計算機アーキテクチャ59−15
オペレーティング・システム29−15
(1985・10・25)

並列オペレーティングシステム（ＳＳＳ）における割込み処理の基礎的検討

A Fundamental Study of Interrupt Processing in Parallel Operating System SSS
— Space Sharing System —

曽和　将容，　林　宏也
Masahiro Sowa，　Hiroya Hayashi

群馬大学　工学部
Department of Computer Science, Gunma University

1．まえがき

高性能並列処理の実現をめざして、データフローコンピュータが研究され、その一部が実験機として実現されている。これらのデータフローコンピュータには、ホストコンピュータとしてノイマン式コンピュータが付加されており、記憶管理、処理装置管理、装置管理、情報管理のようなオペレーティングシステムの機能を、このホストコンピュータが行なっている。そして、データフローコンピュータ部は、一種の高速演算装置のような取り扱いをうけている。データフローコンピュータを、真の意味での高性能並列コンピュータとするためには、データフローコンピュータをノイマン式コンピュータから独立させ、データフローコンピュータの上で働くオペレーティングシステムを開発する必要がある。当然のことながら、このオペレーティングシステムは、データ駆動原理の命令により動く、並列データ駆動型オペレーティングシステムでなければならない。

このようなオペレーティングシステムに関しては、Henderson［6］らの関数型オペレーティングシステムや、Arvind［9］らによる資源管理などの研究があるが、現在のところまだほとんど未開発の分野といっても過言ではない。我々の研究室では、１９７９年の第一世代データフローコンピュータの実現化以来、１９８２年の第二世代のデータフローコンピュータの試作と、データフローコンピュータに関する研究を続けてきており、これらの上で働くオペレーティングシステムや言語に関する研究が不可欠の問題となっている。特に、これらソフトウェアのハードウェアに対する影響を調べることが、データフローコンピュータの基本構造を決定するためには、不可欠である。本論文では、我々の研究室で１９８３年以来検討を続けてきた、データフローコンピュータ用のオペレーティングシステムＳＳＳ（space sharing system）のうち、割込みについて基礎的考察を与えると伴に、割込みに関係が深い、入出力機構について述べる。

2．割込み機構の再考

現在ノイマン式コンピュータで行なわれる割込みとして、以下のようなものを上げることができる。

・入出力割込み（終了割込み、故障割込み）
・プログラムエラー割込み
・スーパバイザコール割込み
・外部割込み（ユーザによる割込みボタン押下）

入出力操作を行う場合、オペレーティングシステムによりＩ／Ｏコントローラが起動される。Ｉ／Ｏコントローラは、Ｉ／Ｏコントローラのためのプログラムを実行し、実際に入出力を行う。Ｉ／Ｏコントローラが入出力操作を行っている間、プロセッサは、他の処理を実行することができる。オペレーティングシステムに対し、Ｉ／Ｏコントローラが、入出力操作の（正

常、または異常）終了を知らせるのが入出力割込みである。入出力割込みにより、オペレーティングシステムは、入出力操作が終了し入出力装置が使用可能となったことを知る。このため、入出力割込みが、他のプログラムに対して優先的に処理されないと、オペレーティングシステムは、入出力装置が使用可能となったことを知ることができないので、次に同じ入出力装置を使用する場合、入出力操作を待たねばならない。従って、入出力割込みは、他のプログラムに対して優先的に処理される必要がある。特に、入出力操作が異常終了した場合、オペレーティングシステムは、復帰操作を行い、復帰不可能な場合は、ユーザに通知しなければならないので、優先的に処理されねばならない。

プログラムエラー割込みは、プロセッサが命令を実行したとき、オーバーフローや、0による除算などのエラーが起こった場合に発生する割込みである。データフローコンピュータでは、命令にデータが到着したとき、その命令が実行され、また、実行可能な命令が複数存在した場合、それらは並列に実行されるため、ある1つの命令でエラーが発生した場合、この命令の結果データの到着を期待している命令を実行できず、結局プログラムが停止してしまう危険性がある。従って、プログラムエラー割込みは、他のプログラムに対して優先的に実行される必要がある。エラーが発生した場合、エラーデータを次の命令に渡すことにより、プログラムが停止することを回避する方法が検討されている。（[7]）しかし、この方法では、プログラムが終了しない限りエラーが発生したか否かを知ることができず、エラーが発生した場合、エラーを発生させた命令以降の命令は全て、エラーデータを処理することになるため、期待するプログラムの結果を得ることができない。このエラーデータを処理するためのオーバーヘッドは、大きなものとなる。このため、ここでは、プログラムエラー割込みを利用し、エラーが発生した場合、次の命令にデータを渡すのを待ち、エラー処理を行うものとする。プログラムエラー割込みの結果、回復可能なエラーであれば、回復後次の命令に正常なデータを渡すことができる。しかし、回復不可能なエラーの場合、エラーを発生した命令を持つプログラムを中断しなければならない。データフローコンピュータでは、データによりプログラムが駆動されているため、プログラムを中断するためには、そのプログラムのデータを消去すればよい。ところが、一般にデータフローコンピュータには、データとプログラムを対応づける機構がないため、ある1つのプログラムのデータのみを消去することはできない。このことを可能にするためには、データにタグを付け、データとプログラムを対応づけなければならない。そして、特定のプログラムを中断する場合、このプログラムのタグの付いたデータのみを消去することにより、このプログラムを中断することができる。いづれにしろ、エラー処理は、緊急に処理されなければならない処理の1つである。

スーパバイザコール割込みは、ユーザによりスーパバイザコールが行われたとき発生する。スーパバイザコールは、オペレーティングシステムの特別なプロシジャが、ユーザのプログラムにより呼び出される一種のプロシジャコールと考えることができる。ただし、普通のプロシジャコールと異なり、スーパバイザコール割込みの発生により、このスーパバイザコールがユーザに対して許されているか否かの判定を行い、許されている場合、呼び出されたスーパバイザルーチンが優先して処理されるべきか否かを判定する。優先して処理されるべきスーパバイザルーチンが呼び出されたならば、その処理は、他のプログラムに対して優先的に処理される。このように、スーパバイザ割込みが発生した場合、優先して処理されるべきスーパバイザルーチンが要求されている可能性もあるので、この割込みは、他のプログラムに対して優先的に処理される必要がある。

外部割込みは、ユーザにより割込みボタンが押されたり、内部タイマにより発生する。ユーザへの応答は、すみやかになされるべであるので、ユーザにより割込みボタンが押された場合、この割込みは、他のプログ

ラムに対して優先的に処理されねばならない。また、割込みの結果、プログラムを中断する場合、先の回復不可能なプログラムエラー割込みが発生した場合と同様に、タグ付きデータを導入することで実現できる。内部タイマは、ノイマン式コンピュータにおいて、タイムシェアリングを行うときに、主に用いられている。データフローコンピュータにおいて、タイムシェアリングを行うと仮定すると、タイムスライスの終了（内部タイマにより知ることができる）した、プログラムのデータを全て取り出し、保存しなければならない。タイムスライスの終了したプログラムのデータを、他のプログラムのデータと区別することは、先に述べたタグ付きデータを導入することで実現できる。しかし、データを取り出し、保存するためのオーバーヘッドは、非常に大きくなる。データフローコンピュータでは、時間で処理を分割しなくとも、資源があれば、並列にプログラムを処理できる。従って、データフローコンピュータに、タイムシェアリングを導入することは、あまり大きな意味をみたさない。従って、内部タイマによる割込みは、本質的な時間に対する割込みに限定してもよいようである。

3．データフローコンピュータにおける割込み

3．1　割込みの定義

2章での再考をもとに、データフローコンピュータにおける割込みを次のように定義をする。

定義

任意のプログラムの処理中に、緊急に処理されるべき事象が発生した場合、その事象の処理を優先して行うこと。

ノイマン式コンピュータでは、直列処理のみ可能であるため、同時に複数の処理をすることはできない。従って、割込みが発生した場合、現在処理中のプログラムを中断し、緊急事象に対処しなければならなかった。ところが、データフローコンピュータでは、並列処理が可能であるため、同時に複数の処理をすることができる。従って、割込みが発生した場合、緊急事象を優先的に処理すれば、現在処理中のプログラムを必ずしも中断する必要はない。従って、以上の定義となる。

3．2　割込みの原理

先節でで述べたように、割込み処理は他のプログラムに対し、優先的に実行される必要がある。データフローコンピュータでは、処理に必要なトークンがそろった命令（命令はアクタと呼ばれる）から実行（アクタが実行されることを発火と呼ぶ）される。また、アクタに必要なトークンは、まとめて格納されるている。ここでは、それらをトークンパケットと呼ぶ。アクタの発火は、この必要なトークンの入ったトークンパケットの取り出しによって始まる。従って、割込み処理プログラムを優先するためには、割込み処理プログラムのトークンパケットを優先的に取り出さなければならない。しかし、今までのデータフローコンピュータでは、トークンパケットが、割込み処理プログラムのトークンを持つのか、他のプログラムのトークンを持つのかという区別をしていないため、割込みを他のプログラムに対し、優先的に実行することはできない。そこで、トークンに対し、割込み処理のトークンであるのか、他のプログラムのトークンであるのかを識別するための情報を付加する必要がある。

割込みのトークンが、優先的にアクタに取り込まれ発火できるように、識別情報として、優先順位という情報を導入する。割込みを優先的に処理することは、割込みのトークンに対し、高い優先順位を付けることにより可能となる。優先順位の高い順にトークンが、アクタに取り込まれ発火すると決めれば、他のプログラムよりも優先順位の高い割込みのトークンが、先にアクタに取り込まれ発火する。結果として、割込みは、

優先して実行される。割込み処理が他のプログラムに対し、優先して処理され続けるためには、割込み処理の全てのトークンに対して、高い優先順位を付けなければならない。このために、各アクタは、発火後の出力トークンに対し、入力トークンが持っていたのと同じ優先順位を付ける機能を持たねばならない。各アクタが、このような機能を持つことで、各処理は、その処理が終了するまで、全て同じ優先順位の付いたトークンにより駆動される。

ノイマン式コンピュータにおいて、１つの割込みが発生し、その処理中に他の割込みが発生した場合、後から発生した割込みは、先の処理が終了するまで、その処理を待たねばならない。ところが、データフローコンピュータにおいては、割込みは、高い優先順位をもつトークンをアクタの入力アークに置くのみであり、トークンさえそろえば並列に処理ができる。従って、１つの割込みが発生し、その処理中に他の割込みが発生しても、先の割込みとは関係なく処理することができる。同様に、同時に割込みが発生した場合でも、割込み処理は同時に実行される。

トークンに付けられた優先順位を応用し、割込み以外のプログラムにも、優先順位を付けることができる。基本的にはプログラムは、同じ条件で処理されるが、緊急な処理を要求するプログラムのトークンに対し、他のプログラムよりも高い優先順位を与えることで、このプログラムは、他に優先して処理される。但し、割込み処理の優先順位と等しいか、それを越えることは許されない。これは、オペレーティングシステムのジョブ管理の問題である。この問題を考えるためには、オペレーティングシステムに優先順位を管理するアクタを導入する必要がある。

３．３　割込みの機構

データフローコンピュータは、次の４つの構成要素を持つ。

・ファンクショナルユニット（ＦＵ）
・トークンメモリ（ＴＭ）
・プログラムメモリ（ＰＭ）
・データメモリ（ＤＭ）

ＦＵは、データフローコンピュータのプロセッシングユニットである。ＴＭは、トークンバケットを格納するためのメモリである。ＰＭは、プログラム（アクタの集り）を格納するためのメモリであり、各ＦＵに対して、読み出し専用の共有メモリである。ＤＭは、各ＦＵに対して、読み出し、書き込み可能な共有メモリであり、構造体データのような大量データを格納するためのメモリである。

アクタを発火するのに必要なトークンが全てそろうと、つまり、ＴＭに格納されているトークンバケットが完全になると、ＦＵは、ＴＭよりそのトークンバケットを、また、ＰＭよりそれに対応するアクタを取り込み発火する。最後にＦＵは、アクタを発火した結果のトークンをＴＭへ格納し、このアクタから解放される。この時点で、アクタの実行は終了（鎮火）する。以上が、データフローコンピュータの基本動作である。（［1］，［2］）

トークンに優先順位を付けるために、次の２つの方法が考えられる。

（１）トークンバケットに対し、優先順位を付ける。
（２）既存のＴＭの他に、割込み用のトークンのためのＴＭ（ＩＴＭ－interrupt ＴＭ）を新たに設ける。

（１）の方法では、ＴＭに格納されているトークンバケットに対し、優先順位を示す情報を付け、割込みのトークンを持つトークンバケットに対して、高い優先順位を持たせる。優先順位の高いトークンバケットが、優先的にＦＵに取り込まれアクタを発火する。この方法によれば、割込みの処理間に対して優先順位を付けることも可能であり、またそれ以外のプログラムに対して、優先順位を付けることも可能である。

（２）の方法では、割込み処理のためのトークンは、必ずＩＴＭへ格納され、ＦＵは、まずＩＴＭよりトー

クンパケットを取り込むことを試みる。ＩＴＭに、トークンのそろったトークンパケットが存在したなら、それを取り込みアクタを発火させる。そのうよなトークンパケットが存在しないならば、既存のＴＭよりトークンパケットを取り込むことを試みる。このように、割込みを他のプログラムに対して、優先的に処理することが可能となる。

複数の割込み処理が、同時に実行されているとき、これらの割込み処理の並列度が、データフローコンピュータのＦＵの数以上となった場合、または、データフローコンピュータのＦＵの数以上の割込みが、同時に発生した場合、割込み処理は優先順位に従って実行されなければならない。（１）の方法では、割込みに対して、優先順位を付けることが可能であるので、このような場合、優先順位に従って、割込み処理を実行できる。ところが、（２）の方法では、割込みが、他のプログラムに対して、優先的に処理されるのみで、割込みに対して優先順位を付けることはできない。従って、このような場合に対処するため、割込み用のトークンの間で、優先順位を付けなければならない。つまり、ＩＴＭに格納されるトークンパケットに対して、（１）の方法と同様に、優先順位を示す情報を付けなければならない。

また、並列オペレーティングシステムのジョブ管理機能のことを考慮に入れると、プログラムの実行に関して優先順位を付けることが必要となる。このとき、（１）の方法では、プログラムを駆動するトークンパケット自体に優先順位が付いているため、プログラムの実行に優先順位を付けることは、トークンパケットの優先順位を操作することで、簡単にできる。（２）の方法では、このような考慮がなされていないため、プログラムの実行に関して、優先順位を付けることは不可能である。（２）の方法で、ユーザプログラムの実行に関する優先順位を付けるためには、既存のＴＭに格納されるトークンパケットに対して、（１）の方法と同様に、優先順位を示す情報を付けなければならない。

以上のことから、（２）の方法よりも（１）の方法の方が、合理的であることがわかる。従って、（１）の方法をとり、トークンパケットには、割込みも含めて優先順位を付けることとする。

４．入出力の機構

４．１　入出力の原理

図１に、入出力アクタの、データフローグラフでの表現を示す。入力アクタは、その入力アークにトークンが到着したとき発火する。ここで、トークンは、入力ファイルを示すポインタである。入力アクタは、一度発火すると、データの入力が終了するまで発火しつづける。そして、データの入力が終了すると入力アクタは鎮火し、出力アークにトークンが出される。ここで、出力アークに出されるデータは、入力したデータ、または入力したデータの構造体を示すポインタである。出力アクタは、その入力アークにトークンが到着したとき発火する。ここで、トークンは、出力データ、または出力データの構造体を示すポインタである。出力アクタは、データの出力が終了するまで発火しつづける。そして、データの出力が終了すると、出力アクタは鎮火する。

データフローコンピュータに入出力割込みを導入し、図１に示した入出力アクタを変更する。入力アクタについて変更したものを図２に示す。入出力アクタは、コールアクタ、リターンアクタの２つのアクタとして表現される。このように表現することで、入出力は、入出力を実行するプログラムを呼び出す、一種のプロシジャコールとして考えることできる。（[３]）コールアクタは、その入力アークにトークンが到着したとき発火する。ここで、コールアクタの入力アークに到着するトークンは、先に述べた入出力アクタの入力アークに到着するトークンと同一なものである。コールアクタの発火により、Ｉ／Ｏコントロールプログラ

ムを起動するためのトークンが、Ｉ／Ｏコントロールプログラムへ渡される。この時点で、コールアクタは鎮火し、Ｉ／Ｏコントロールプログラムの実行が開始される。リターンアクタは、その入力アークが、Ｉ／Ｏコントロールプログラムの出力アークと機能的につながっており、Ｉ／Ｏコントロールプログラムからのトークンにより発火する。リターンアクタの発火により、入出力操作の終了状態がチェックされ、入力アクタに対応する場合ならば、出力アークに入力したデータ、またはデータのポインタを出し鎮火する。リターンアクタの入力アークにトークンが置かれることが、入出力割込みに対応する。従って、リターンアクタのトークンは、他のプログラムのトークンよりも優先順位が高い。図１の入出力アクタでは、入出力割込みの概念を導入していないため、入出力アクタが発火すると、入出力操作が終了するまで発火し続けた。つまり、入出力操作が終了するまで、プロセッサは、入出力アクタに専有されてしまった。ところが、図２の考え方では、コールアクタの発火によりＩ／Ｏコントロールプムグラムを開始してしまえば、コールアクタは鎮火し、プロセッサは入出力アクタに専有されることなく、他の処理を実行できる。入出力操作の終了は、Ｉ／Ｏコントロールプログラムから、リターンアクタの入力アークへトークンが渡されることにより、オペレーティングシステムに知らせることができる。

４．２　入出力の構造

図３に、入出力機構を持つデータフローコンピュータの構造を示す。入出力メモリ（ＩＯＭ）は、入出力データバッファとして使用され、また、Ｉ／Ｏコントロールプログラムを格納するメモリである。Ｉ／Ｏコントローラは、入出力専用のプロセッサであり、オペレーティングシステムにより起動されると、ＩＯＭに格納されているＩ／Ｏコントロールプログラムに従って、入出力操作を行う。

入出力アクタを発火させるためのトークンが、ＴＭへ格納されると、ＦＵは、ＴＭよりこのトークンを、ＰＭより対応するアクタを取り込み発火する。ただし、この場合、先に述べたように、実際に発火するのは、Ｉ／Ｏコントロールプログラムにトークンを渡すためのコールアクタである。コールアクタの発火により、ＦＵは、入力トークンの情報に基づき、Ｉ／Ｏコントロールプログラムのためのトークンを ＴＭへ格納する。この時点で、アクタは鎮火し、ＦＵは、他のアクタの発火に移ることができる。一方、Ｉ／Ｏコントローラは、ＴＭにＩ／Ｏコントロールプログラムのためのトークンが、格納されたか否かを常に監視している。そして、Ｉ／Ｏコントロールプログラムのためのトークンが、ＴＭに格納されると、これを取り込みＩ／Ｏコントロールプログラムの実行を開始する。Ｉ／Ｏコントローラは、プログラムに従い、入出力操作を実行し、入出力操作が終了した時点で、先のコールアクタに対応するリターンアクタのためのトークンをＴＭへ格納する。いままで、入出力データの取り込みを、（１）ＩＯＭを入出力データバッファとして、ＴＭとＩ／Ｏコントローラの間で、入出力データのポインタの授受を行うという方法で行ってきてたが、（２）ＴＭとＩ／Ｏコントローラの間で、直接入出力データの授受を行うという方法もある。

データフローコンピュータでは、基本的には、データはＴＭへ格納され、このデータがＦＵに取り込まれアクタを発火する。また、複数データを扱う場合、これらのデータを全てＴＭへ格納することは、コストの面で現実的でないので、構造体データをＤＭへ格納し、そのポインタのみをＴＭへ格納する方法が考えられている。入出力データもこれらと同じように考えると、単一データの場合、（２）の方法が有効であり、複数データの場合、（１）の方法が有効である。

この構造では、ＴＭが、ＦＵとＩ／Ｏコントローラにより共有されている。ＦＵとＩ／Ｏコントローラは、ＴＭに格納されるトークンパケットが、ＦＵにより取り込まれるべきか、Ｉ／Ｏコントローラにより取り込

まれるべきかを区別することができない。従って、トークンバケットに対して、ＦＵ用か、Ｉ／Ｏコントローラ用かを示す情報（proccesor identifire）を設けなければならない。先に導入した優先順位を示す情報と、この情報を付加したトークンバケットの構成を図４に示す。

図３のデータフローコンピュータの構造では、ＴＭとＩ／Ｏコントローラとが接続されており、Ｉ／Ｏコントローラは、ＴＭに格納されたトークンにより起動され、入出力操作の終了をＴＭへトークンを格納することでオペレーティングシステムに知らせた。この構造とは別に、ＦＵとＩ／Ｏコントローラを接続する構造を考えることができる。この構造を図５に示す。この構造では、入出力アクタを発火させたＦＵが、直接Ｉ／Ｏコントローラを起動する。Ｉ／Ｏコントローラを起動したＦＵは、アクタを鎮火し、他のアクタの発火に移る。Ｉ／Ｏコントローラは、入出力操作を終了すると、任意のＦＵに対してリターンアクタのためのトークンを渡す。このとき、Ｉ／ＯコントローラからのトークンをどのＦＵに割り当てるために、割り当て回路として、アービタを導入しなければならない。リターンアクタのトークンを受け取ったＦＵは、リターンアクタを発火する。この構造では、先の構造にはなかったアービタを、新たに導入しなければならない。このため、先の構造に比べ、ハードウェアが複雑になってしまう。先の構造においても、Ｉ／Ｏコントローラからの情報をＴＭへトークンを格納するため、アービタのような割り当て回路を必要とするが、ＴＭが本来この機能を持っているため、新たにアービタを導入する必要はない。また、この構造では、Ｉ／Ｏコントローラから、直接ＦＵにトークンが渡されるため、Ｉ／Ｏコントローラからのトークンは、最も優先して処理されることになる。Ｉ／Ｏコントローラからのトークンよりも高い優先順位を持つトークンが、ＴＭに格納されている場合、Ｉ／Ｏコントローラからのトークンは、ＴＭにあるトークンよりも優先して処理されてしまうので、トークンの優先順位が保たれない。これは、第３章で導入した、トークンに対して優先順位を付けることに矛盾する。

以上より、データフローコンピュータの入出力機構として、図３に示した構造をとることが最適である。

５．むすび

以上、データフローコンピュータにおける割込み機構について考察を行い、その実現方法を検討した。また、割込み制御に関係の深い、入出力機構についても考察した。そして、トークンに優先順位を付けることにより、割込み機構を実現することを提案し、また、入出力機構については、トークンに、一般プログラム用と、入出力用の種類を設けることを提案した。

この方法で、データフローコンピュータ上に割込み機能を実現するには、トークンの優先順位に従って、トークンを取り出すことができるようなハードウェアが必要であり、またトークンの種類に従って、トークンを取り出す機構が必要である。これらの機構のインプリメントは、我々のデータフローコンピュータ（［１］，［２］）に関しては、それほど困難ではない。

本論文では、データフローコンピュータ上で、データフロー言語によるオペレーティングシステムを開発することを目的として、割込みを中心として述べてきた。しかしながら、研究はまだ始まったばかりで、まだまだ、研究しなければならないことが山積みである。今後は、装置管理についてのより詳細な研究を進めると伴に、記憶管理、処理装置管理、情報管理等の研究を進め、完全な並列オペレーティングシステムを構築するつもりである。

文献

[1] M.Sowa and T.Murata,"A data flow computer architecture with program and token memo-

ries,"Proc. of 1980 Asiloman Conf. on Circuites,Systems and Computers,IEEE Computer Society,Nov.1980.

[2] M.Sowa,F.D.Ramos and T.Murata,"Constraction and structure of prototype data flow computer DFNDR-1 and subroutine implementation," IEEE 1st. Int'l Conference on Computers and Applications,pp.480-497,June 1984.

[3] 曽和，Ｆ．Ｄ．ラモス，"データフローコンピュータＤＦＮＤＲにおけるプロシジャコールのインプリメンテーション，" 信学研資，ＥＣ８３－１６，１９８３－７．

[4] Donovan,J.J.,"System programming,"Mcgraw-hill International Book Company,1972.

[5] Masnick,S.E.,"Operating system,"Mcgraw-hill International Book Company,1974.

[6] Henderson,P.,"Purely functional operating system,"Fanctional programming and applications,Cambridge Univercity Press,1982.

[7] Wetherell,C.S.,"Error data values in the data-flow language VAL,"ACM Trans. on Programming Languages and Systems,vol.4,No.2, pp.226-238,April 1982.

[8] Oldehoeft,A.E.,Jennings,S.F.,"Dataflow resource managers and synthesis from open path expressions,"IEEE Trans. on Software Engineering,vol.SE-10,No,3,May 1984.

[9] Arvind and Brock,J.D.,"Resource managers in functional programming,"Journal of Parallel and Distributed Computing,1,pp.5-21,1984.

[10] Joseph,M.,Prasad,V.R.,and Natarajan,N.,"A multiprocessor operating system,"Prentice-Hall,London,1984.

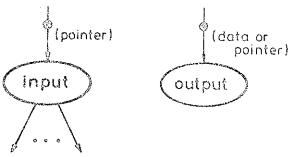

図１　入出力アクタのデータフローグラフ表現

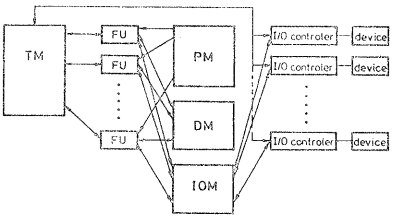

図３　入出力機構を持つデータフローコンピュータの構造

actor name	priority	processor id.	data

図４　トークンパケットの構成

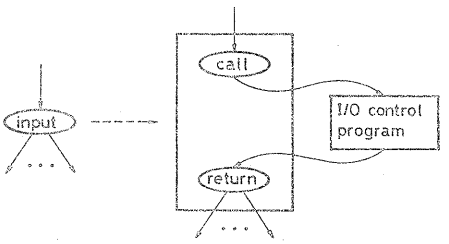

図２　入出力割込みを考慮した入力アクタ

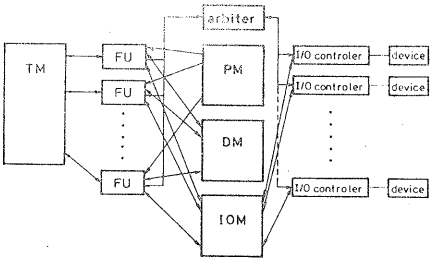

図５　その他の入出力機構を持つデータフローコンピュータの構造