計算機アーキテクチャ　89−14
（１９９1.　7.　１9）

条件実行アーキテクチャＧＩＦＴのメモリインタフェース

鈴木 英俊[*1]　小松 秀昭[*2]　深澤 良彰[*1]　門倉 敏夫[*1]

*1：早稲田大学理工学部
*2：日本アイビーエム東京基礎研究所

ＶＬＩＷアーキテクチャにスケーラビリティを持たせるとともに，ジャンプによる実行遅延を軽減する命令供給機構と，制御依存を越えた先行実行を効率良くサポートする方法について述べる。スケーラブル化を，コードをキャッシュにコピーする時にリソーススケジュールを行なうことで，命令実行性能を通常のＶＬＩＷと同等にすることができる。また，命令キャッシュ内で命令語にシーケンスアドレスを付加し，これを動的に変更することによって，パイプライン分断の抑制を可能とした。さらに，１つの命令に通常用と制御依存を越える先行実行用とを用意し，先行実行用命令が発生させたページフォールトやエラーなどの復旧を先送りさせる方法を提案する。

The Memory Interface for GIFT

Eishun Suzuki[*1]　Hideaki Komatu[*2]　Yoshiaki Fukazawa[*1]　Toshio Kadokura[*1]

*1:School of Science & Engineering, Waseda University
*2:IBM Japan,Ltd. Tokyo Research Laboratory

In this paper, we describe three features of our extended VLIW architecture GIFT (Guarded Instruction architecture for Fine-grain Technique). (1) Scalable VLIW architecture : By rescheduling instructions not compile time but cache recovering time, our architecture has same performance as traditional VLIWs. (2) A new instruction supply facility : In execution jump-instruction, flushing of instruction pipeline is reduced by both adding microprogramming-like sequence address to each instruction word in cache, and updating it dynamically to newest jump address. (3) Effective eager evaluation : The recovery for page faults, arithmetic errors and so on caused by eager evaluation is put off by proparing two kinds of instruction. One is normal, and another is eagerly evaluated.

1．はじめに

現在，並列計算機は科学技術計算の分野ではアレイプロセッサやベクトルプロセッサによって良い成果を修めている。しかし，ＯＳやコンパイラなどのソフトウェアに対してはあまり成果が上がっていない。これは，これらのソフトウェアの持つ並列性が元来少ないことや，ジャンプやデータの生成待ちなどによって実行パイプラインが分断されることによっている。これまで，このようなソフトウェアにも効率よく対処していこうとする細粒度並列アーキテクチャが数多く提案されている[1][2][3]。我々もＶＬＩＷ形の命令語を持つ条件実行アーキテクチャＧＩＦＴ[4](Guarded Instruction Architecture for Fine-grain Technique)を提案してきた。今回，条件実行アーキテクチャを用いて，ＶＬＩＷにスケーラビリティを持たせることを試みた。また，単純な大域的なコードの移動による並列化に比べ，大幅に並列性を向上させることができる条件分岐点を越えた先行実行時のメモリロードや，演算エラーによる効率低下を抑える手法も提案する。

2．研究目的

現在，代表的な細粒度並列アーキテクチャにはスーパースカラ方式とＶＬＩＷ方式がある。スーパースカラ方式は実行時に細粒度の並列性を抽出し，並列実行しようとするものである。これは，実行時にハードウェアによってデータ依存解析やリソース競合解析を行なうことによって実現されている。このため，並列度の異なるマシン間でもプログラムのバイナリレベルでの互換性があり，拡張性に優れている。しかし，実行時に多くの作業をするため回路が複雑になり，オーバヘッドが大きくなってしまう。それゆえ，スーパースカラ方式ではマシンの並列度を上げにくく，構成要素に非常に高速でコストの高いものが要求される。また，並列性を抽出やコードスケジューリングはプロセッサ内で行なわれるため，局所的なものになってしまう。

ＶＬＩＷ方式は細粒度の並列性の抽出やスケジューリングをコンパイル時に行い，並列実行しようとするものである。このため，並列性の抽出を大域的に行なうことができ，また，実行時には，プロセッサは静的にスケジュールされた命令語を実行するだけでよいので構成が簡単になり，比較的高速で高並列度のマシンを作りやすい。ＶＬＩＷ方式はマイクロプログラミングのように非常に長い命令語を持ち，一つの命令語の中に複数の演算命令がスケジュールされている。各命令は命令語のあるフィールド上に置かれ，それぞれの演算器を独立に操作する。各フィールド上に置ける命令の種類を制限した機能非均質形のＶＬＩＷ方式の場合，命令語の自由度が減るが，コストの増加を抑えることができる。しかし，コードスケジューリングがコンパイル時にそれぞれのマシンの命令語フォーマットに合せて静的に行なわれるため，バイナリレベルでの互換性はなく，拡張性が乏しい。

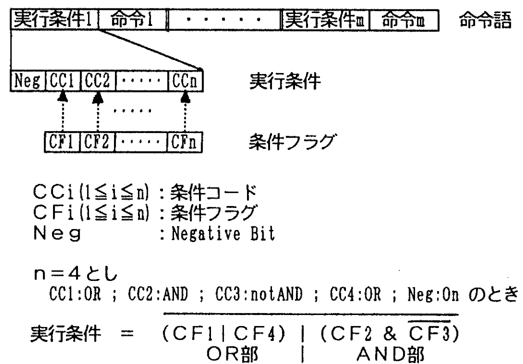
本研究ではＶＬＩＷ方式の高速化，高並列化の可能性に注目しながら，条件実行によって並列性抽出度をさらに向上させ，新しい命令供給機構の開発によってスーパースカラが持つような拡張性を持たせる拡張を行なう。

また，キャッシュメモリ中の命令語にマイクロプログラムの様に次のサイクルで投入する命令語の位置するアドレスを付加し，ジャンプ発生時のパイプラインの分断による効率低下を防いでいる。

さらに，条件分岐点を越えて先行実行された命令がページフォールトや演算エラーなどを起こした場合，その命令の生成値を使用する時点で，これらの復旧処理を行なわせる。これによって，条件分岐を越えて投機的に実行したが後に条件に適合せず，本来ならば実行されない命令が発生させる遅延を隠匿し，性能の低下を抑える。

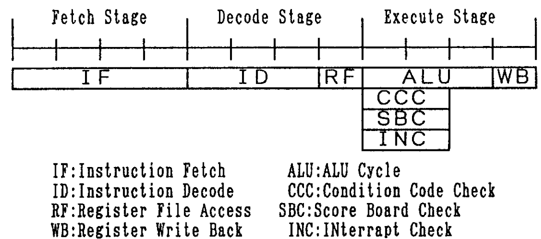
3．ＧＩＦＴの概要

ＧＩＦＴは複雑な条件構造に対しても柔軟な対応が可能な機能非均質形ＶＬＩＷ拡張アーキテクチャである。ＧＩＦＴではそれぞれの命令に，実行する場合に満足しなければならない条件「実行条件」を付加している。実行時に，マシンの持つ条件フラグの状態がその実行条件を満たすものであれば，それを持つ命令は実行される。実行条件は一つ以上の条件を持ち，それぞれの条件はプロセッサの持つ条件フラグと対になっている。この対は固定的である。条件は対となっている条件フラグを実行条件にどのように関与させるかを記述していて，Don't Care，OR，AND，notANDの四種類の属性を持っている。Don't Careは対となっている条件フラグを実行条件に含めないことを示す。OR，AND，notANDは対となっている条件フラグを，実行条件にOR結合，AND結合，否定のAND結合させることをそれぞれ示している。また，この実行条件の全体の否定を新たに実行条件にすることも可能である。各命令の実行と実行条件の判定は並行して行なわれる。実行条件が適合しなかった命令に関しては，演算結果をレジスタに書き込むことを禁止したり，パイプラインの途中で演算を取り消すことによって無効化している。



実行条件は，OR結合の条件フラグとAND結合の条件フラグのORとなる。Negがオンのときには，この否定が実行条件となる。

図1　ＧＩＦＴの条件実行機構



CCC：実行条件を評価する時間
SBC：スコアボードを評価する時間
INC：割込みの発生を評価する時間

同一の区間に存在している処理は並行に行なわれる。すべての命令の実行条件やスコアボード，割込みのチェックは，最初の実行ステージで演算と並行して行なわれる。

図2　基本パイプライン構成

ＧＩＦＴはＶＬＩＷ形アーキテクチャでありながらスケーラビリティを持っている。ＶＬＩＷ方式では並列性の抽出や依存性の検出，命令語の構築などすべてをコンパイラが静的に行なっている。一方，スーパースカラ方式ではこれらの処理はプロセッサが実行時に動的に行なっている。ＧＩＦＴでは，並列性の抽出，依存性の検出などのスケジュールはコンパイラが無制限のリソースを仮定して静的に行ない，そして，マシンのリソースに依存したスケジュールは動的に行なうという，二段階のスケジューリングを行なっている。ここで，スーパースカラ方式と異なる点は，動的なスケジューリングはプロセッサ内ではなく，メインメモリから命令用キャッシュメモリに格納される間で行なわれるという点である。命令がメインメモリからキャッシュメモリにフェッチされると，命令の種類が調べられ，プロセッサの持つ並列度や機能にあわせて再スケジュールし，リソース構成に従った命令語に変換してキャッシュメモリに格納している。ＧＩＦＴでは，この静的にスケジュールされた命令語を「論理命令語」，動的にスケジュールされた命令語を「物理命令語」と呼んでいる。

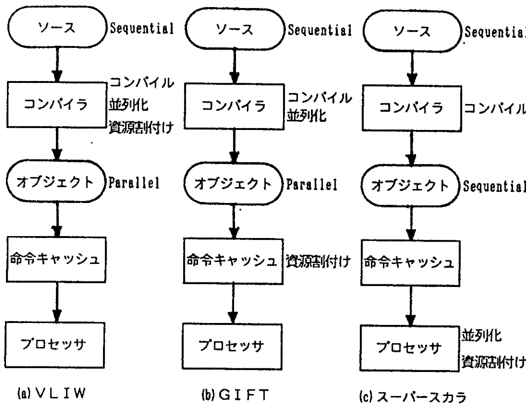


図3 二段階スケジューリング

また，スケーラブル化により，一つの論理命令語がプロセッサの持つリソースによっては１サイクルでは実行できないことがあり，この時には，複数のサイクルに分割されて実行されることになる。この場合，一つの論理命令語中で同一のレジスタに対する読込みと書込みをする「リードモディファイライト」を行なうと、プログラムの意味が変化してしまうことがある。本来，読込みを行なう命令は，これを含む論理命令語の実行直前にレジスタに設定されていた値を読み込むことを想定して，スケジュールされている。しかし，実行が分割された場合，先に実行された書込み命令がレジスタの内容を破壊してしまい，読込み命令が実行直前の値を読み込めない事態が発生してしまう。そこで，どのようなリソース構成のマシンで実行しても，コンパイラがスケジュールしたデータの生産－消費の関係を維持するために，レジスタファイルをリード用とライト用で二重化している。これによって，一つの論理命令語が実行を終了するまで，読み込む値を破壊されないようにしている。

キャッシュメモリ内の格納された命令語には，マイクロプログラムのように，次のサイクルでプロセッサに投入する命令語が位置するアドレス「次候補アドレス」が付加され，キャッシュメモリはこれを参照してプロセッサに命令語を供給している。ジャンプがある場合には，次候補アドレスをジャンプアドレスに書き換えることによって，２回目以降は連続的に命令語を供給している。

条件分岐点を越えて先行実行された命令が，計算値を生成するにあたって，キャッシュミスやページフォールト，エラーを発生させ，かつ，その計算値を後の処理で使用しなかった場合，プログラムの実行は大きな遅延を被ることになる。ＧＩＦＴでは命令に通常用と先行実行用の二種類を用意し，先行実行用命令がこれらの事態を発生させたときは，これを完全に復旧させず，その計算値を後の処理で通常用命令が使用する時点で復旧を行なうために必要な情報をレジスタに格納するのみとして，無用な遅延を抑えている。

４．ＧＩＦＴのスケーラブル化

ＧＩＦＴにスケーラビリティを持たせるために次の機構を開発した。

○可変長命令供給機構
○可変長－固定長命令変換機構
○二重レジスタファイル機構

可変長命令供給機構と可変長－固定長命令変換機構は対象となるプロセッサにあわせてプログラムコードを再スケジューリングする機構で，キャッシュメモリがこれを持つ。二重レジスタファイルは同一サイクル内でのリードモディファイライトを実現する機構である。

４．１ 可変長命令供給機構

ＶＬＩＷの命令語はリソースに依存した特定のフォーマットを持つ固定長命令語であり，一つの命令語中の命令が並列実行される。スケーラブル化すると，対象となるプロセッサが一定でない以上，メモリ上に存在している論理命令語は特定のリソースの構成依存しない，単に並列実行可能な命令が集まった可変長命令語となる。しかし，メモリからマシンへの命令供給はある幅を持った固定長なので，可変長命令語の境界を明示しなければならない。このためにＧＩＦＴでは各命令にストップビットを付加している。ストップビットは一つの命令語の終端を示すもので，これがオンとなっている命令までが並列実行可能である。つまりストップビットで仕切られた範囲が一つの固定長命令語である物理命令語になり得る。本機構はメモリから供給された命令をストップビットに基づいて該当する可変長命令語を抽出するものである。

| Memory1 | - | Fixed1 | S | Float1 | - | Memory2 | - | Jump1 | S |

供給単位

| Memory1 | - | Fixed1 | S |

| Float1 | - | Memory2 | - | Jump1 | S |

論理命令語

Fixed：固定小数点命令，Float：浮動小数点命令，Memory：メモリアクセス命令，Jump：ジャンプ命令，Ｓはストップビットがオンであることを表す。ここでは，命令の供給単位を１単位当り５命令としている。

図4 可変長命令供給

４．２ 可変長－固定長変換機構

本機構は，可変長命令供給機構から供給された論理命令語を，プロセッサの持つ物理命令語に変換するものである。論理命令語はリソース構成に依存しない，並列実行可能な命令の集まりでしかないため，直接実行はできない。そこで論理命令語をプロセッサの持つリソース構成に合った実行可能な物理命令語に変換しなければならない。これは論理命令語に

含まれる命令の種類を判別し，プロセッサのリソース構成にあわせて論理命令語の中から命令を抽出し，物理命令語に並べ替えることによって行なう。

4.2.1 論理命令語のスケジューリング

物理命令語の並列度が論理命令語の並列度より小さい場合には，一つの論理命令語は複数の物理命令語に並べ替えられる。一つの論理命令語に含まれる各命令は並列実行可能なので，特に並べ替えの順序は問われない。しかし，あらゆる並列度のマシンにおいても効率的な実行を行なうためには，より実行頻度の高い命令順に並べ替えなければならない。これは，条件実行アーキテクチャの場合，一つの命令語の中に実行条件の異なる命令が存在できるため，実行条件が満たされないために実行されない命令のみをプロセッサに投入する可能性があり，単に命令をやり過ごすためだけのサイクルが生じてしまうことがあるためである。そこで，静的なスケジューリングの段階で，コンパイラは実行条件が満たされない命令の投入をできる限り防止するために，ジャンプ命令によってカットが行なわれるようにスケジュールしている。例えば，図5 (a)は条件木で， 各枝のI，F，J，Mは命令であり，Iは整数命令，Fは浮動小数点命令，Jはジャンプ命令，Mはメモリアクセス命令とする。この条件木中の命令は依存性がなく並列実行が可能であるとする。つまり，これらの命令は一つの論理命令語に割り付けることができる。この木中のL1～L3は，この木の実行が終了した後に，制御を移す先とし，L1，L2，L3に至るパス順に実行頻度が高いものとする。このとき，コンパイラは実行頻度を考慮しながらこの木を図5 (b)のように論理命令語に割り付ける。図5 (b)では，最も実行頻度の高い命令I_1は最下位アドレスに，最も実行頻度の低いJ_3は最上位アドレスに割り付けている。 仮に，この論理命令語を一命令ずつ実行するとした場合，L1へ至るパスが選択されるとI_1，I_2，F_1，F_2 の順に命令がプロセッサに投入され，最後にL1へのジャンプ命令J_1が投入される。J_1が実行されるとL1へ制御が移るので，その後の命令の投入を防ぐことができる。L2へ至るパスが選択されたときはF_2，J_1が，L3のときはF_1，F_2，J_1，I_3，J_2 が投入されるが実行されない命令となってしまうが，実行頻度を考慮することによって，小さな損失に抑えることが出来る。ここではジャンプを用いてそれ以降の命令のカットを行なっている。しかし，ジャンプ命令を多用するとパイプラインの分断が多発し，効率が低下してしまう。そこでジャンプ動作時にも連続的な命令供給が可能な命令供給機構を後章で提案する。

4.2.2 物理命令語のスケジューリング

前節で述べたように，論理命令語は効率的な実行を行なえるように静的にスケジュールされている。そこで命令をメインメモリからキャッシュメモリへ格納する際に行う論理命令語から物理命令語への動的なスケジューリングは，できるだけ下位アドレスから順に実行されるように行う。より効率的な実行を考えた場合，ルックアサイドバッファを設け，論理命令語の全域にわたって命令の種類を調べることによって，物理命令語にできるだけ多くの命令を割り付け，実行サイクル数をより小さくすることができる(図5 (c))。このとき，ある命令より上位のアドレスの命令が先に実行されてしまうが，実行条件によって無効化されてしまう。また，無効化される命令のみの物理命令語は増加しないので，実行効率に影響は及ぼさない。ジャンプ命令については，プログラムのコンテクストを変えてしまうので、それより下位アドレスの命令より先に実行してはならない。しかし、この方法ではハードウェアが非常に複雑になってしまう。そこで，このようなルックアサイドバッファを設けて論理命令語の全域を調べる方式ではなく，論理命令語中の下位のアドレスの命令から順に調べて物理命令語に割り付けてゆき，リソースの不足により割り付けられない命令が出現した時点で，割付けを打ち切る方式が考えられる(図5 (d))。この方法では並列性が犠牲になる可能性があるが，ハードウェアは比較的簡単になる。両方式の選択は色々な要素の間のトレードオフによって決るが，特定の機能のリソースが不足しやすい低並列（2～4並列）のマシンでは少ないリソースを有効に利用できる前者の方式が適していると考えられる。逆に，多くのリソースを持つ比較的高並列（8～16並列）なマシンでは，実行に必要な十分なリソースを提供できる可能性が高いので，後者の方式が適していると考えられる。理想的には論理命令語は無制限のリソースを仮定してスケジュールされるが，現実には効率化のために有限のリソースを意識してスケジュールされる。例えば，並列性抽出のとき，リソースを全く意識せずに常に最大の並列性を抽出しようとすると，コンパイル時間が長くなり，しかし，それに比べてそれほどの利得が得られないことが多い。そこで，マシンのファミリーの中の最大並列度のものを想定してこれを越える並列性の抽出を行なわないことで，コンパイル時間の短縮ができる。計算機はその性能によってファミリーを構成するが，そのバリエーションが数十種になることは稀で，現実には数種類である。しかも，そのファミリーの最大の並列度は，問題の持つ並列性を合理的な時間で抽出できる度合い[5]やマシンのコストの制限などから自然に決ってくる。

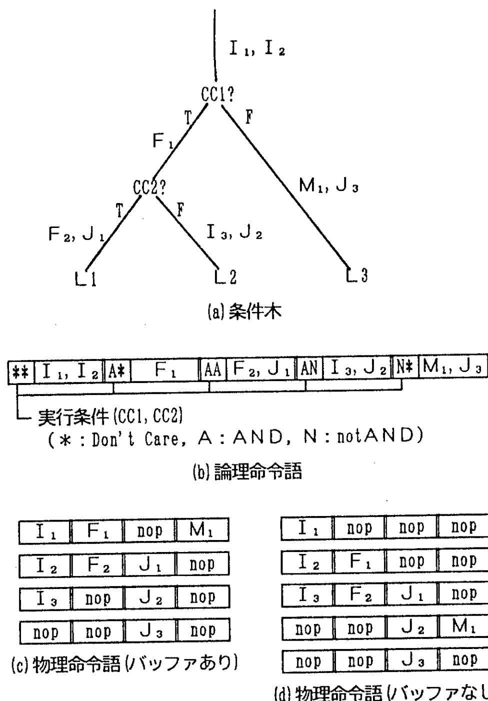


図5 論理命令語と物理命令語のスケジューリング

４．２．３ 物理命令語の格納法

動的に再スケジュールされた物理命令語は，いくつかの付加情報とともにキャッシュメモリに格納される。付加情報とはリソース識別子，命令数，次候補アドレス，連続ビットである。各リソースにはそれぞれ識別子がつけられいて，再スケジューリングの際に各命令に選択されたリソースの識別子を付加し，キャッシュメモリに格納している。従って，キャッシュメモリに格納されている物理命令語はリソース構成に依存した命令語フォーマットに適合したものではなく，実行時に投入されるリソースが決められた命令の集合となる。プロセッサに投入する時点で，それぞれの命令の持つリソース識別子に従って並べ替えられて完全な物理命令語に変換してプロセッサに投入される。命令が割り付けられなかったリソースにはダミーのＮＯＰを割り付ける。ＧＩＦＴの命令用キャッシュメモリは，連想の単位であるエントリ１つに物理命令語を１つ格納している。物理命令語を完全な形でキャッシュメモリに格納すると，プロセッサの命令語長と同じライン長が必要になる。並列度が比較的高い機能非均質型のプロセッサでは，リソース選択の自由度が低くなるために，使用されないリソースが比較的多くなる。このため，プロセッサの命令語長と同じライン長を持ってもダミーのＮＯＰが占める割合が多くなり，キャッシュメモリの利用率が悪くなる。しかし，この方式を用いると，キャッシュメモリのライン長がプロセッサの命令語長よりも短くとも命令供給ができ，これによって，キャッシュメモリの有効利用が可能となる。

命令数は物理命令語に含まれている命令数を示していて，各エントリごとに付加されている。ＧＩＦＴは可変長命令語を用いているためアドレスの増分が均一とならない。そこで引き続く物理命令語のアドレスを知るために必要となる。命令数は動的なスケジューリングの際に求められる。次候補アドレスはこれを持つ物理命令語を投入した次のサイクルで投入する物理命令語のアドレスであり，連続ビットは次のサイクルで投入する命令のアドレスが連続しているか否か，いいかえると，その物理命令語が前回の実行でジャンプ動作を行なったか否かを示している。これらは後章で述べる命令供給機構で用いている。

４．３ 二重レジスタファイル機構

通常ＶＬＩＷ方式の命令語中の命令は，同一サイクルで実行されることを前提に，静的にスケジュールされている。このため一つの命令語の中で，あるレジスタの値の読み出しと書込みを行なう命令が混在するリードモディファイライトを実現でき，並列性を高めることができる。しかし，スケーラブル化すると一つの論理命令語が，プロセッサの並列度や機能に従って，複数のサイクルに分けられて実行されることがある。このため，ある論理命令語内でのリードモディファイライトを行なうと，直前の論理命令語の実行終了時に設定されていた値ではなく，その論理命令語内の命令によって書き込まれた値を読み込んでしまう可能性があり，リードモディファイライトが実現できない。これにより，レジスタを有効に利用できなくるばかりでなく，プログラムの並列性が犠牲になり，実行サイクル数も増加させてしまう。ＧＩＦＴでは一つの論理命令語内でのリードモディファイライトを実現するために，レジスタファイルを二重化している。レジスタは値の読込みを行なうリードレジスタと，計算値の書込みを行なうライトレジスタとに分れている。論理命令語中の全命令が実行されたとき，またはジャンプ命令が実行されたとき，つまり次の論理命令語の実行の直前に，ライトレジスタの持つ値をリードレジスタにコピーする。これによって，一つの論理命令語が複数のサイクルにわけられて実行されても，直前の論理命令語の実行終了時に設定されていた値を読み込むことが可能となる。また，レジスタファイルと同様にレジスタスコアボード機構も二重化されていて，リードレジスタ用とライトレジスタ用の二つに分かれている。このため，一つの論理命令語がどのようなサイクル数で実行されようとも，その論理命令語の実行が終了するまで，リードレジスタ用のスコアボードをロックできないので，リードモディファイライトを行なってもデッドロックすることはない。

レジスタファイルに対する読込みや書込みは，図２に示すような時間で行なわれる。図２において，読込みはデコードステージの最後の1/4サイクル，書込みは実行ステージの最後の1/4サイクルで行なわれ，これらはオーバーラップしている。ライトレジスタからリードレジスタへのコピーが行なわれないサイクルでは，レジスタファイルが二重化されているので，レジスタの読込みと書込みは同時に行なっても支障がない。しかし，コピーが行なわれるサイクルでは，一旦リードレジスタにコピーした後に，読込みを要求している演算器に値を供給すると，時間が長くなり，サイクル時間を短縮することができなくなる。従って，コピーと演算器への最新の値の供給を，同時に行なわなければならない。このために，ＧＩＦＴではレジスタファイルにバイパスルートを設けている。すでにライトレジスタ格納されている値については，リードレジスタに書き込まれるのと同時に，リードレジスタをバイパスして直接に演算器の入力にも送られる。このサイクルで生成される値については，さらにライトレジスタへも書き込まれる。

５． 命令供給機構

パイプライン化されたプロセッサでは，命令のフェッチや実行などが並列に行なわれるため，命令の供給は予測的になる。通常は，プログラムの直列性から，引き続くアドレスの命令が実行されることが多いので，この命令が供給される。しかし，ジャンプが発生すると，供給アドレスが不連続になるため，予測的に供給した命令を無効にしなければならず，パイプラインが分断され，効率が低下してしまう。一般的な問題では，ジャンプが頻出するので，これによる損失が大きくなりがちである。ＧＩＦＴでは命令用キャッシュメモリに格納されている各物理命令語に，次候補アドレスと連続フラグを付加することで，ジャンプ時にもパイプラインを分断しない命令供給機構を開発した。

この命令供給機構は，次候補アドレスを決定する上で，プロセッサと非常に密接に関係していて，一体のものとして設計される。次候補アドレスとは，これを持つ命令語が実行された後に実行すべき命令語のアドレスである。命令語の供給はすべて次候補アドレスを参照して行なわれるので，通常のようなプログラムカウンタを用いた命令フェッチは行なっていない。命令用キャッシュメモリにアクセスすると，命令語と次候補アドレスが供給される。ここで命令語はプロセッサへ送られ，次候補アドレスはプロセッサに送られるのと同時に再びキャッシュアクセスに用い，命令供給が続けられる。最初，キャッシュメモリに命令語が格納される際に，次候補アドレスはその命令語のアドレスと含んでいる命令数から計算されるアドレスを持ち，この命令語でジャンプが発生したか否かを示す連続ビットはオンにセットされる。従って，初期状態では，通常のプロセッサのように，引き続くアドレスの命令語が供給される。しかし，ジャンプが発生すると，供給アドレスの連続性が崩れ，これまで予測的にパイプラインに投入した命令語を無効にして，新たにジャンプアドレスの命令語から供給しなければならず，これが損失となってしまう。そこでジャンプを発生させた命令語の次候補アドレスを

ジャンプアドレスに書き換えることによって，同一の命令語で連続してジャンプが発生した場合には，無損失の供給を可能にしている（図6 (a)）。

ジャンプ命令が実行されると，ジャンプアドレスと次候補アドレスが比較される。一致した場合はそのまま処理が行なわれるが，もし，一致しない場合，つまり，未実行命令語であるか前回ジャンプを行なわなかった，もしくは，ジャンプアドレスが前回とは異なったときは，このジャンプアドレスから命令供給を行なうよう命令供給機構へ通知するとともに，パイプラインを無効化する。この通知を受けた命令供給機構は次のサイクルでジャンプ先の命令語を供給すると同時に，ジャンプ元の命令語の持つ次候補アドレスを，このジャンプアドレスに変更し，連続ビットをオフにする。このように，ジャンプアドレスと次候補アドレスが一致した場合を「ジャンプヒット」，一致しなかった場合を「ジャンプミス」と呼ぶ。

また，前回にはジャンプを行なったが，今回は行なわない場合，次のサイクルでは引き続くアドレスの命令語実行しなければならないが，すでに供給されている命令語は前回のジャンプ先の命令語である。前回ジャンプを発生させた命令語の連続ビットはオフになっている。そこで，連続ビットはオフになっている命令語がジャンプを発生させなかったときは，引き続くアドレスの命令語を供給するよう通知するとともに，パイプラインを無効化する（図6 (b)）。この通知を受けた命令供給機構は，次のサイクルで引き続くアドレスの命令語を供給すると同時に，この命令語の持つ次候補アドレスを引き続くアドレスに変更し，連続ビットをオンにする。この場合のように，ジャンプを行なわないために次候補アドレスが実際に必要とされるアドレスと一致しなくなった場合を「非ジャンプミス」と呼ぶ。

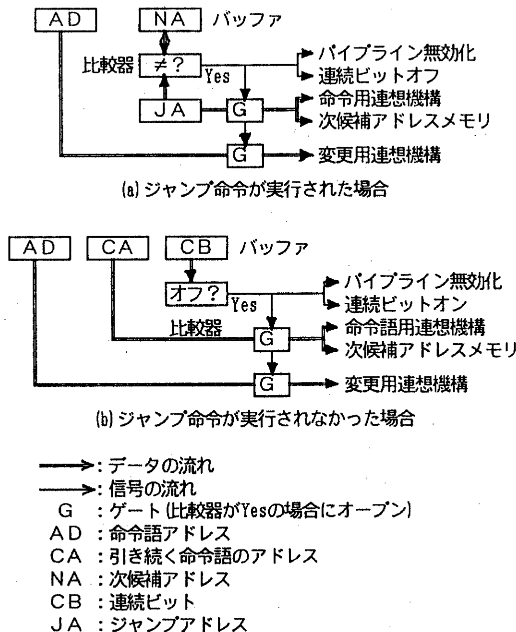
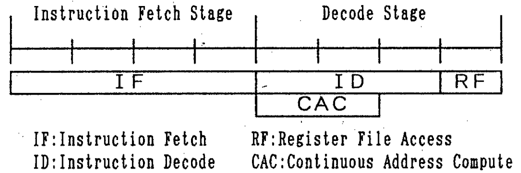


図6 命令供給機構

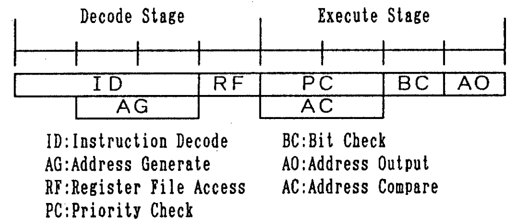
ジャンプミス時のジャンプ元の命令語のアドレスや非ジャンプミス時の引き続くアドレスを知るために，パイプライン中に存在する命令語のアドレスやこれに引き続く命令語のアドレスをプロセッサ内にバッファリングしている。また，次候補アドレスや連続ビットについても，同様にしている。ある命令語に引き続く命令語のアドレスは，命令供給時に，その命令語のアドレスとそれに含まれる命令数により計算できる。この計算は命令供給と並列に行なえるので，プログラムカウンタを用いた場合に比べても，時間的に余裕がある。また，命令供給と次候補アドレスや連続ビットの書き換えを並列に動作できるように，キャッシュメモリ中の次候補アドレスと連続ビットを記憶するメモリと連想機構は，命令供給用と変更用とでデュアルポート化がなされている。



CAC：命令語のアドレスとキャッシュメモリから供給された命令数から引き続く命令語のアドレスを算出する時間

図7 引き続くアドレスの算出タイミング

ジャンプアドレスと次候補アドレスの比較や連続ビットとジャンプの発生の有無の判断，そしてこれらの結果から導かれる命令供給機構への通知などの処理のための準備はジャンプ命令の実行と並列に行なわれる。そして，これらの処理を開始させるかどうかは実行条件の評価によって決定される。ジャンプユニットが複数ある場合には，上記の準備が各ユニットで並列に行なわれ，実行条件と優先順位の評価によって処理の開始が決定される。これによって，実行サイクル中の多く時間をこれらの処理に当てることができるので，サイクル時間の短縮のボトルネックになりにくい。



AC：次候補アドレスとジャンプアドレスを比較する時間
BC：連続ビットとジャンプの発生を比較する時間
AG：プログラムカウンタ相対ジャンプのためのジャンプアドレスを計算する時間

GIFTではプログラムカウンタ相対ジャンプとレジスタによる直接ジャンプをサポートしている。

図8 ジャンプユニットのタイミング

6． 先行制御の効率化

パーコレーションスケジューリング [6] などの大域的に命令の移動を行なうスケジューリング法により，条件分岐を越えて命令が移動した場合，本来なら実行されるべきではない命令が実行されてしまことがある。このため，使用しないデータの生成を待つことによる性能低下が生じてしまったり，除算命令の零割り等の演算エラーが発生し，プログラムの実行に支障をきたす可能性がある。したがって，高性能を維持するためにはこれらの問題を解決しなければならない。

6.1 使用しない値の生成待ち対策

条件分岐を越えて演算が先行実行されるが，その演算結果の値を使用しない場合がある。このとき，この値の生成が終了するまで次の処理が待たされたり，この演算のために処理速度に損失を与えたりする可能性がある。例えば，レジスタスコアボーディングロックやキャッシュミス，ページフォールトである。通常，レジスタスコアボードにより，同一レジスタへの演算結果の書込みはその演算の発生順序に従って行なわれることが保証される。このため，演算が先行実行された場合，使用しない値の生成を待つ可能性があり効率低下につながる。浮動小数点演算などの実行時間が知られている場合には，レジスタ割付けをうまく行なうことにより回避できる。しかし，メモリロードの場合は動的に実行時間が変化するため，事態が深刻である。先行実行ロードがキャッシュミスやページフォールトを起こし，その後，ロードした値を使用しなかった場合の損失は非常に大きい。これに対処するため，各レジスタに先行ロード用の制御ビットとしてページフォールトビット，キャッシュミスビットの2ビットを付加し，先行ロードに対して以下の様な例外処理を行なう。

6.1.1 ページフォールト時

先行ロードがページフォールトを引き起こした場合，通常のような復旧処理のための割込みを発生させず，ロード値を書き込むはずであったレジスタにロードアドレスを格納し，そのレジスタの持つページフォールトビットをオンにする。ページフォールトビットとはこれを持つレジスタの内容がページフォールトを発生させたロードアドレスであることを示すビットである。これらの処理はキャッシュヒット時のロードの実行時間と同一の時間で完了できる。この後に，このロード値を使用するためレジスタに読込みが発生した場合，ページフォールトビットをハードウェアが検出し，これによって割込みを発生させる。割込みが発生すると，実行中の物理命令語は無効化されて待機状態になり，新たに復旧処理ルーチンが実行される。復旧処理ルーチンでは割込みの原因となったレジスタから格納されているロードアドレスに従ってページフォールトを復旧する。そして，有効なロード値をそのレジスタに格納すると同時に，ページフォールトビットをオフにして復帰して，無効化された物理命令語から実行が再開される。しかし，このロード値が使用されずレジスタに新たな書込みが発生した場合には，新しい値をレジスタに書込みページフォールトビットをオフにするのみである。

6.1.2 キャッシュミス時

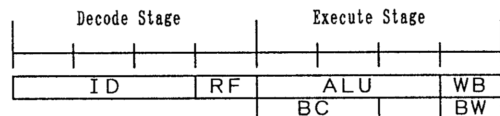
先行ロードがキャッシュミスを引き起こした場合もページフォールトの場合と同様に，即時にはその復旧処理を行なわずにロード値を使用するためにレジスタに読込みが発生した時点で，その物理命令語は無効化され待機状態になり，メインメモリをアクセスして復旧を行なう。ページフォールトの場合と異なるのは，ロードアドレスを格納したレジスタのキャッシュミスビットを操作することと，復旧処理はハードウェアで行なわれることである。キャッシュミスビットとは，これを持つレジスタの内容がキャッシュミスを発生させたロードアドレスであることを示すビットである。これは，キャッシュミスがページフォールトに比べ発生頻度が高いことや要求される復旧速度が速いためである。

6.2 演算エラー対策

演算命令が条件分岐を越えて先行実行されると，発生するはずのない演算エラーが発生する可能性がある。これは，条件分岐が除算の零割りや関数の定義域外値の代入などの演算の例外判断を行なっていた場合である。これはオリジナルのプログラムとの等価性がなくなるばかりではなく，通常これらの例外が発生すると割込みが発生しエラー処理ルーチンが実行されることになり，プログラムの実行は中断されてしまう。この中断のコストは大きく，演算命令の移動によって得られる利得をなくしてしまう可能性がある。よって，その後の処理に支障をきたさない例外処理は行なわない必要がある。

これに対処する方法として，演算結果を保持する各レジスタに「演算エラービット」を付加する。演算エラービットは，それが付加されているレジスタが保持している演算値が，生成される過程における演算エラーを発生させたか否かを示している。演算過程でエラー発生と同時に割込みによる例外処理ルーチンの実行はせず，演算結果を保持するはずであったレジスタの演算エラービットをオンにし，レジスタにはエラーコードを格納する。また，複数のサイクルにわたって先行実行を行なったときも，後のサイクルにエラーの発生を伝達するために，それぞれの命令は先行実行用と通常用の二種類の命令セットを持っている。先行実行用の命令が演算エラービットの立ったレジスタの値を読み込んだときにも，演算や割込み要求を行なわずに，演算結果を格納するはずであったレジスタの演算エラービットをオンにしてエラーコードをコピーする。これによって，並列に実行されない複数の命令も，条件分岐点を越えて効率的な先行実行が可能となる。その後の通常用の命令においてこの演算エラービットが立ったレジスタを読み込んだ時点で割込みを発生させ例外処理を行なう。また，演算エラービットがオンになったレジスタに，エラーではない値が書き込まれたときには，演算エラービットもオフにされる。これにより，先行した演算の演算エラーの発生による無意味な割込みを防げるので，実行速度を維持でき，またプログラムの持つ意味も保証できる。

これらのビットの評価や，評価結果に従って要求される処理のための準備は，実行条件の評価などと並行して行なう。処理の開始は，レジスタを使用しようとする命令の実行によって，決定される。また，これらのビットの操作はレジスタへのライトバックと同じタイミングで行なう。これによって，特別なサイクルを必要としないので，サイクル時間が引き延ばされることはない。



IF:Instruction Fetch　BC:bits Check
ID:Instruction Decode　BW:Bits Write back
RF:Register File Access　ALU:ALU Cycle
WB:Register Write Back

BC：読込みレジスタのページフォールトビットやキャッシュミスビット，演算エラービットの状態を評価する時間
BW：実行結果に従って，ページフォールトビットやキャッシュミスビット，演算エラービットを操作する時間

図9 ページフォールトビットやキャッシュミスビット，演算エラービットに関するタイミング

7. おわりに

今後の課題としては，GIFTアーキテクチャの評価を進めていくことである。現在，VHDLを用いて，GIFTのシミュレータを作成するために論理設計を行なっている。シ

ミュレータの制作途中においては，タイミング的にクリティカルな箇所の改善を行なっていかなければならないであろう。マシンの仕様は固定小数演算，浮動小数演算，ジャンプ，メモリアクセスのそれぞれの命令について４並列，計１６並列のプロセッサと１エントリ当たり８命令を格納できる命令用キャッシュメモリを持つものをファミリーの最上位マシンと考えている。これは，文献[5]の評価に基づいている。また，これに並行してパーコレーションスケジューリングを拡張し，ＳＳＡ[7]などの手法を用いた最適化コンパイラを作成している。これらを用いて，ＧＩＦＴと他のアーキテクチャとの比較やＧＩＦＴの各並列度のマシンの性能比較などを行なう予定である。

今回，命令供給機構とプロセッサについて述べたが，現在，データ用キャッシュメモリの開発も行なっている。これはバンク化されたキャッシュメモリとプロセッサを相互結合網によって接続したもので，スヌープキャッシュシステムに比べ競合による待ち時間が小さく，比較的高並列化にも対応できるものを目指している。

参考文献

[1] J.A.Fisher : "Very Long Instruction Word Architectures and the ELI-512", proc. of 10th Int. Symp. on Computer Architecture, pp.140-150 (1983)

[2] R.P.Colwell, et al. : "A VLIW Architecture for a Trace Scheduling Compiler", 2nd Int. Coff. on Architectural Support for Programming Languages and Operating Systems, pp.180-192 (1987)

[3] K.Murakami et al. : "SIMP (Single Instruction stream/Multiple instruction Pipelining) : A Novel High-Speed Single-Processor Architecture", Proc. 16th ISCA, pp.78-85, May 1989

[4] 鈴木，小松，深澤，門倉 : "条件分岐の効率的実行を可能とする細粒度並列アーキテクチャ", SWoPP'90, CPSY90-53, pp.91-96

[5] T.Nakatani, et al. : "Using a Lookahead Window in a Compaction-Based Parallelizing Compiler, IEEE Computer Society Press (1990) pp.57-68

[6] K.Ebcioglu and A.Nicolau : "A Global Resource-Constrained Parallelization Technique", ACM SigARC International Conference on Supercomputing, 1989

[7] Ron Cytron, et al. : "An Efficient Method of Computing Static Single Assignment Form", POPL 16, (1989)