロプログラムが誤っている時にシステムのクラッシュやハングアップが起る可能性がある。単純なミスとしてはUNIBUS をホールドしてリリースするのを忘れたり，GLによる割込みルーチンから起動されたマイクロプログラムがLAMをクリアーしなかったりすることによって容易にシステム的なトラブルが発生する。したがってすでに使用しているプログラムを変更す

るような場合はあまり問題がないが，新しいプログラムをテストする場合には他のユーザーがいない環境で行った方がよい。

ここではＭＢＤの回路の変更やＶＡＸ－１１／７８０の特殊性に起因するプログラミング上の注意を２，３あげておく。

ＧＬによる割込み

ＰＤＰで使用している場合に，ＧＬによる割込みが発生した時，１つのＧＬに対してもう一度割込み要求が出され，しかも２回目の時は割込みベクターアドレスがクリアーされているということがまれに発生した。これを避けるためにFig. 2 に示すような回路の変更を行い，ＧＬによる割込みが受けつけられると，以後のＧＬによる割込みを禁止するようにした。そして割込みルーチンから起動されたマイクロプログラムが

```
MV    0, GLR
```

を実行することによって，再びＧＬによる割込みを可能にするようにした。この処置がＶＡＸの場合必要かどうか不明であるが，ＶＡＸ－１１／７８０の割込み処理が時間がかかること，また割込みルーチンからユーザープロセスのＡＳＴルーチンを起動する際の問題等も考慮してＶＡＸでも同じやり方をとることにした。したがってＧＬによる割込みルーチンによって起動されたマイクロプログラムはＧＬの源であるＬＡＭをクリアーすると共に上のインストラクションを実行して，ＧＬによる割込みの禁止を解除しなければならない。

マイクロプログラムからの割込み要求

マイクロプログラムからホストに対して割込みを行うためにはＰＤＰでは

```
BCT   $, INB
CON   INT
```

を実行し，その後すぐEXITしても問題がなかった。ＶＡＸ－１１／７８０でのテストプログラム実行では，このようにしてマイクロプログラムがEXITした直後に，ＧＬ（１７－２４)が発生してＤＭＡチャンネルのマイクロプログラムが起動され，まだ割込みが受けつけられる以前にＤＭＡを実行するとMachine check のエラーを起しシステムがクラッシュすることが判明した。

これを防ぐためには

```
BCT   $, INB
CON   INT
BCT   $, INB
```

と割込みが受けつけられるのを待ってからEXITすればよいことがわかった。これ以外のやり方があるかも知れないが，さしあたり必ずこのようにプログラムしたければならない。

ＭＢＤ－１１のエラーによる割込み

ＭＢＤはブランチハイウェイのタイムアウトまたはUNIBUS のタイムアウトが発生した時にホストに対して割込みを行う。しかし以前に購入したＭＢＤでは，このエラーステータスを

ＣＳＲにラッチしていなかったので割込みルーチンはどちらのエラーが生じたか判定ができなかった。このためFig. 3 のような回路を追加してＣＳＲのエラービットをセットし，このビットのクリアーはMS Ｋレジスターの Read によって行うようにした。

一方最近購入したMB Ｄではこの対策が行われておりブランチハイウェイのタイムアウトのビットをＣＳＲにラッチして，クリアーはＣＳＲのRead によって行われるようにしてある。このためBDDRIVER ではどちらの場合も問題がないようにＣＳＲのビットをチェックした後MS Ｋレジスターをテストするダミーの命令を入れてある。この問題はユーザーのプログラムには直接関係はない。

シングルサイクルモードでのトラブル

BDDRIVER はINITあるいはINIT－Modify 中MB Ｄをシングルサイクルモードで使用する。この場合Ｉ ＲレジスターにインストラクションをロードすることによってMB Ｄインストラクションを実行するが，次にチャンネルをきりかえるためにＣＳＲにＣＩとチャンネルのビットをセットした時，最後に行われたインストラクションがもう一度実行されることがわかった。これは多分回路の設計上のミスであろうと思われるが現在のところチェックしてない。このような問題を防ぐためにはチャンネルを切換える前に実害のないインストラクションを実行しておけばよい。BDDRIVER では

```
JP    0
```

を実行することにしている。ユーザープログラムではMB Ｄをシングルサイクルモードで使用することはないのでこの問題は直接関係ない。

## 6.4 マイクロプログラムの例

例としてLISTタイプのマイクロプログラムを用意するサブルーチンPrep__LIST__Proのリストを示す。このプログラムはテスト用に用いたプログラムを少し書き直したものである。実際のデータ収集システムでこのまま使えるはずである。

このプログラムは大きく分けて4つの部分から成っている。最初がＶＡＸのサブルーチン部分であり，次がIO$__Complete__BufferのＱＩＯによって起動されるＣＩチャンネルのマイクロプログラム，次にコインシデンスユニットからのＧＬによって起動されるＤＭＡチャンネルのマイクロプログラム，最後にＡＤＣやコインシデンスユニットに対するCAMACコマンドが置かれている。

サブルーチンの呼び方はリストの最初にコメントとして示されている。アーギュメントとして標準の5個の他に，コインシデンスユニットのＣとＮ，ADC を読むCAMACコマンドのテーブルの（VAX)アドレスとＤＭＡチャンネルのマイクロプログラムの先頭（ＭＢＤ）アドレスを主プログラムに返すためのアーギュメントが指定されている。サブルーチンは渡されたＣＮを用いてコインシデンスユニットに対する3個のCAMACコマンドのＣＮのフィールドをセットしている。ＡＤＣのデータを読むためのCAMACコマンドはラベルADC__Command：以下に32ADC分の領域がとられ，各ＡＤＣに対しコマンドと読んだデータとＡＮＤを行うマスクワードが入れられ，最後のＡＤＣの次にはデリミッターとして0を入れるようにしてある。このサブルーチンではこれらのデータをサブルーチン内でセットしないで，この領域のアドレスを主プログラムに返すようにしている。FORTRAN では値で返されたアドレスを直接参照できないので，主プログラムから別のサブルーチンを

CALL Set__ADC (% val ( Addr__ADC__Command ) )

のように呼んで，サブルーチン中でコマンドをセットすればよい。このサブルーチンでは渡された領域を通常のやり方でアクセスできる。

ＤＭＡチャンネルのマイクロプログラムの先頭アドレスはユニット1のＣＴＲにロードされる。ＤＭＡチャンネルのマイクロプログラムはGL17 によって直接起動されるが，バッファー0にデータを入れる場合とバッファー1にデータを入れる場合とで，それぞれエントリーポイントをBuffer__0, Buffer__1と変えている。これはプログラムを単純にするためと実行スピードを多少上げるためであり，プログラム自体は使用するファイルレジスターが異なる以外は全く同じである。プログラムはまず各ＡＤＣのデータを読みマスクワードとＡＮＤを行った後ＤＭＡでバッファーに書き込む。コマンドが0の時は1つのイベントの終了と判断する。なおスピードをあげるためCAMACコマンドの実行とＤＭＡを並行して行っている。このやり方によって1つのＡＤＣあたり4.5μsでデータがとり込まれる。イベントが終了した時に，バッファーが一杯になったかどうか判断し，一杯になっていなければＬＡＭをクリアーしてEXITして次のＧＬを待つ。バッファーが一杯になった時の処置は4.でのべた通りである。ユーザープロセスとの同期のために各バッファーの先頭の2ワードを次のように用いている。

| ＩＤ（下位１６ビット） | |
|---|---|
| フ　ラ　グ | ＩＤ(上位８ビット) |

３バイト長のＩＤはＭＴにダンプする際ユーザープロセスが書き込む。マイクロプログラムはバッファーが一杯になった時フラグバイトの最上位ビットをセットし，ユーザープロセスはＭＴへのダンプが終了した時にこのビットをリセットする。次のバッファーのフラグがリセットされていない時はＣＴＲをカウンターとして使用して待ちＣＴＲのビット１２がセットされたら(４０９６回ループしたら) 一旦ＥＸＩＴしている。この時間はもう少し短い方がよいかも知れない。待っている間はコインシデンスユニットのＬＡＭはディスエィブルされている。

なおここではＶＡＸの割込みは次のバッファーの処理が終るまで待っているが，実際には待つのはバッファーの切換えだけでよく，割込みはバッファーが一杯になった時にすぐ行ってもよい。こうすればＭＴへのダンプスピードは少し上がるはずである。ここではＩＯ$__Complete_Buffer のＱＩＯとの関係でやさしいやり方をとっている。

ＣＩチャンネルのマイクロプログラムはまずバッファー中のイベント数を，指定されたＤＭＡ領域のワードに書き込む。このアドレスはUNIBUS ワードアドレスとしてＰＤＲを通じてＡＴＲに入っている。次にバッファー中の残りの領域にダミーデータとして－１を書き込み，ＤＭＡチャンネルのマイクロプログラムへブランチしてバッファーが一杯になった時の処理を行っているづ

以下にマイクロプログラムのリストをあげる。

```
        .TITLE  Prep_LIST_Pro   Prepare LIST micro-program
;*************************************************************
; <LIST.MPR>   Prepare micro program for LIST-mode
;       Y.Tomita  .... 9-JUL-1985
;
;  Calling form:
;       CALL  Prep_LIST_Pro(Addr_MPRO,size,address,Out_addr,
;             MPRO_status,C,N,Addr_ADC_Command,DMA_MPRO_addr)
;               Addr_MPRO: Address(VAX) of the start of
;                          the microprogram(integer*4)
;               size:      Program word size(integer*2)
;               address:   Start MBD address
;                          of the microprogram(integer*2)
;               Out_addr:  Modified start address.(integer*2)
;               MPRO_status:    Relocate status.(integer*4)
;                               SS$_LENVIO,
;                                  if <BCT><BCF> out of range
;               C,N:       Crate and Station #
;                          of the Coincidence Unit.(integer*2)
;               Addr_ADC_Command:   Address(VAX) of CAMAC
;                          commands to read ADCs.(integer*4)
;               DMA_MPRO_addr:  Start address(MBD) of
;                          DMA-channel program.(integer*2)
;
;--

        .ENTRY  Prep_LIST_Pro,^M<>

        INSTBD
        RELOCATE_MBD

        MOVAB   Coincidence_Command,R0  ; ==>Commands for
                                        ;       coincidence unit.
        MOVZWL  #3,R1                   ; Number of commands
10$:    INSV    @24(AP),#13,#3,(R0)     ; set C
        INSV    @28(AP),#4,#5,(R0)      ; Set N
        ADDL    #2,R0
        SOBGTR  R1,10$
        MOVA    ADC_Command,@32(AP)     ; Return the address of
                                        ;    ADC-read commands.
        MOVW    offset,@36(AP)          ; <offset> is defined
                                        ;     in <RELOCATE_MBD>
        ADDW    #Buffer_0,@36(AP)       ; Return the start of
                                        ;  DMA-channel program.
        RET
```

```
;************************************************
;  Micro program for LIST-mode.("LIST" type)
;
;   File registers
;        ILR:  Number of events to fill a buffer
;        DAR:  UNIBUS word address of start of buffer-0
;                                  (do not include ID words)
;        CCR:      "     "       "       "       "   -1   "
;        WCR:  Number of unfilled events
;        GP2:  Buffer pointer
;        CTR:  Pointer for CAMAC comands and mask words.
;        GP1:  Not used.

        .SHOW   MEB

        Start_of_microprogram

;*******************************************************
;*  CI-channel microprogram (IO$_Complete_Buffer)     *
;*******************************************************

        DEP     MAR             ; Address of a word
                                ;  to receive # of events
        CON     BK0             ; Switch to Bank-0 register set.
        MV      WCR,ATR         ; # of unfilled events
        SB      ILR,MDR,WTR     ; Return # of events
                                ;      in the current buffer
        BCT     $,DCB
        MV      WCR,ATR
        BCT     In_wait,ZF      ; In <wait-for-Buffer-ready>
;  Fill "-1"
        MV      GP2,MAR         ; Buffer pointer
        LD      All_Bit
        DEP     MDR             ; Set "-1"
        LCI     ADC             ; Read command and mask
Fill = $
        CON     WTR             ; Write "-1" to LIST-buffer
        INM     CTR,CTR
        INM     CTR,CTR
        MV      MEM,ATR
        BCT     Filled_a_event,ZF       ; End?
        BCT     $,DCB
        INM     MAR,MAR
        JP      Fill
Filled_a_event = $
        BCT     $,DCB
        INM     MAR,MAR         ; Advance pointer
        INM     MAR,MAR         ; Skip delimitter
        DEM     WCR,WCR
        BCT     End_fill,ZF     ; all events end?
        LCI     ADC
        JP      Fill            ; Fill next event
End_fill = $
        MV      GP2,ATR         ; Current pointer
        SB      CCR,ATR         ; in Buffer-1?
        BCT     Full_1,NF       ; Yes
        BCT     Full_1,ZF       ; Yes
        JP      Full_0          ; No, in Buffer-0
```

```
In_wait = $
        MV      GP2,ATR         ; Current pointer
        SB      CCR,ATR         ; in Buffer-1?
        BCT     Wait_1,NF       ; Yes
        BCT     Wait_1,ZF       ; Yes
        JP      Wait_0          ; No, in Buffer-0

All_bit = $
        HEX     FFFF            ; Dummy data


;*********************************************************
;*   DMA-channel microprogram                            *
;*      Triggered by a LAM from the coincidence unit.    *
;*********************************************************

; <<< Entry point for buffer-0 >>>
Buffer_0=$
        MV      GP2,MAR         ; Buffer pointer
        LCI     ADC             ; Read command and mask
        BCT     $,BRB
        MV      MEM,BAR,BC0     ; Read the 1st ADC
        INM     CTR,CTR
Read_0=$
        MV      MEM,ATR         ;MASK
        BCT     $,BRB
        AND     BDL,MDR,WTR     ; Write to LIST-buffer
        INM     CTR,CTR
        MV      MEM,BAR
        BCT     End_event_0,ZF  ; End?
        INM     CTR,CTR,BC0     ; read next
        BCT     $,DCB
        INM     MAR,MAR
        JP      Read_0
End_event_0=$
        BCT     $,DCB
        INM     MAR,MAR         ; delimiter
        DEM     WCR
        BCT     Full_0,ZF       ; Branch, if buffer full
        LD      Clear_LAM
        DEP     BAR,BC1
        LCI     Buffer_0        ; Start address
        BCT     $,BRB
        INM     MAR,GP2,EX2
Full_0=$
        MV      DAR,MAR
        DEM     MAR,MAR,RDH     ; HI ID word for buffer-0
        LD      Full_flag
        BCT     $,DCB
        IOR     MDR,MDR,WTR     ; set Buffer-full flag
        BCT     $,DCB
        MV      CCR,MAR         ; Start of buffer-1's data area
        DEM     MAR,MAR,RDR     ; read HI ID word
        BCT     $,DCB
        MV      MDR,ATR
        BCF     Reset_LAM_0,NF  ; Branch if ready
        LD      Disable_LAM
        DEP     BAR,BC1
        LD      Clear_LAM
        BCT     $,BRB
```

```
        DEP     BAR,BC1
Wait_0 = $
        MV      CCR,MAR
        DEM     MAR,MAR         ; ==>HI ID word
        MV      SWS,CTR         ; loop count
loop_0 = $
        CON     RDR             ; Read HI ID word
        BCT     $,DCB
        MV      MDR,ATR
        BCF     Ready_1,NF      ; Branch, if buffer-1 ready
        BCT     Exit_1_0,BDF    ; Exit-1, if any demands.
        INM     CTR,CTR         ; increment loop count
        BCF     loop_0,C12      ; loop until bit-12 is set.
Exit_1_0 = $
        LCI     Wait_0          ; Restart address
        CON     EX1             ; Temporary exit
Ready_1 = $
        LD      Enable_LAM
        BCT     $,BRB
        DEP     BAR,BC1         ; Reenable LAM
        JP      Fin_0
Reset_LAM_0=$
        LD      Clear_LAM
        BCT     $,BRB
        DEP     BAR,BC1
Fin_0=$
        LCI     Buffer_1        ; Start address
        MV      CCR,GP2,INT
        MV      ILR,WCR         ; Reset unfilled counter
        BCT     $,BRB
        BCT     $,INB
        CON     EX2

; <<< Entry point for buffer-1 >>>

Buffer_1=$
        MV      GP2,MAR         ; Buffer pointer
        LCI     ADC             ; Read command and mask
        MV      MEM,BAR,BC0     ; Read the 1st ADC
        INM     CTR,CTR
Read_1=$
        MV      MEM,ATR         ;MASK
        BCT     $,BRB
        AND     BDL,MDR,WTR     ; Write to LIST-buffer
        INM     CTR,CTR
        MV      MEM,BAR
        BCT     End_event_1,ZF  ; End?
        INM     CTR,CTR,BC0     ; read next
        BCT     $,DCB
        INM     MAR,MAR
        JP      Read_1
End_event_1=$
        BCT     $,DCB
        INM     MAR,MAR         ; delimiter
        DEM     WCR,WCR
        BCT     Full_1,ZF       ; Branch, if buffer full
        LD      Clear_LAM
        DEP     BAR,BC1
        LCI     Buffer_1        ; Start address
        BCT     $,BRB
```

```
        INM     MAR,GP2,EX2
Full_1=$
        MV      CCR,MAR
        DEM     MAR,MAR,RDH     ; HI ID word for buffer-1
        LD      Full_flag
        BCT     $,DCB
        IOR     MDR,MDR,WTR     ; set Buffer-full flag
        BCT     $,DCB
        MV      DAR,MAR         ; Start of buffer-0's data area
        DEM     MAR,MAR,RDR     ; read HI ID word
        BCT     $,DCB
        MV      MDR,ATR
        BCF     Reset_LAM_1,NF  ; Branch if ready
        LD      Disable_LAM
        DEP     BAR,BC1
        LD      Clear_LAM
        BCT     $,BRB
        DEP     BAR,BC1
Wait_1 = $
        MV      DAR,MAR
        DEM     MAR,MAR         ; ==>HI ID word
        MV      SWS,CTR         ; loop count
loop_1 = $
        CON     RDR             ; Read HI ID word
        BCT     $,DCB
        MV      MDR,ATR
        BCF     Ready_0,NF      ; Branch, if buffer-0 ready
        BCT     Exit_1_1,BDF    ; Exit-1, if any demands.
        INM     CTR,CTR         ; increment loop count
        BCF     loop_1,C12      ; loop until bit-12 is set.
Exit_1_1 = $
        LCI     Wait_1          ; Restart address
        CON     EX1             ; Temporary exit
Ready_0 = $
        LD      Enable_LAM
        BCT     $,BRB
        DEP     BAR,BC1         ; Reenable LAM
        JP      Fin_1
Reset_LAM_1 = $
        LD      Clear_LAM
        BCT     $,BRB
        DEP     BAR,BC1
Fin_1 = $
        LCI     Buffer_0        ; Start address
        MV      DAR,GP2,INT
        MV      ILR,WCR         ; Reset unfilled counter
        BCT     $,BRB
        BCT     $,INB
        CON     EX2

Full_flag = $
        HEX     8000            ; Flag for buffer-full
```

```
;*****************************************************
;*     ADC read commands and mask words              *
;*       These are set by the caller after RETURN.   *
;*****************************************************
ADC_Command:
ADC = $
        .NLIST  MEB
        .REPT   32              ; Max 32 ADCs
        FCNA    0,0,0,0         ; Command to read ADC
        DATA    0               ; Mask
        .ENDR
        .LIST   MEB
        DATA    0               ; Delimiter.
                                ;  Must be next to the last ADC.

;*****************************************
;*  Commands for the coincidence unit    *
;*****************************************
Coincidence_Command:
Clear_LAM = $
        FCNA    10,1,0,0
Disable_LAM = $
        FCNA    24,1,0,0
Enable_LAM = $
        FCNA    26,1,0,0

        End_of_microprogram
        .END
```

# 7. BDDRIVERに関するメモ

BDDRIVER はあまり標準的でないドライバーなので，今後の改良や書き換え等のために，内部のロジックやデータ構造について必要と思われることをここでのべておく。BDDRIVERを使用するだけの読者は読まなくてよい。

なお以下の記述を完全に理解するためにはマニュアル

Guide to Writing a Device Driver

だけでは多少不十分で

VAX／VMS Internals and Data Structures

の少くとも6，7章は読んでおいた方がよい。また必要に応じてOSのSYSのリストを参照した方がよい。

BDDRIVER内部におかれるデータ

MBD_STATUS：.BKLW 1

MBDのソフトウェア上のステータスで通常はUCB中に置かれるものである。

__VIELD MBD

によって次のようなビットが定義されている。

ATTACHD IO$__Attachが終了している。

ATTACH_PROG IO$__Attach中。

PAGFLT DMA領域をメモリーにロックするために，ページフォールトの割込みを発生させ，QIOをやり直す。

MAPALLOC UNIBUSマップレジスターがアロケートされている。

START DMAがSTART状態。

START_PROG DMAのSTARTを実行中。

STOP_PROG DMAのSTOP を実行中。

INIT INIT中またはINIT−Modify 中。

MODIFY INIT−Modify 中。

INIT_END INITあるいはINIT−Modifyが完了している。

ADR_UCB：.BLKL 16

MBDの各ユニットのアドレスがAttach時に入れられる。

MBD_DMA_SVAPTE：.BLKL 1

MBD_DMA_BCNT：.BLKL 1

DMA領域のPTEのアドレスとDMA領域のバイト数がAttach 時に入れられる。

Detach時にDMA領域をアンロックするために用いられる。

Mailbox_UCB：.BLKL 1

Attach時に指定された永久メールボックスのUCBのアドレスが入れられる。

INTERRUPT_UCB： .BLKW 24

GL(1－16)による割込み，チャンネルプログラムからの割込みを受けるＵＣＢのアドレス。０ならその割込みは予期していない割込みである。はじめの１６ワードがＧＬのため残りの８ワードがチャンネルプログラムのために用いられる。

INTERRUPT_PDX： .BLKW 24

割込みルーチン中からＣＩチャンネルのマイクロプログラムを起動する際ＰＤＸにロードされるデータ。IO$_LINK_GL のＱＩＯで入れられる。はじめの１６ワードだけが使用されている。

ＵＣＢの拡張領域

UCB$L_MBD_AST： .BLKL 1

特別なＡＳＴ（アテンションとバッファー・フル）のＡＳＴサービスルーチンのアドレス。

UCB$L_MBD_GL_FORK： .BLKL 6

割込みルーチン中からＣＩチャンネルを起動する時に用いられるＦＯＲＫブロック。

UCB$L_MBD_ACB：

特別なＡＳＴのためのＡＣＢ。現在３ブロックの領域がとられている。１ブロックあたり１３ロングワード。

UCB$W_MBD_CI ： .BLKW 1

ＣＳＲにロードするＣＩ コマンド。

UCB$W_MBD_MPRO： .BLKW 1

マイクロプログラムのタイプを示すコード。プログラムがロードされていない時は０。

UCB$W_MBD_GL： .BLKW 1

割込みを行ったＧＬの番号。アテンションＡＳＴのために用いられる。

ＤＭＡ領域のとりあつかい

ＤＭＡのために使用するUNIBUS マップレジスターは，ドライバーのロード時に起動されるユニットイニシャリゼーションルーチンでアロケートする。この時通常はＣＲＢ中の

CRB$L_INTD+VEC$B_NUMREG

のバイトにアロケートされたレジスターの数が入れられるが，バイトフィールドなので２５５個までしか指定できず，BDDRIVER のように２５６個のレジスターをアロケートするとこの値は０になってしまう。このフィールドはデアロケート時に用いられるが，BDDRIVER は永久にアロケートしたままなのでかまわない。したがってドライバー中でこのフィールドを参照してはならない。

IO$_AttachのＱＩＯでＤＭＡ領域をマップするが，通常行われるように最後のレジスターをインバリッドにすることはせず２５６個のレジスター全部をマップするために使用する。ＭＡＰレジスターのロードは標準のシステムサブルーチンでは６４ＫＢまでしか行えないので，サブルーチンIOC$LOADUBAMAPを修正して使用している。

ＤＭＡ領域はＩＯ$__AttachのＱＩＯでメモリー上にロックされるが，通常はＱＩＯの終了時にアンロックされてしまう。このアンロックはＩＲＰ内のIRP$L__ＳＶＡＰＴＥとIRP$L__ＢＣＮＴ を用いて行われるので，メモリーのロックが終ったらこれらをドライバー中のＭＢＤ__ＤＭＡ__ＳＶＡＰＴＥとMBD__DMA__BCNTにセーヴした後クリアーしてしまう。こうすることによってＩＯ$__AttachのＱＩＯ 終了後もＤＭＡ領域はメモリー上にロックされたままになっている。最後にＩＯ$__DetachのＱＩＯ でこれらを再びＩＲＰにロードすることによってアンロックを行う。プログラムが異常終了した時のためにはＩ／Ｏキャンセルルーチンでもすべてのユニットがアサインを解除された時，内部サブルーチンＭＢＤ__ＭＥＭＯＲＹ__ＵＮＬＯＣＫによってメモリーをアンロックする。このサブルーチンはシステムルーチンMMG$ UNLOCKを修正したものである。

ＭＡＰレジスターのいくつかをＵＣＢをマップするようにすれば，マイクロプログラムとドライバー間の同期がとりやすくなるが，マイクロプログラムがエラーを起した場合にシステムクラッシュをひき起す可能性が高くなるので行っていない。

ＩＮＩＴ時のＱＩＯ

ＩＮＩＴ中あるいはＩＮＩＴ−Modify 中に行われるＱＩＯ はＩＯ$__Commandを除きすべてＦＤＴルーチン中で行われ，ＩＲＰ をキューしてＩ／Ｏスタートルーチンで実行を開始するという手順をふんでいない。これは単にプログラミングの手間を多少省いただけ（ＦＤＴルーチン中でのみユーザープロセスの領域を自由に参照できる）であり標準的な書き方に直すことも容易である。しかし，ＭＢＤ__ＳＴＡＴＵＳ中のビットを参照して正しくない順序のＱＩＯを排除しているのでこのままで全く問題がないはずである。このためにＩＮＩＴ中あるいはＩＮＩＴ−Modify 中はCommand タイプのマイクロプログラム以外は起動されないようになっている。実際にはCommand タイプのマイクロプログラムもこの期間中に実行する必要はなく，すべてのマイクロプログラムの起動を禁止した方がすっきりするかも知れない。

ユーザープロセスへのメッセージ伝達

個々のＱＩＯに関連するエラーコードはシステムサービスのリターンステータスまたはＩ／Ｏステータスブロックに入れられてユーザープロセスに返される。しかしＭＢＤは形式的にはＱＩＯがだされていない状態でＩ／Ｏ が行われている場合が多く，別の形のメッセージ伝達が必要となる。このためにＩＯ$__Attach で指定された永久メールボックスに対して，システムルーチンＥＸＥ$ＷＲＴＭＡＩＬＢＯＸ を使用してメッセージを送るやり方をとっている。このルーチンはＩＰＬ$__ＭＡＩＬＢＯＸ 以下のＩＰＬ で呼ばなければならないので割込みルーチンからメッセージを送るためにはＦＯＲＫ プロセスを作成して行っている。このためのＦＯＲＫブロックはＵＣＢを使用できないのでＢＤＤＲＩＶＥＲ 中の

ＭＢＤ__ＥＲＲＯＲ__ＦＫＢ

に３ブロック分の領域をとってある。使用中のＦＫＢを書きかえないためにＦＫＢ$Ｂ__ＦＩＰＬのフィールドが０の時ＦＫＢは使用可能としている。すべてのＦＫＢが使用中の時はメッセージは送られない。

メッセージの定義のためにはマクロ

Mail

が，使用可能なＦＫＢのアドレスを得るためにサブルーチン

Get__Error__FKB

が，メッセージを送るためにはマクロ

Send__Mail

が用意されている。

割込みサービスルーチン

エラー割込み以外の割込みを統一的に処理するために，割込みベクターの順に０－２３の割込みインデックスを定義している。GL（1－16）の割込みに対しては０－１５のインデックスが，チャンネル（０－７）の割込みに対しては１６－２３のインデックスが割当てられている。先にのべたINTERRUPT__UCBとINTERRUPT__PDXはこのインデックスでポイントされる。

ＧＬによる割込みが発生した時，対応するINTERRUPT__UCBにアドレスが入れられているＵＣＢのユニットにロードされているマイクロプログラムが起動される。この際INTER__RUPT__PDX に入っているデータがＰＤＸにロードされる。この起動はＵＣＢ内のUCB$__MBD__GL__FORKの領域をＦＫＢとして使用してFORKプロセスを作成して行う。またUCB中のUCB$__MBD__ＡＳＴにＡＳＴルーチンのアドレスが入れられている時はUCB 中のUCB$__ACB の領域をＦＫＢとしてＦＯＲＫプロセスを作成し，ユーザープロセスのＡＳＴルーチンの起動をキューする。

マイクロプログラムからの割込みに対しては，対応するUCB$__MBD__ＡＳＴが０でなければLISTタイプのマイクロプログラムからの割込みなので上と同じようにＦＯＲＫプロセスを作成してユーザープロセスのＡＳＴルーチンを起動する。そうでない場合はＱＩＯの終了なので通常のＱＩＯの終了処理を行う。

INTERRUPT__UCBに０が入ついてる時は予期していない割込みなので先にのべたやり方でユーザープロセスに対してエラーメッセージを送る。

なお割込みルーチンからＣＩ要求を行うためのＦＫＢはＵＣＢ中に１個だけしか用意されていない。これは6.3でのべたようにＧＬによる割込みが発生したら，マイクロプログラムが起動されて

ＭＶ　　0, ＧＬＲ

を実行するまでＧＬの割込みを禁止していることによって可能になっている。この禁止を行わない場合はその都度システムのノンページドダイナミックプールからＦＫＢをアロケートしてＦＯＲＫプロセスを作成するというやり方をとらなければならない。

特殊なＡＳＴ

上にのべたように，割込みが生じた時対応するＵＣＢのUCB$L__MBD__ＡＳＴが０でない時には，ユーザープロセスのＡＳＴルーチンが起動される。このＡＳＴはＱＩＯの終了に関連

したものでないという意味で特殊である。このためにUCB$L_MBD_ACB の領域がＡＳＴコントロールブロック（ＡＣＢ）として使用される。現在ＵＣＢ毎に３ブロックの領域がとられている。

ＡＳＴのキューのし方はTTDRIVER のOUT_of_BAND ASTとほぼ同様であり，システムルーチンCOM$SETCTRLAST およびCOM$DELCTRLAST と似たコーディングを行っている。まず空きブロックをさがしBUSYビットをセットする。このブロックの後半にはすでにIO$_SETMODE のＱＩＯでＡＳＴのために必要なデータが入れられている。まずブロックの前半をＦＯＲＫブロックとして使用してＦＯＲＫプロセスを作成する。このＦＯＲＫプロセスが起動された時，ブロックの後半に入っているデータを前半に移してＡＣＢのＦＯＲ－ＭＡＴにして，ＡＣＢをキューに入れる。ＡＣＢ中のACB$B_RMOD にはTAST$M_PKASTがセットされており，ユーザープロセスのＡＳＴルーチンが起動される前にピギーバックカーネルモードＡＳＴルーチンが起動され，ＡＣＢのBUSYビットをクリアーする。この時点でＡＣＢ中の必要なデータはスタックに入れられており，ユーザープロセスのＡＳＴルーチンが起動される前に別の割込みが行われてＡＣＢが書き直されても問題ない。

またACB$B_RMODにはTAST$M_NODELETE のビットもセットされており，ASTルーチン起動時にＡＣＢがデアロケートされないようにしている。

空きブロックがない場合にはＡＳＴはキューされないが，ＬＩＳＴタイプのマイクロプログラムからの割込みではこのようなことは起らない。

ＣＩによるマイクロプログラムの起動

ＣＩによってＶＡＸからＣＩチャンネルのマイクロプログラムを起動する際，CSRのREADYビットがセットされるのを待たなければならない。BDDRIVER ではくり返しこのビットをチェックし一定の回数チェックしたらタイムアウトとするやり方をとっている。標準的なやり方ではＲＥＡＤＹでない時はＣＩ要求をキューに入れて，ＣＩによって起動されたマイクロプログラムが終了時に行う割込みを受けて，キュー中のＣＩをとり出して実行するというやり方になる。しかしＶＡＸ－１１／７８０では割込みが受けつけられるのにかなり時間がかかること，さらにＣＩチャンネルのマイクロプログラムはいつも割込みを行うというやり方がとりにくい場合もあることから，センスモードで待つというやり方をとった。このやり方は１つのマイクロプログラムの実行時間があまり長くない限り問題ない。

**参考文献**

1) 菊池，富田，河原崎，大内，竹内，丸山：JAERI－M 9136，原研20MVタンデム加速器データ収集・処理システム（1980）
2) 富田：JAERI－M 9283，20MVタンデム加速器データ収集システムの拡張（1981）
3) Biswell L.R. and R.E.Rajala：LA－5144（1973）
4) BiRa社：MBD－11 Programmer's Manual
5) DEC社：VAX／VMS Guide to Writing a Device Driver
6) DEC社：VAX／VMS Internals and Data Structures

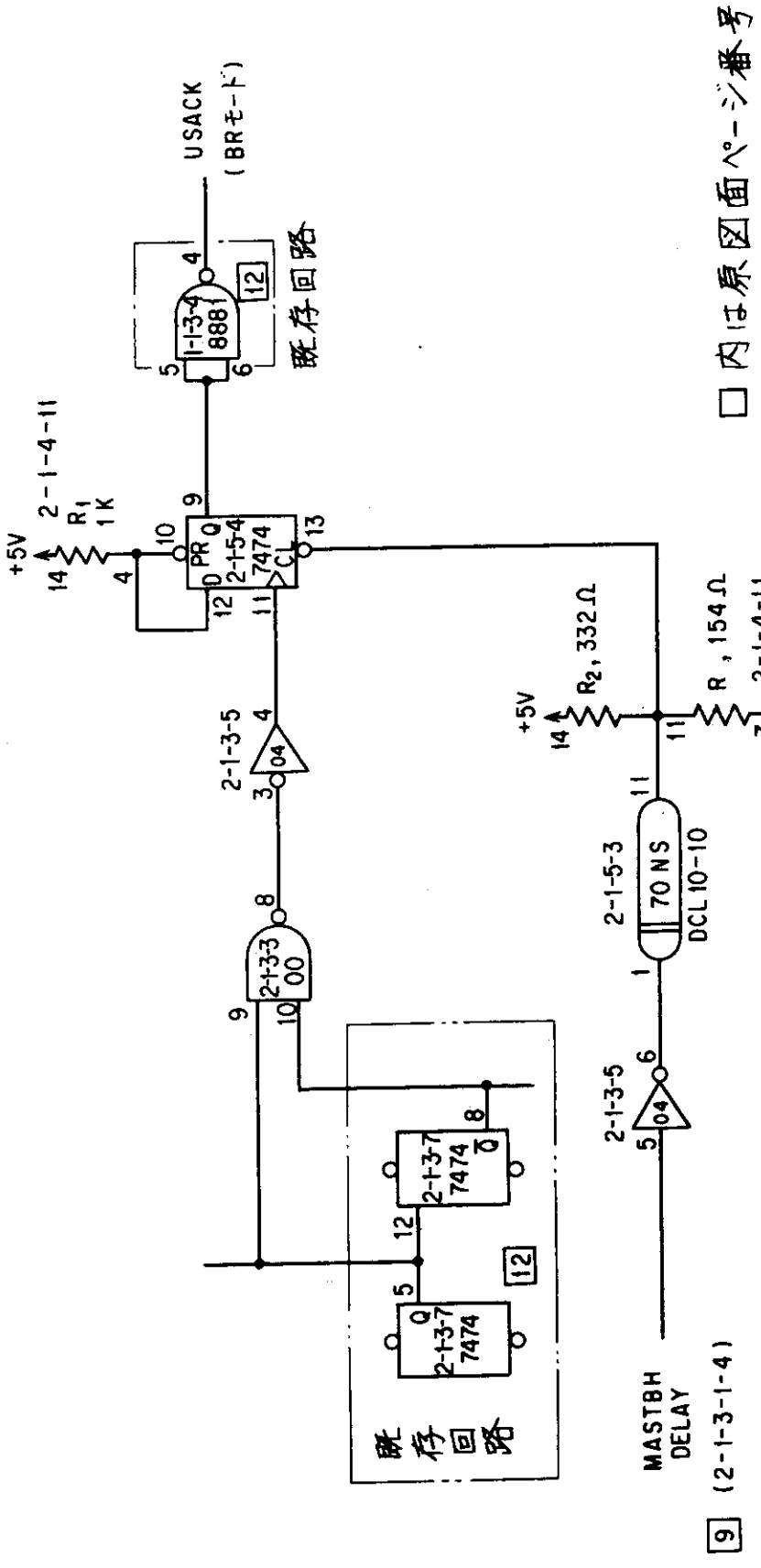



Fig. 1　割込み要求時のタイミングの変更

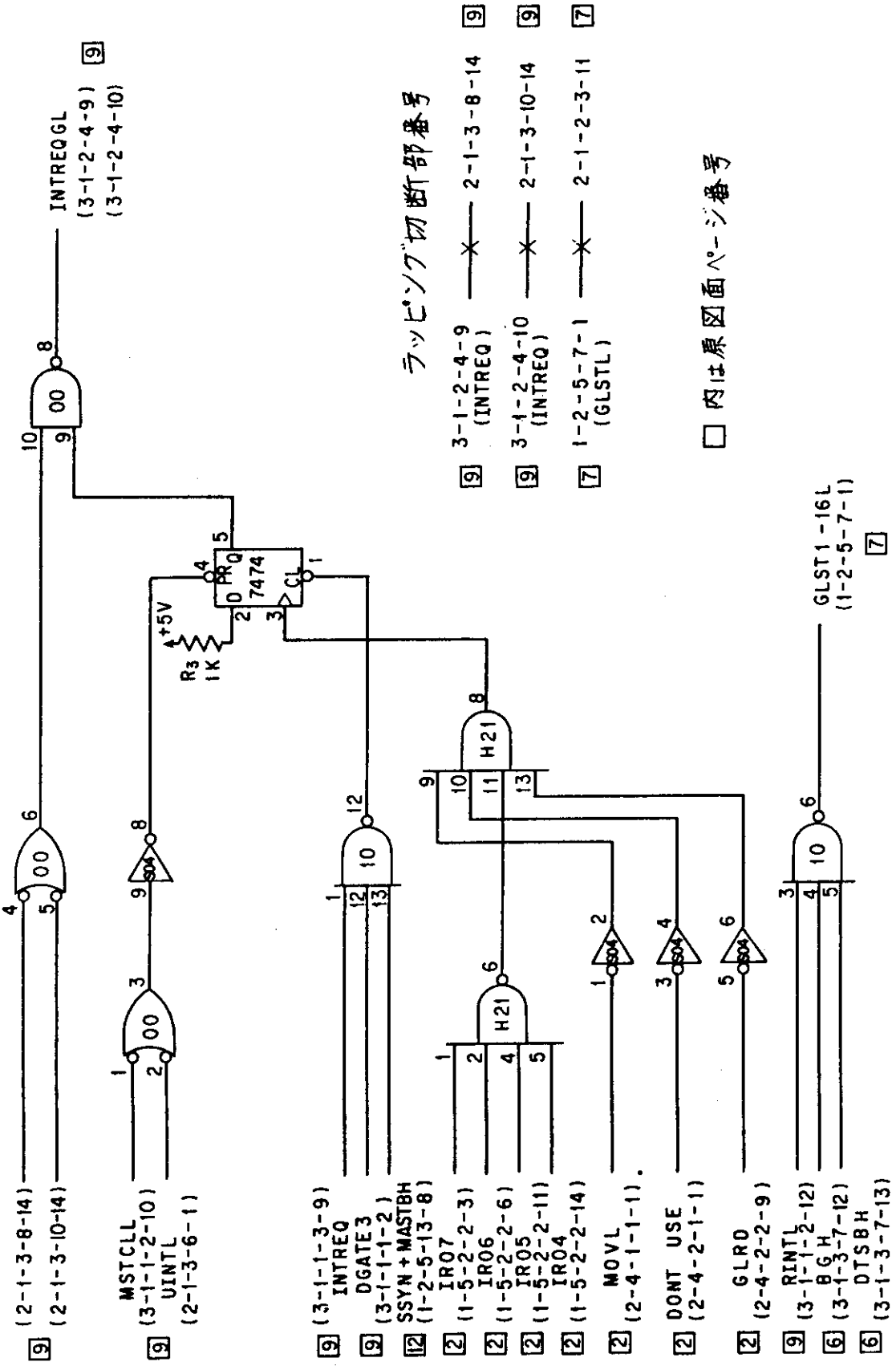



Fig. 2 GLによる割込み要求に関する変更

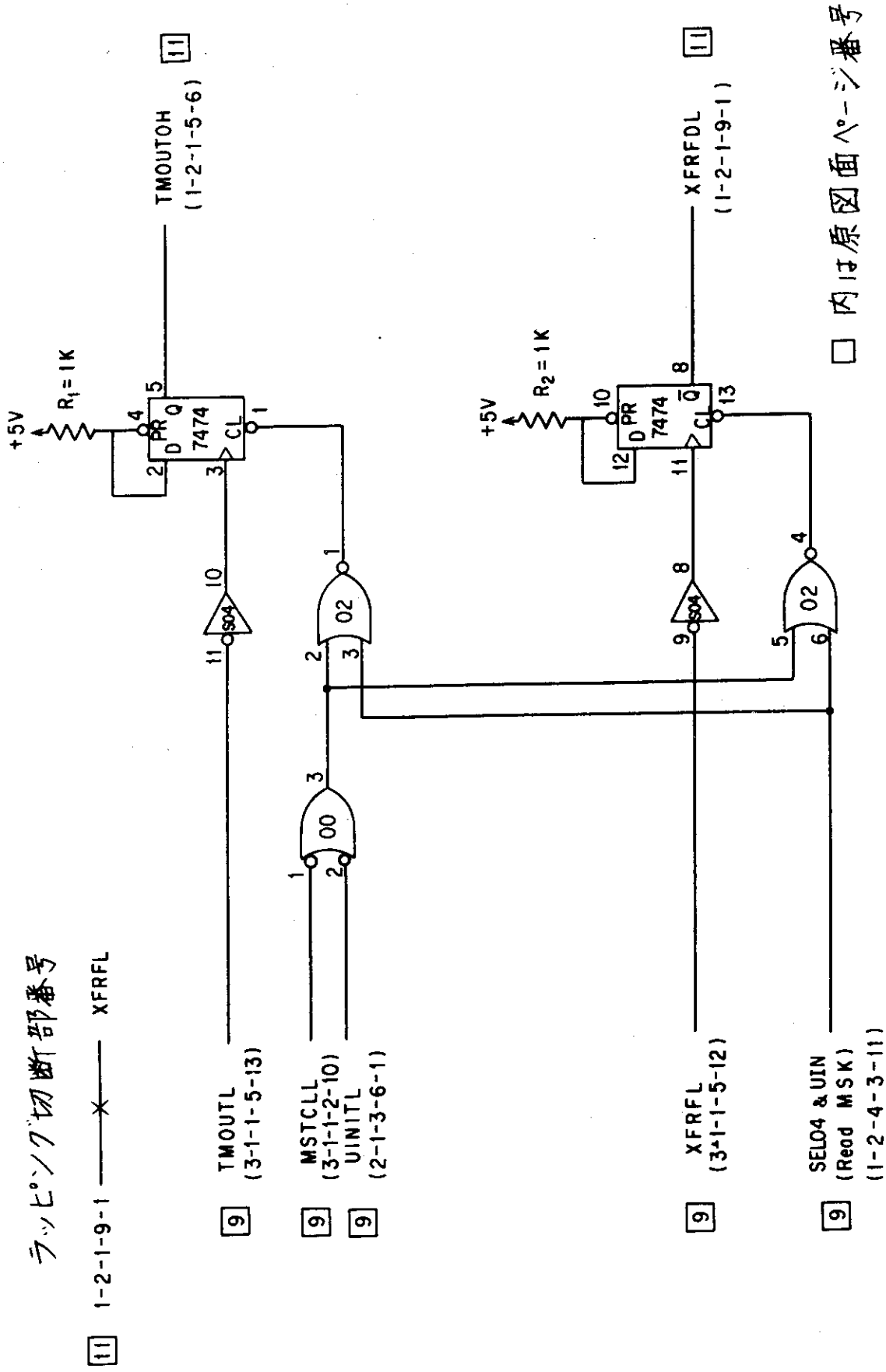



Fig. 3 エラービットのラッチ回路