計算機アーキテクチャ 72－1
（１９８８．９．１６）

循環パイプライン計算機　ＦＬＡＴＳ２

市川周一[1]、加藤紀行[1、2]、後藤英一[3、4、1]

１．新技術開発事業団、　２．三井造船システム技研、
３．東京大学理学部、　４．理化学研究所

循環パイプライン方式（ＣＰＡ；Cyclic Pipeline Architecture）を採用した数値・記号処理用計算機FLATS2について報告する。ＣＰＡでは、パイプラインの各ステージを複数の仮想プロセッサに割り当てることにより、ハザードを防いでスループットを最大限に利用することができる。更に、FLATS2ではアドレス・タグ、アドレス範囲検査、命令内分岐を１命令内で併用して、効果的なメモリ管理／配列アクセス機能を提供する（ＢＬスキーマ）。演算パイプラインは、最大４本がチェインして並列に動作し、高い数値処理性能を実現する。
このほか、FLATS2の命令レベル・アーキテクチャを概観し、内部アーキテクチャとパイプライン制御についても簡単に述べる。

FLATS2 : An implementation of the Cyclic Pipeline Architecture

Shuichi ICHIKAWA, Noriyuki KATO, and Eiichi GOTO

Computer Architecture Group, GOTO Quantum Magneto Flux Logic Project,
Research Development Corporation of Japan,
C/O Mitsui Zosen Systems Research, 5-6-4, Tsukiji Chuo-ku, Tokyo 104, Japan

This report presents the architecture of FLATS2, which adopts the Cyclic Pipeline Architecture (CPA). In the CPA, the pipeline is shared among plural virtual prosessors to avoid the pipeline hazards and to make full use of its throughput. The FLATS2 architecture also includes the address range checking facility with the address tag and the in-word branch (BL-scheme), which efficiently implements the memory/data managements and the array accesses. To realize higher numerical performance, four arithmetic pipelines can be chained to work in parallel.

Also, this report roughly presents the instruction set architecture, the internal structure, and the pipeline design of FLATS2 machine.

1．はじめに

FLATS2[1]は、数式処理計算機FLATS[2,3,4]での経験と実績をふまえ、新たに数値・記号処理用計算機として設計された循環パイプライン計算機（Cyclic Pipeline Computer；CPC）である。FLATSが数式処理を主たる目的とした応用指向のLISPマシンであったのに対し、FLATS2は科学技術計算と記号処理をバランス良く実現するため、強力なメモリ・アクセス能力と高い数値演算性能を持つ、より汎用的色彩の濃いマシンとなっている。

本稿では、まずFLATS2アーキテクチャの特色を、（1）『循環パイプライン方式』、（2）『ＢＬスキーマ』の順にまとめ、さらに（3）FLATS2の命令レベル・アーキテクチャ、（4）FLATS2の内部アーキテクチャとパイプライン制御、について略述する。最後に（5）今後の課題を整理する。

2．循環パイプライン方式

2.1 動機

循環パイプライン方式（Cyclic Pipeline Architecture；CPA）は、ジョセフソン素子を用いた計算機に適したアーキテクチャとして提案された[5]。しかし、ＣＰＡは決してジョセフソン素子固有のアーキテクチャではない。FLATS2の目的の一つは、従来のシリコン技術上でＣＰＡの有効性を実証することである。

2.2 shared resource multiprocessors

ＣＰＡは、shared resource multiprocessors（Flynn[6]）と呼ばれるタイプのMIMD型アーキテクチャである。このタイプのアーキテクチャを持つ計算機は非常に少なく、他に商用機として Denelcor社の HEP[7,8]（pipelind MIMD）が知られるのみである。そこで、以下にこのアイデアを簡単に紹介しておく。

通常、パイプラインが深くなると、パイプライン内で実行中の命令同志が資源の競合を起こし（ハザード）、後続命令の実行が遅らされて実効性能が低下する。このため、パイプラインを細分化して深くしても、性能が向上するとは限らない。従来、この性能低下をいかにして防ぐかが、パイプライン設計上のポイントであった。

ＣＰＡでは、パイプライン内で並列に実行される命令をそれぞれ別の命令列（仮想的プロセッサ）に振り分ける。各命令列が異なったプロセッサのものであれば、使用する資源は異なっており、従って基本的に資源の競合は起こらない。こうして実現された計算機は、ユーザから見ると１つのハードウェアをパイプライン的に共有する MIMD型計算機である。

このようにして、ＣＰＡでは、深いパイプラインでもハザードにともなう性能低下を防ぐことができる。そこで、従来のパイプラインに比べて各ステージの遅延時間τを小さく設定し、より段数の多いパイプラインを設計することによって、ハードウェアとしてのスループット（各仮想プロセッサのスループットの総和）を高めることができる。

2.3 pipelined memory access

前節で述べた各仮想プロセッサは、同じハードウェアを時分割で共有しているため、プロセッサ外の資源を容易に共有することができる。例えば、ハードウェアとしての主記憶はプロセッサ外の資源にあたるが、これはきわめて自然に共有される。主記憶とＣＰＵ間の転送路を各プロセッサが時分割で使用することによって、特別なハードウェア（スイッチ・ネットワーク等）を付加することなく、主記憶共有型マルチプロセッサを実現することができる。

もちろん、τを短縮してＣＰＵのスループットを上げた以上、それに合わせて主記憶の転送幅も大きくする必要がある。この辺の事情はどのようなアーキテクチャについても共通であるため、従来からキャッシュやインターリーブ等、各種の実現技法が用いられてきた。ＣＰＡに対してもこれらの手法は適用可能であるが、主記憶へのアクセスをＣＰＵと同じτでパイプライン化することによって、より自然に解決することができる[5]。

パイプライン化された主記憶は、アクセス要求を受けると処理を開始するが、その処理が継続中でもτ後には次の要求を受け付ける。従って、ＣＰＵに釣り合う転送幅を実現することが可能である。また、インターリーブ・メモリにおけるバンク・コンフリクトの問題等も生じない。

パイプライン化された主記憶を効率的に実現するには、数段にパイプライン化されたメモリ・チップが不可欠である。しかし、このようなチップは未だ実現されていない。こうしたパイプライン化メモリ・チップは、スーパー・コンピュータの大容量ベクター・レジスタなどにも転用可能であるため、チップ化されればたいへん有益であると考えられる。

2.4 循環パイプラインの特徴

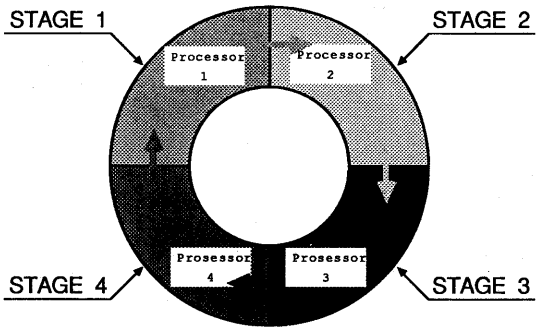
HEPは、FLATS2と同じ shared resource MIMD でありながら、設計思想が大きく異なっている。HEPでは、仮想プロセッサの代わりに「プロセス」をハードウェアでサポートし、メモリ・アクセスもパケット交換網を介して行なう。このように、システムの柔軟性が非常に高いため、複数のＣＰＵやメモリをもつシステムを自由に構築することができる。総じて、極めて並列指向の強いシステムであるといえる。

その一方、HEPのシステムは複雑で、１命令あたりの処理時間が比較的長いため、プロセスあたりの性能（スカラー性能）が低い。処理にキューやパケット網が介在するため命令処理時間も一定せず、先行制御によってスカラー性能を上げることがほとんどできない。

一方FLATS2では、メモリ・アクセスを含めた計算機全体をパイプライン化するため、先行制御によって並列度を抑え、スカラー性能を改善することができる。FLATS2では「並列性は必要悪である」という観点にたって、先行制御を積極的に採用し、仮想プロセッサ数を減らしてスカラー性能を上げている。

筆者たちは、このように
1）shared resource MIMD を採用し、
2）ＣＰＵと主記憶を同じτでパイプライン化し、
3）可能な限り先行制御も取り入れる
アーキテクチャを、循環パイプライン・アーキテクチャ（Cyclic Pipeline Architecture; CPA）と呼んでいる。主記憶を含めた計算機システムの中を、複数の仮想プロセッサが還流しているというイメージである。

循環パイプラインについての詳細は、参考文献[5]を参照されたい。また、文献[9]では、HEPとFLATS2が比較・検討されている。併せて参照されたい。

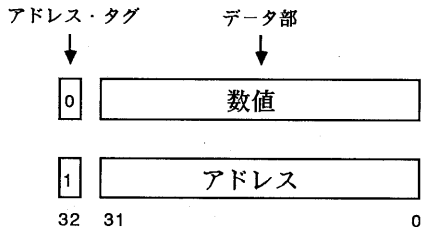


3．BLスキーマ

3.1 アドレス・タグ

FLATS2では、データ32ビットにつき１ビットのタグがある。このタグはアドレス・タグと呼ばれ、データ部がアドレスであるとき１になる。このタグは、次に述べるアドレス範囲検査機能の基本になる。

一般に記号処理用計算機では４ビット以上のタグを持つことが多いが、FLATS2の場合、数値計算にも必要な範囲で最小限のタグをもたせるにとどめた。



3.2 アドレス範囲検査とアクセス・エラー処理

FLATS2では、1)新たなアドレス・データを作る場合、または2)メモリにアクセスする場合は、必ずアドレス範囲の検査が行なわれる。アドレスの検査は、ベース・アドレスとリミット・アドレスのペアに対して行なわれ、

1)ベースとリミットは正当なアドレスか、

2)アドレス計算のオペランドは正当か、

3)アドレス計算の結果がベースとリミットの間にあるか、

を検査して、問題があればアクセスは拒否される（アクセス・エラー）。

ベースとリミットは、命令で指定したレジスタ・ペアから読み出されるため（ＢＬペア）、プログラマの目的に応じて様々な用途に使用することができる。また、アクセス・エラーが発生した場合には、分岐またはトラップによって制御フローを変えることができる。

以上のように、FLATS2ではメモリ・アクセス時にアドレス計算と範囲検査を同時に行ない、さらにエラー処理方法まで命令内で指定する。しかも、正常にデータがアクセスできた場合は、同一命令内で演算も行なう。これらの処理が全て１サイクルで実現されるため、メモリ上のデータを大変効率よく処理することができる。

3.3 数値演算におけるＢＬスキーマ

FLATS2では、配列の上限と下限を指すレジスタ・ペアによって、配列型のデータを表現する。ループ内でこの配列を参照するときは、あらかじめこの配列を示すＢＬペアを作っておいて、アクセス時にＢＬペアとして指定する。

この範囲検査機能によって、不当な添字による配列アクセスを検出し、トラップによってＯＳに制御を渡すことができる。

また、ループ内で、添字を変えながら配列にアクセスする場合、ループの脱出判定も行なえる。配列の終端に達すると、アクセス・エラーが起きるので、エラー・ジャンプによってループを脱出することができる。これについては、後の節で実例を示す。

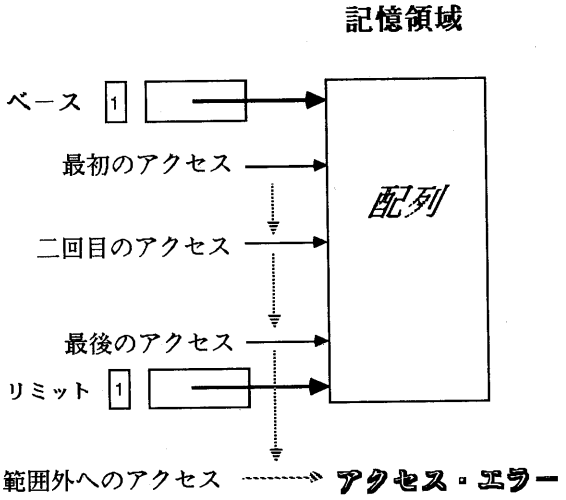
3.4 記号処理におけるＢＬスキーマ

例として、LISPの実現を考える。LISPでは実行時の型判定が大変重要なので、ＢＬスキーマを使ってこれをどのように実現するか、例を挙げて述べる。

まずアドレス・タグによって、「整数」か「オブジェクトへのポインタ」かが判定できる。加算命令等でアドレス・タグの立ったデータを使用すると、オペランド異常が検出されてトラップが発生する。トラップ・ハンドラでbignum処理を行えば、generic演算が実現できる。

carやcdrのような関数については、リスト領域を指すＢＬを使ってメモリを読むだけでよい。引数がリスト以外の型だった場合はアクセス・エラーが発生するので、引き続きエラー・ジャンプ先で対処する。consのフリー・セル領域の管理なども、ＢＬペアで行なうことができる。フリー・セルが無くなったら、アクセス・エラーでガベジ・コレクタ・ルーチンに飛べばよい。

ＢＬスキーマは、LISPマシンにおけるタグ・アーキテクチャほどは、LISPのセマンティクスに合っていない。しかし、より一般的な考え方であるため、柔軟性に富んだ使い方ができる。また、動作が単純で１サイクルで実行されるため、たいていの基本操作については、LISPマシンより高速に実行できると考えられる。



4．命令レベルのアーキテクチャ

4.1 設計思想

4.1.1 命令の単一サイクル実行

FLATS2では、命令が基本的に１サイクルで解釈実行される。そのため、命令長は固定され、形式も単純化されている。この意味で、FLATS2は RISC[10] と似た面を持っている。

しかし、FLATS2はRISCではない。第一の相違点は、FLATS2の命令が大変に多く、決して命令数を削減したとは言えないことである。RISCと呼ばれるマシンでは、通常命令数が100に満たないが、FLATS2は200を越える命令を実装する。

また、RISCで積極的に否定しているような、複雑な機能を持つ命令も多く実装している。例えば、浮動小数点演算、乗除算、整数／浮動小数点数の型変換、等の命令である。その上、内積プリミティブ命令、ＦＦＴプリミティブ命令、といった特定用途向け命令や、LISPプリミティブ命令群など、反RISC的命令も多く実装している。もちろん、これらの命令も１サイクルで実行される。

4.1.2 ＣＰＡの命令セットへの影響

FLATS2において、複雑な命令が単一命令サイクルで実装できるのは、循環パイプラインのお陰である。

通常、複雑な機能を実装すれば、どうしても論理が複雑になり、遅延時間が増大する。ハザードの関係上、パイプライン段数には実質上の制限があるので、複雑な機能をハードウェアで実装すると、クロックが下がる（性能が下がる）。しかし、循環パイプラインでハザードを回避すれば、パイプラインを細分化することによって、クロックを落とさずに比較的複雑な命令を実装することができる。

4.1.3 数値計算のサポート

ＣＰＡでも、スカラー性能を重視するなら、複雑な命令を排除してパイプラインを短くした方がよい。しかし、FLATS2は科学技術計算をターゲットとしているので、そのために必要な命令は積極的にハードウェアでサポートした。

このため、FLATS2では、

(1) ハードウェアによる浮動小数点演算
　（加減乗除、型変換、など）
(2) 演算パイプラインのチェイン
　（内積、ＦＦＴ、複素演算、など）
(3) ループ最適化の支援
　（ＢＬスキーマ、演算型分岐命令、など）

等を行っている。これらの比較的複雑な機能も、ＣＰＡによって１命令（１サイクル）で実現できた。

4.2 プログラミング・モデル

4.2.1 アドレス空間

FLATS2のアドレス空間は、32ビットのアドレスで表現され、データ用のＤ空間（$00000000～$7FFFFFFF）、命令用のＩ空間（$80000000～$BFFFFFFF）、レジスタ・フレーム用のＶ空間（$C0000000～$FFFFFFFF）に分割されている。メモリ・セルは、32ビットのデータと１ビットのアドレス・タグから構成される。

各空間は仮想化されており、Ｄ空間にはバイト・アドレス、Ｉ／Ｖ空間にはワード・アドレスが付されている。

4.2.2 レジスタ構成

レジスタは、各33ビットで64本あり、うち32本（０～31番）がプロセス内で共有されるグローバル・レジスタ、残り32本（32～63番）がcall／return命令によって切り替わるローカル・レジスタである。ＢＬスキーマでは、積極的にＢＬベアを作ってメモリ・アクセスに使用するため、レジスタ数が多めに設計されている。

ローカル・フレームはＣＦＰ（Current Frame Pointer）レジスタによってＶ空間にマップされ、同じくグローバル・フレームはＧＦＰ（Global Frame Pointer）でＶ空間にマップされる。call／return命令実行時には、ＣＦＰレジスタの書き換えによってローカル・フレームを切り替える。

この他、数値演算ユニットのアキュムレータ／作業用レジスタとして、Ｐ，Ｑ，Ｒ，Ｓ，Ｔ，Ｕレジスタが用意されている。これらは全て２ワード長で、数値演算用命令でオペランドとして使用できる。

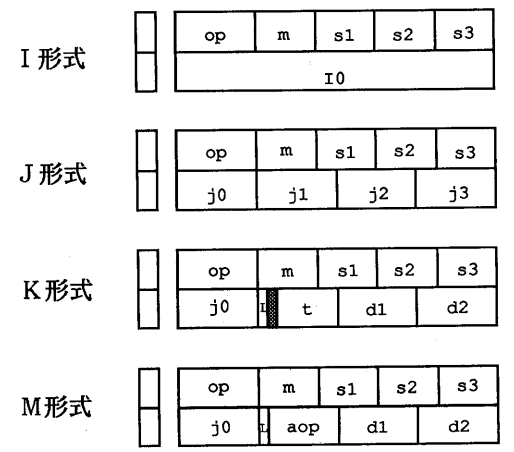
また、プログラム制御用には、ＰＣ，ＣＦＰ，ＧＦＰ，ＣＣ，ＰＳＷ等のレジスタがある。これらについては、普通の汎用機と同じである。

4.3 命令

4.3.1 命令形式

FLATS2の命令は、全て２ワードの固定長である。ただし、命令が長いイミディエイト値をとる場合は、命令の後ろにイミディエイトが２ワード続いて、４ワード長になる。

命令形式は４つで、主にレジスタ間演算に用いるＩ形式、分岐命令に用いるＪ形式、アドレス計算とロード／ストアに用いるＫ形式、メモリ・オペランドを用いた演算操作に用いるＭ形式、に分類できる。



この中ではＭ形式が最も一般的で、なおかつ、FLATS2の特徴をよく表している。Ｋ形式はＭ形式の変形である。Ｊ形式は、ループの構成に用いる演算型分岐命令や、条件分岐命令（テスト＋４方向分岐）に用いられる。

4.3.2 フィールド

opは、命令コード・フィールドである。ただしＭ形式の

場合、opフィールドでアドレッシング・モードを指定し、aopで演算方法を指定する。mはモード・フィールドで、オペランドを指定する。s1,s2,s3は、読み出すレジスタの番号である。FLATS2のレジスタは3 read port／1 write portなので、s1～s3はレジスタの読み出しアドレスになる。書き込みアドレスは、K形式ではtフィールドで指定するが、それ以外では命令によってs1～s3のいずれかに定まっている。

j0～j3は、命令内の分岐フィールドである。各8ビットで、-128から127までの相対分岐先を指定する。d1,d2,I0はディスプレースメント・フィールドで、アドレス計算時の偏位や、演算時の即値データに用いられる。

4.3.3 アドレシングとBLスキーマ

FLATS2では1サイクルでアドレス計算を行うため、アドレシング・モードが比較的単純で、基本的に以下の3種類しかない。

(1) インデックス　　BL : base + index
(2) オフセット　　BL : base + offset
(3) ポインタ　　BL : pointer + offset

「BL:」は、計算したアドレスをBLペアでチェックすることを意味する。baseは、BLペアのベース・レジスタである（従って、アドレス型データ）。indexはレジスタの整数型データ、pointerはレジスタのアドレス型データ、offsetは命令内のイミディエイト・データである。

これらのアドレシング・モードには、副作用も指定できる。副作用には、プリ・モディファイ（プッシュ）、ポスト・モディファイ（ポップ）が用意されている。

4.3.4 命令の基本動作

ここでは、実際にM形式の命令を例に取って、FLATS2の命令動作を説明する。まず、配列の総和を計算するプログラムを考える。

```
for i := 0 to IMAX do
    S := S + array[i];
od;
```

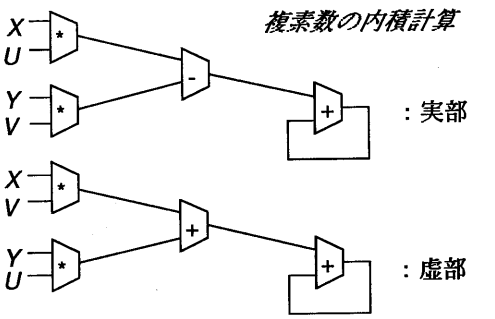
このループは、次のような1命令ループに最適化される。

```
loop:  add.l.j  vr10:vr10@<4, S, S, loop
```

ただし、あらかじめ、vr10にベース・アドレス（array[0]のアドレス）、vr11にリミット・アドレス（array[IMAX]のアドレス）を用意しておく必要がある。この計算は1命令で済む。また、ループ本体も1サイクル／回で実行できるため、大変効率のよい処理が可能である。

ループ内では、vr10は副作用で4ずつ増加し、arrayの要素が次々読み出されてSレジスタに累算される。さらに、範囲検査でエラーが検出されなければ、命令内のジャンプ・フィールドで指定された番地（loop）に分岐する。配列の境界に達して、vr10＋4の値がvr11を越えると、副作用でvr10に書き込まれる値からアドレス・タグが落ちて、vr10がアドレスでなくなる。すると、次のサイクルで範囲検査が失敗し、loopへの分岐が起こらないので、ループから脱出する（次の命令に進む）。

このようにFLATS2の命令セットでは、BLスキーマによって、単純なループを1～数命令ループに最適化することが可能である。検討の結果、LINPACライブラリの多くの下位ルーチンが、3命令程度のループに最適化できることがわかった。また、専用の演算命令を使用して演算器をチェインすると、複素配列の内積は1命令ループ、FFTの最内周ループは2命令ループに最適化できる。FLATS2では、これらの専用命令を、『数値演算用特殊命令』と呼んでいる。

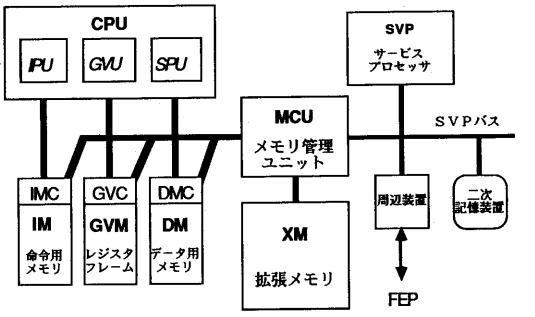


図：パイプラインのチェイニング

5．内部アーキテクチャとパイプライン制御

5.1 全体構成

図に、FLATS2のハードウェア構成を示す。



図：FLATS2計算機の構成

CPUは大きく3つのユニットに分かれており、それぞれIPU（命令処理ユニット）、GVU（アドレス演算ユニット）、SPU（数値演算ユニット）と呼ばれている。IPUはIM（命令用メモリ）、GVUはGVM（大容量レジスタ用メモリ）、SPUはDM（データ用メモリ）に直結されている。各メモリから並列にデータ転送を行うことにより、各ユニットを並列に動作させ、高い性能を得る。

各メモリ・モジュールは、CPUの他、MCUにも結合されている。MCUは、元々メモリ制御ユニットという意味で、メモリに対するDMA制御を行う。MCUは、SVPバスという外部バスにつながっており、SVPバス上のデバイスに対して入出力を行う。また、MCUは、ステー

ジング用の大容量メモリ（拡張メモリ；XM）にも結合されている。

実装技術には、主としてＥＣＬ（10K,100K）＋多層基板が用いられる。

5.2 メモリ・システム

ＩＭ、ＧＶＭ、ＤＭは、キャッシュ・メモリの構成をとり、おのおの以下のような仕様を持つ。

(1) ＩＭ　　4 way set associative, 66bit幅
(2) ＧＶＭ　　1 way set associative, 66bit幅
　　3 read ports, 1 write port
(3) ＤＭ　　4 way set associative, 66bit幅

実装には、アソシエーション部にＥＣＬの論理とＲＡＭ（1K×4bit）、メモリにＣＭＯＳスタティックＲＡＭ（64K×1bit）を使用する。サイクル・タイムは50nsの予定である。

5.3 ＣＰＵ

ＣＰＵは多層基板１７枚で構成され、分散型水平マイクロプログラムで制御されている。制御記憶（ＷＣＳ）の幅は、全体で300ビット余りである。各基板には、200～400個のECLチップが実装される。

ＩＰＵは、命令のフェッチ、デコード、ＰＳＷの管理を担当する。基板４枚から構成され、100ビット強のＣＳを持つ。ＧＶＵは、レジスタ間演算、アドレス計算、ＢＬスキーマの実現を行なう。基板３枚から構成され、ＣＳは60ビット強である。ＳＰＵは、データのフェッチ／ストア、オペランドの選択、演算処理を担当する。基板は、ＤＭの管理に３枚、汎用ＡＬＵに２枚、乗算器に２枚、演算器の入出力管理に３枚の、計１０枚使用されている。

5.4 パイプライン制御

5.4.1 クロックと期待性能

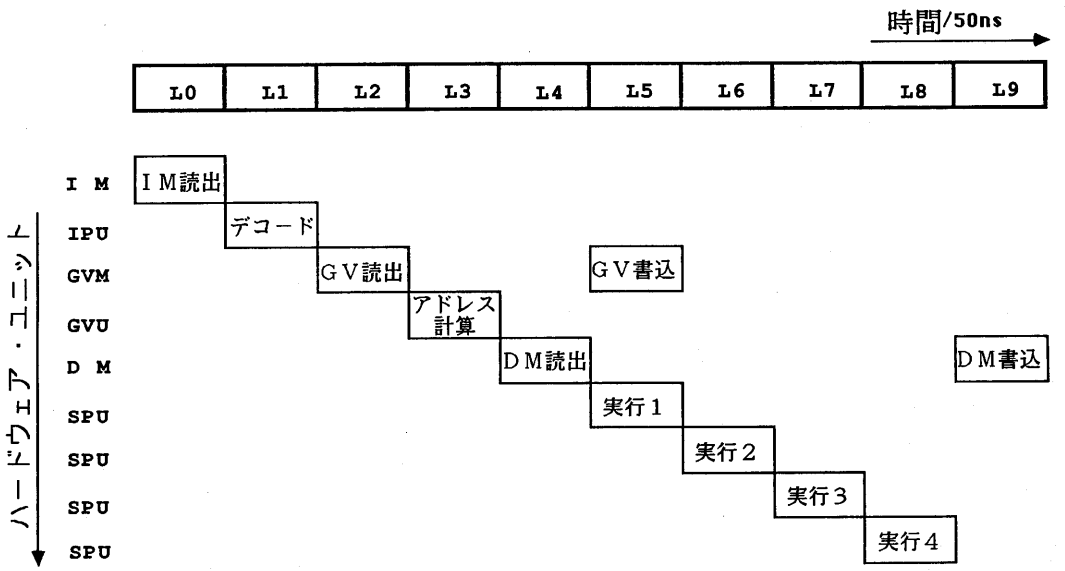
FLATS2の命令処理は10段にパイプライン化されており、基本クロック・サイクルは50nsである。即ち、50nsに１段ずつ命令がパイプライン内を進む。命令は、２つの連続するパイプライン・スロットを用いて実行され、各２回のメモリ・アクセスが許されている。つまり、パイプラインは100nsに１つの割合で命令を処理するから、ハードウェアとしての最大スループットは１命令／100ns＝１０MIPSとなる。

FLATS2は、プロセッサ数が２の循環パイプライン計算機であるため、２つの命令列の命令が交互にパイプラインに投入される。従って、プロセッサ１台あたりのスループットは１０MIPS／２＝５MIPSとなる。この値は、最近のプロセッサと比べて特に大きい値ではないが、FLATS2の命令が非常に高級な機能を含むことを考えれば、そう悪くはない値だと考えられる。

また、数値演算用特殊命令を使用した場合、一命令につき最大４本（倍精度）／８本（単精度）の演算器が並列に動作する。従って、単精度で最高８０MFLOPS（複素数の内積）～５０MFLOPS（複素数のＦＦＴ）、倍精度で最高４０MFLOPS（複素数の内積）の性能を得ることができる。

5.4.2 パイプライン・チャート

図に、FLATS2の基本的なパイプライン・チャートを示す。パイプラインの各フェーズで行われる処理は以下の通りである。



図：基本的命令のパイプライン処理

Ｌ０：　命令メモリ（ＩＭ）から命令を読み出す。
Ｌ１：　命令の即値オペランドを、ＩＭの
　　（ＰＣ＋１）番地から読み出す。
　　フェッチした命令をデコードする。
Ｌ２：　ＧＶＭからレジスタの値を読み出す。
Ｌ３：　レジスタの値を用いて実効アドレスを計算する。
　　実効アドレスの範囲検査を行う。
　　新たなＰＣ（次命令の番地）を決定する。
Ｌ４：　データ用メモリ（ＤＭ）から
　　メモリ・オペランドを読み出す。
　　レジスタに残す副作用の計算を行なう。
Ｌ５：　オペランドのセレクトを行なう。
　　演算ユニット（ＳＰＵ）で演算を実行する。
　　レジスタ（ＧＶＭ）への書き込みを行う。
Ｌ６：　演算ユニット（ＳＰＵ）で演算を実行する。
Ｌ７：　演算ユニット（ＳＰＵ）で演算を実行する。
Ｌ８：　演算ユニット（ＳＰＵ）で演算を実行する。
Ｌ９：　演算結果をデータ用メモリ（ＤＭ）に書き込む。

このパイプラインに、２つの命令列から命令が交互に投入され、実行が進む様子を図に示す。FLATS2のパイプラインでは５つの命令がオーバーラップして実行されるが、先行制御の導入により命令列の数（プロセッサ台数）を５から２に減らしている。これにより、各プロセッサの性能（スカラー性能）を改善している。

FLATS2では、先行制御の工夫として２つの手法を採用した。１つめはデータの依存性を解消するためのバイパス機構、２つめは条件分岐を高速化するための工夫（例外遅延高速分岐方式；FBED）である。これらの工夫について、以下で簡単に説明する。

5.4.3　バイパス制御

ある命令がメモリの内容を変更し、その次の命令が同じ番地を参照する場合を考える。FLATS2のパイプラインでは、後続命令がＤＭＲフェーズでデータを読み出す時、先行する命令はまだＤＭＷフェーズまで進んでいないため、メモリの内容がまだ古い値のままである。

プロセッサ数を増やせば（例えば４台）、先行する命令が書き込みを終了してから後続命令を実行できるが、スカラー性能が半減してしまう。そこで、FLATS2ではバイパス機構を採用した。即ち、先行命令のＤＭＷと並行して、後続命令に変更後のデータを供給する（図の矢印）。

5.4.4　条件分岐の高速化

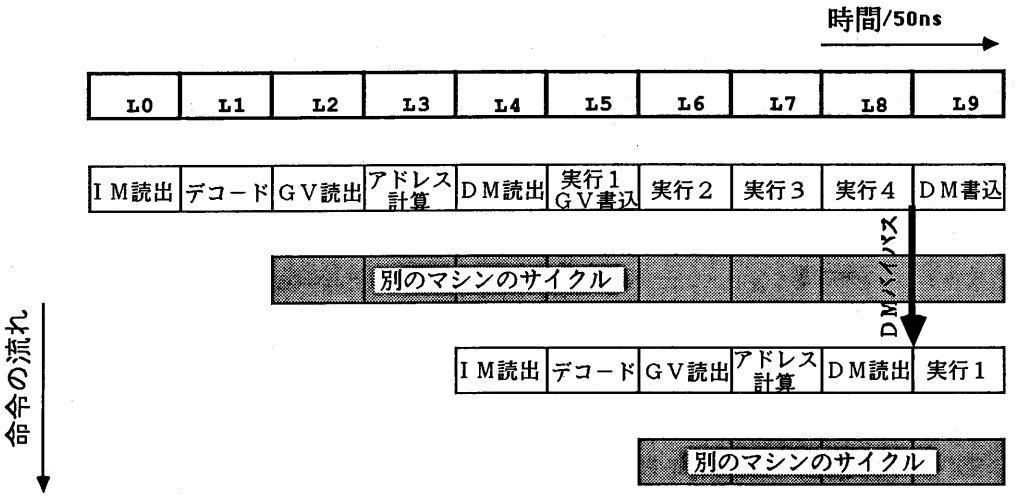
パイプライン計算機では、条件分岐にともなってパイプラインの流れが乱れ、スループットが下がることが知られている。FLATS2には以下の２通りの条件分岐が存在し、それぞれ次のように実行される。

●アドレス計算に伴う条件分岐

FLATS2の命令は命令内に分岐フィールドを持ち、ＢＬスキーマに伴う条件分岐は演算と並列に行われる。即ち、分岐条件はアドレス演算と同時に判明し（Ｌ３終了時）、次命令のフェッチ（Ｌ４）は遅れを伴わずに実行される。

●数値演算に伴う条件分岐

ＳＰＵで実現される数値演算については、条件コードがＰＳＷ内に存在し、条件分岐命令によって条件の検査と分岐操作が行われる。この場合、条件コードが演算の「実行３」フェーズ終了時までに判明すれば、次命令の条件分岐操作は余分な遅れなしに分岐先を判定することができる。ＳＰＵでの多くの数値演算はこの場合に含まれ、実行３フェーズまでに演算結果および条件が確定する。



図：命令のオーバーラップとバイパス機構

しかし、ＳＰＵで乗算器を使う数値演算では、実行４フェーズ終了まで演算結果が判明しない。実行３フェーズまでに条件コードが確定しないとパイプラインに遊びが生じてスループットが低下するため、FLATS2では（可能ならば）条件コードを演算結果に先だって求めることにより、性能低下を防いでいる。

現実には、オペランドの概数（log値）を使って、大抵の場合、実行３フェーズに条件コードを確定させることができ、性能低下を例外的な場合のみに抑えることができる。FLATS2ではこの方法をＦＢＥＤ（例外遅延高速分岐方式）と名付けた。ＦＢＥＤの詳細に付いては、参考文献[11]を参照していただきたい。

このように、循環パイプラインは shared resource MIMD でありながら、積極的に先行制御を採用して並列度を下げている。循環パイプラインは、先行制御の実現コストを下げ、先行制御の効率を向上させる技術であるとも言える。

６．今後の課題

FLATS2のハードウェアは、現在製作・デバッグ中である。ハードウェアと並行して、アセンブラ、コンパイラ（C,Fortran）等の開発が進んでおり、FLATS2シミュレータの上で動作している。また、FLATS2のオンライン保守ツールも、ほぼ完成している。これらのソフトウェアは、ハードウェアのテスト／デバッグに利用される予定である。FLATS2のソフトウェア・システムについては、また別の機会に発表することとする。

FLATS2の実機での性能評価については今後の課題である。また、FLATS2シミュレータ上での性能評価についても、機会を改めて発表する予定である。

FLATS2はプロセッサ数が２の循環パイプライン計算機（ＣＰＣ）であるが、ジョセフソン素子を用いて構築されるＣＰＣでは、かなりプロセッサ数が多くなると予想されている。そのためにも、まず比較的プロセッサ数の少ないFLATS2で実際に論理設計を行い、循環パイプラインの設計上の特色を明らかにし、設計経験を積むことが重要である。また、今後、プロセッサ数の大きな循環パイプラインを設計するために、一層の検討と考察を行なう必要がある。

参考文献

[1] 後藤・市川・他： FLATS2のアーキテクチャ、第35回全国大会、6C-4.

[2] E.Goto, T.Soma, et al.: "FLATS: A Machine for Symbolic and Algebraic Manipulation," in The Second RIKEN Int'l Symp. on Symbolic and Algebraic Computation by Computers, ed. N.Inada and T.Soma, pp.231-246, World Scientific, Singapore, 1985.

[3] K.Hiraki and E.Goto: "An Architecture of FLATS - A Computer for Symbolic and Algebraic Computations," 情報処理学会論文誌（邦文）, vol.27, no.1, Jan 1986.

[4] M.Suzuki, K.Hiraki, et al.: "FLATS architecture and its Evaluation," in the Proceedings of IEEE TENCON87, Seoul, Korea, 1987.

[5] K.Shimizu, E.Goto, and S.Ichikawa: "CPC(Cyclic Pipeline Computer)-An Architecture suited for Josephson and Pipelined Machines," IEEE Trans. on Computer(to appear).

[6] M.J.Flynn: "Some Computer Organization and Their Effectiveness," IEEE Trans. on Comp., vol.C-21, pp.948-960.

[7] H.F.Jordan: "Performance Measurements on HEP - A Pipelined MIMD Computer," Proc. of 10th Annual Int'l Symp. on Computer Architecture,pp.207-212.

[8] H.F.Jordan: "HEP Architecture, Programming, and Performance," in Parallel MIMD Computarion : HEP Supercomputer and Its Applications, ed. J.S. Kowalik, The MIT Press, 1985.

[9] S.Ichikawa: "A Study On A Cyclic Pipeline Computer: FLATS2," 修士論文、東京大学理学部情報科学科、1987.

[10] D.A.Patterson: "Reduced Instruction Set Computers," CACM, vol.28, no.1, pp.8-21, Jan.1985.

[11] S.Ichikawa and E.Goto: "Fast Branching with Exceptional Delay with a proposal for standardization," 参考文献[9]の付録.