計算機アーキテクチャ 133－1
（1999．5．21）

# 非同期式プロセッサ TITAC-3 の命令供給機構

藤木崇宏† Byung-Soo Choi† 小沢基一†
中村宏† 南谷崇†

より高い命令供給速度実現のためには、高い命令フェッチ幅、高い分岐予測精度と分岐先をより速く送出することが必要である。同時にフェッチしようとしている複数命令内に分岐命令があるとき、分岐先の命令を次サイクルにフェッチするのではなく、分岐先の命令も同時にフェッチすれば、より高い命令フェッチ幅を得ることができる。そこで、このように分岐を越えて命令をフェッチする方式を非同期式プロセッサTITAC-3の命令供給方式として提案する。またそれを実現するための Branch Target Buffer(BTB)、分岐予測機構、命令キャッシュの構成を提案する。

# Instruction Issue Mechanism for Asynchronous Processor TITAC-3

Takahiro Fujiki,† Byung-Soo Choi,† Motokazu Ozawa,†
Hiroshi Nakamura and Takashi Nanya†

High fetch bandwidth, accurate branch prediction and early supply of predicted target address are needed to sustain high instruction fetch rate. In conventional method, when a branch instruction is included within instructions being fetched, instructions following the branch instruction are fetched in the next cycle, which limits instruction fetch bandwidth. However higher fetch bandwidth can be accomplished by fetching instructions over branch in the same cycle.

This paper proposes a new instruction fetch mechanism for an asynchronous processor, TITAC-3 with branch target buffer(BTB), branch predictor and instruction cache.

## 1. はじめに

素子の高速化によりゲート遅延が小さくなる一方、回路面積の増大によって配線遅延は絶対的・相対的に増加している。そのため、チップ全体を単一のクロックで同期させる同期式回路では、素子の高速性を十分活かすことができない。

この問題を解決する方法の1つとしてシステムを非同期式で構成する方法がある[1]。非同期式ではチップ全体を同期させるクロックを用いず、信号遷移の因果関係のみを用いた自律的な動作でシステムを実現する。そのため、システム全体に分配するクロック信号が不要となり，素子の高速性を活かすことが可能になる。我々は既に32ビット非同期式汎用プロセッサTITAC-2[1]を試作し、稼働させている。しかし、これは単純なスカラプロセッサであり、現行の同期式プロセッサが行うような複数命令の同時発行を行っていない。そこで、我々は複数命令同時発行の非同期式プロセッサTITAC-3[2]を開発している。本論文では、TITAC-3の命令供給機構について述べる。

## 2. 命令供給方式

スーパスカラ・マシンにとって、分岐による命令流の分断は大きな性能低下を引き起こす。性能向上のために通常分岐予測を行うが、分岐の方向と分岐先をより速く、より正確に予測することが重要である。従来の分岐先バッファ(Branch Target Buffer(BTB))[3]は、同時にフェッチしようとしている複数命令内に分岐命令がある場合、分岐先の命令は次のサイクルにフェッチしていた。しかしこの方法では、分岐命令以降の命令は同時にフェッチできないため、分岐によってフェッチ幅が制限されてしまう。しかし分岐命令でフェッチ幅を区切るのではなく、分岐先の命令も同時にフェッチすれば、より高いフェッチ幅を得ることができると考えられる。

そこで、このような分岐を越えて命令をフェッチする命令供給方式について考える。

複数の分岐を越えた命令列を同時に発行する方式として、*Collapsing Buffer*を用いた方法がある[4]。この

† 東京大学先端科学技術研究センター
Research Center for Advanced Science and Technology, The University of Tokyo

方法では、複数の分岐を越えた命令列をフェッチするために、BTBを複数回参照して、複数の分岐予測を同時に行い、複数のBasic Blockをフェッチ、連結して1つの命令列を作っている。この方法で問題となるのは、複数の分岐を越えた1つの命令列を作るのに大きなコストと時間がかかってしまうことである。その問題を解決する方法として、1度作った命令列をキャッシュに保持して再利用する *Trace Cache*[5] がある。

しかし TITAC-3 の命令実行部[2] は

- 4命令同時発行
- 同時に発行できる分岐命令数は1

であるので、*Collapsing Buffer*を用いた方法のように複雑な構成にする必要はない。1度にフェッチする命令列は、分岐命令までの命令列と分岐先の命令列の2つの命令列である。分岐先の命令列が分岐命令を含んでいれば、その分岐命令はフェッチしない。

分岐命令までの命令列と分岐先の命令列を同時にフェッチするため、従来のBTB[3]では、分岐先の分岐情報を調べるためにBTBを2度参照する必要があった。しかし1度の参照で分岐先の分岐情報がわかるように従来の BTBに分岐先の分岐情報を付加すれば、2度参照する必要はなくなる。

そこで、TITAC-3 の命令供給機構として、以下の方式を提案する。

- 発行しようとする4命令中に分岐命令が含まれるとき、分岐先の命令も同時に発行 (ただし、1度に発行する分岐命令は1つまで)
- BTBは分岐命令とその分岐先の情報だけでなく、分岐先にある命令の分岐情報も保持 (3.1節)
- より高い精度の分岐予測をするためにBTBと分岐予測機構を分割
- 分岐予測と命令フェッチを並列に行うために、命令供給機構を2つのステージに分割

## 3. 命令供給機構の構成

非同期式プロセッサ TITAC-3 の命令供給機構は、Branch Target Buffer、分岐予測機構、2-port の命令キャッシュから構成される (図1)。

**Stage1** 発行しようとする4命令内に分岐命令があるかBTBで調べ、分岐命令があれば、そのアドレスと分岐先アドレスを出力する。またそれと並行して分岐予測を行い、その結果から Address Generator が命令キャッシュからフェッチするラインを指定する。

**Stage2** Stage1で指定されたラインを命令キャッシュからフェッチし、必要な命令のみ取り出し、並び替えて (Select and Align) 発行命令列を生成する。

### 3.1 Branch Target Buffer(BTB)

通常のBTB[3] は

- これから発行しようとする命令内に分岐命令があるか調べる。
- 分岐命令が含まれるならば分岐先のアドレスを出力する。

の2点を実現する情報を保持するバッファである。

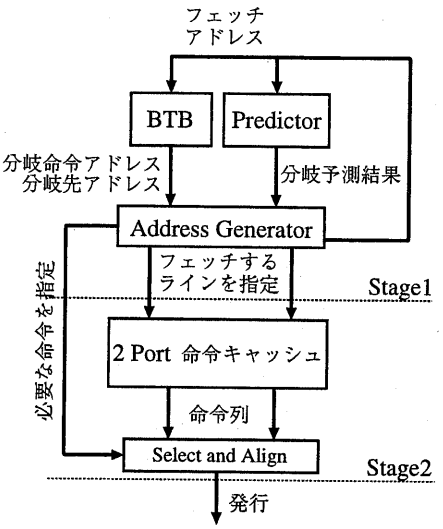


**図1** 命令供給機構の概略図

本稿では、分岐命令までではなく、分岐先の命令も同時にフェッチする手法を提案しているが、上のような BTBの構成では、1度の参照で分岐先に分岐命令があるかわからない。

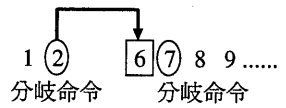


**図2** 命令の流れの例

具体的な例をとって説明する (図2)。分岐命令2が6に分岐するとき、分岐先の命令列も同時にフェッチしたいので命令6は命令1、2と一緒にフェッチする。しかし4命令内に分岐命令は1つまでなので分岐命令7はフェッチできない。1回参照するだけで命令2が分岐命令、その分岐先が命令6、命令7は分岐命令であるとわかるようなBTBを実現することを考える。

そこで、BTBのエントリに分岐先の命令から次の分岐命令までの距離 (命令数) という情報を加えることを考える。図2の例でいえば、分岐先6の1命令後に分岐命令7があるという情報である。この情報を付け加えれば、1回の参照で分岐先の命令列内の分岐情報を知ることができる。さらに発行命令幅は最大4なので、分岐先から後続の3命令までに次の分岐命令があるかわかれば良いので、分岐先から4番目の命令以降は見る必要がない。

そこで、以下のエントリを持つBTBを提案する。

**Address Tag** 分岐命令のTag
(Tagが一致すればその命令は分岐命令である)
ダイレクトマップ方式を考えているので、アドレスが$m$ビットで表現されるとき、BTBが$2^n$エントリならば、Tagは$(m-n)$ビットとなる。

**Branch Type** 分岐命令の種類(無条件分岐、条件分岐、間接分岐)
3種類の分岐命令の判別のため2ビット必要。

**Target Jump Address** 分岐先アドレス
我々が実現するMIPS-IIアーキテクチャではoffset 16ビットで表現される。

**Offset of Next Branch** 分岐先の命令から次の分岐命令までの距離
- 分岐先から後続の3命令内に次の分岐命令の位置がわかれば良いので2ビットあれば良い。
- 分岐の成立・不成立それぞれの場合のエントリが必要である。

したがって **Offset of Next Branch**は、2ビット×2必要である。

上のBTBは、分岐先の分岐情報**Offset of Next Branch** 4ビットを付け加えただけで、従来のBTBでは2度の参照でしか得られなかった情報を1度の参照で得ることができる構成となっている。

また**Offset of Next Branch**は、分岐の成立・不成立、2つのパターンがあるので、どちらかのエントリが空である状態が存在する。その場合は、BTBにエントリがあっても(BTBヒット)、**Offset of Next Branch**が空であるために、4命令内に分岐命令が2つ以上含まれてしまう場合が存在する。

### 3.2 分岐予測機構(Predictor)

#### 3.2.1 条件分岐予測

条件分岐予測機構は条件分岐の分岐方向(成立・不成立)を予測するものである。ただし1サイクルに1分岐しか予測しない。

条件分岐の予測方法として以下のものを考える。

- 2ビット飽和型アップダウンカウンタ方式[3)]
  分岐命令ごとに2ビットのカウンタを持ち、カウンタの値により分岐方向を予測する。分岐が成立すればカウンタの値を増やし、不成立のときはカウンタの値を減らす。BTBのエントリに2ビットのカウンタを付け加えることで実現できる。
- Gshare方式[6)]
  予測する分岐命令アドレスとグローバル分岐履歴をXORしたものでパターン履歴テーブル(2ビットカウンタのテーブル)を参照し、その値により分岐方向を予測する。
- ハイブリッド方式[6)7)]
  2ビット飽和型アップダウンカウンタ方式とGshare方式を組み合わせる。またそれらとは別に選択テーブルを用意して、それを用いてどちらかの予測方法を動的に選択する[7)]。

**ハイブリッド方式の問題点**

ハイブリッド方式には、分岐命令のアドレスを与える必要がある。そのためには、図3のようにBTBとPredictorを逐次的に参照する必要があり、1ステージのレイテンシが長くなるという問題が生じる。

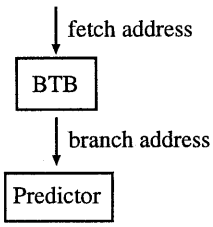


図3 逐次的な参照

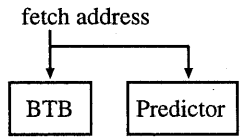


図4 並列な参照

そこで図4のようにフェッチアドレスをPredictorの入力にし、BTBとPredictorを並列に参照することで、ステージ間のレイテンシを短くすることを考えた。

しかし一方で、フェッチアドレスから分岐予測することによる予測精度の低下が懸念される。なぜならフェッチアドレスはフェッチする4命令の先頭のアドレスであって、必ずしも分岐命令のアドレスでなく、同じ命令シーケンスを実行しても、フェッチアドレスそのものが前回実行時のフェッチアドレスとは異なる場合があるからである。

そこで、前後するだろう不確かな下位ビット(4命令発行なので高々2ビット)を切り捨てて、切り捨てたビット分を上位ビットに加えて入力情報とすることも考えられる。そこで、Predictorに与える入力の候補として、図5のようにフェッチアドレスから3通りの生成法があると考え、4.1.1節で評価する。

#### 3.2.2 間接分岐の分岐先予測

分岐する分岐先のアドレスがレジスタの値に依存する間接分岐命令は、他の分岐命令と違い、分岐先が一意に決まらず、レジスタの値を見るまで分岐先のアドレスが確定しない。

間接分岐の出現頻度は、ベンチマークプログラムによって異なるが、決して少なくない。そこで間接分岐の分岐先予測も条件分岐の分岐予測とは別に予測方法を考える必要がある。間接分岐の分岐先予測方法として以下の2つの方法が提案されている。

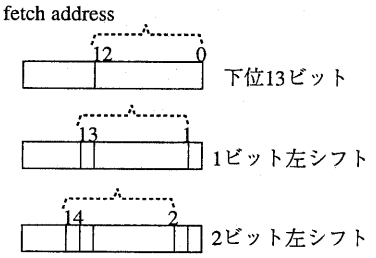


図5　入力とするフェッチアドレスの取り方(例:13ビット)

**BTBを用いた方法**　次の分岐先は、その間接分岐命令の前回の分岐先と同じであると予測する方法で、BTBにある分岐先のエントリをそのまま予測分岐先アドレスとして使う。

**Chang[8]の方法**　間接分岐命令のアドレスとグローバルな分岐の履歴をXORしたもので、パターン履歴テーブルを引く。パターン履歴テーブルには分岐先のアドレスがエントリされていて、それを分岐先として予測する方法。BTBとは別にパターン履歴テーブルが必要になる。

4.1.2節で上の方法の記憶テーブルサイズと性能の関係を評価する。

### 3.3 Address Generator

BTB、Predictorの出力情報から、図1のAddress Generatorは、フェッチする4命令内に分岐命令があれば、分岐命令までの命令列が含まれる命令キャッシュのラインと分岐先の命令が含まれるラインの2つのラインを指定する。あるいはフェッチする命令列がライン境界を越えて2ラインにまたがる場合はその命令列が含まれる2ラインを指定する。

また、4命令内にある分岐命令の位置、次の分岐命令までの距離から、発行すべき命令を決定(必要な命令を指定(図1))し、それを**Select and Align**(3.5節)部に伝える。そして次のフェッチアドレスを決定する。

### 3.4 命令キャッシュ

分岐命令までの命令列と分岐先の命令列を同時にフェッチして4命令幅にするためには、分岐命令までの命令列と分岐先の命令列がそれぞれライン境界を越えるときがあるので、最大4ラインフェッチしなければならない。しかし、4ラインフェッチしなければならない頻度はそれほど多くないと考えられるので、設計の容易さ、コストの面からもバンク競合のない1バンク2-portのキャッシュを考えている。この点について、4.3節で評価する。

### 3.5 Select and Align

最後にキャッシュからフェッチしたラインの中から有効な命令を取り出し、正しい順序に並び替えて4命令を発行する。

## 4. 性能評価

評価には表1のベンチマークのダイナミックトレースを用いる。コンパイラは IRIX6.3のMIPS Pro7.2 compilerを使用した。

(compile option -O2 -non_shared -mips2)

表1　ベンチマークの種類と命令数

| benchmark | 全命令数 | 分岐命令数 |
|---|---|---|
| Dhrystone | 5959996 | 1379999 |
| Mpegtest | 4828723 | 728883 |
| 099.go | 10000118 | 1639806 |
| 124.m88ksim | 9999225 | 2300901 |
| 126.gcc | 10000115 | 2086016 |
| 129.compress | 10016788 | 1904588 |
| 130.li | 10000004 | 2347782 |
| 132.ijpeg | 10011204 | 718376 |
| 147.vortex | 10050166 | 1783830 |

以後の評価結果は表1の全てのベンチマークの算術平均をとったものである。

### 4.1 分岐予測機構(Predictor)の評価

#### 4.1.1 条件分岐の予測精度

3.2.1節で述べた条件分岐予測方法の精度を表1のベンチマークを用いて調べた。

評価を行った予測方法は、

- 2ビット飽和型アップダウンカウンタ方式
- 分岐命令アドレスを入力としたハイブリッド方式
- フェッチアドレスを入力としたハイブリッド方式
- フェッチアドレスの1ビット左シフトしたものを入力としたハイブリッド方式
- フェッチアドレスの2ビット左シフトしたものを入力としたハイブリッド方式

である。

各予測方法の予測精度の評価を図6に示す。

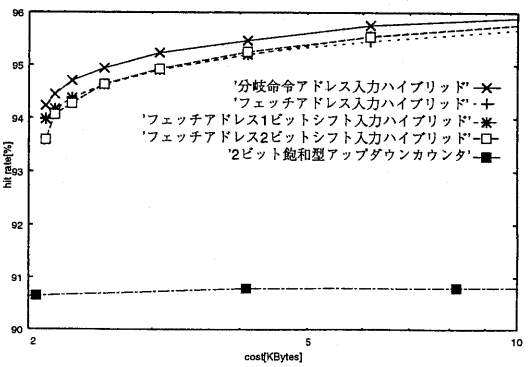


図6　条件分岐予測方法の予測精度の比較

ハイブリッド方式の評価においては、2ビット飽和型

アップダウンカウンタ予測部のテーブル及び選択テーブルのサイズは各々1KBに固定し、Gshare予測部では、nビットの分岐履歴と分岐命令アドレスの下位nビットをXORするものとし、Gshare予測部のnのみを変化させている。したがって、図6の横軸のコスト[KBytes]は、

- 2ビット飽和型アップダウンカウンタ方式ではテーブルのエントリ数
- ハイブリッド方式では分岐履歴nの長さ

で決まる。

フェッチアドレスを入力としたハイブリッド方法は、分岐命令アドレスを入力とした予測方法より精度にして0.2%ほど低いが、2ビット飽和型アップダウンカウンタ方式単独を用いるよりも精度は4%ほど高い。入力とするフェッチアドレスは、ビットシフトしても精度の差は非常に小さいが、フェッチアドレスの2ビット左シフトを入力とする方法が少しだけ精度が良い。

#### 4.1.2 間接分岐の予測精度

3.2.2節で述べた間接分岐予測方法の評価を行う。予測方法は、BTBによる予測方法とChangの予測方法である。Changの予測方法は、n個の分岐履歴と分岐命令アドレスの下位nビットをXORして予測を行う。

予測精度の評価を図7に示す。

図7の横軸のコスト[KBytes]は、BTBによる予測方法ではBTBのエントリ数で決まり、Changの予測方法では分岐履歴の長さでnで決まる。ただしChangの方法のコストは、BTBの4KBytes(固定)を加えたものである。

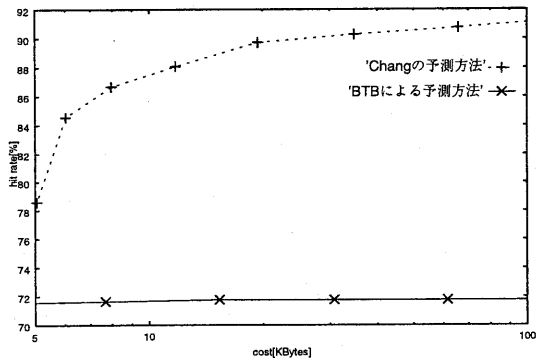


図7 間接分岐の予測精度

Changの予測方法はBTB単独のものに比べてかなり精度が良い(コスト7KBytesで約13%の予測精度向上)。

BTB単独で予測する方法はコストをいくらかけてもあまり精度は向上していない。したがって予測方法自体に問題があると考えられる。

Changの予測方法はBTBとは別の予測機構が予測を行っているので3.2.1節の図3のようにBTBと逐次的に参照するとそのレイテンシが問題になる。しかし条件分岐予測と同様に、図4のように並列参照すればその問題は回避することができる。

### 4.2 BTBの評価

次にBTB(Predictorを含めた)のInstruction Per Cycle(1サイクル当りの実効命令供給数、以下IPC)の評価を行う。

**評価条件**

評価条件は以下の通り。

- 分岐予測機構のコストは7KBytes。
  - 条件分岐予測方法は、フェッチアドレスの2ビット左シフトを入力としたハイブリッド方式(3.2.1節)で、コストは4KBytes (Gshare 2KB,2bit Counter 1KB, 選択テーブル1KB)。
  - 間接分岐予測方法はフェッチアドレスの2ビット左シフトを入力としたChangの方法(3.2.2節)でコストは3KBytes。
- 分岐予測のミスペナルティは3サイクル。
- BTBがミス(エントリがない)のときは、
  - 分岐予測は行わない。
  - 無条件分岐であればペナルティは2サイクル。
  - 間接分岐であればペナルティは3サイクル。
  - 条件分岐であれば
    - 分岐成立時のペナルティは3サイクル。
    - 分岐不成立時のペナルティなし。
- BTBのエントリはヒットだが、**Offset of Next Branch**(次の分岐命令までの距離)(3.1節)がミスで、そのために4命令内に分岐命令を2つ含んでしまった場合はペナルティ2サイクル。

以上のミスペナルティは、Decode Stageでミスが判明するものは2サイクル、Execute Stageでミスが判明するものは3サイクルという仮定に基づいている。

#### 4.2.1 BTBのエントリ数に対するIPCの評価

最大命令発行幅が4であるからといって、IPCの上限値は4ではない。なぜなら4命令内に分岐命令は1つまでという制約があるからである。そこでこの制約によって決まるIPCの上限値を表1のダイナミックトレースを用いて計算してみると、その上限値は**3.677**であった。

次にBTBのエントリ数を替えたときのBTBのヒット率とIPCの評価結果を表2に示す。

表2 BTBのエントリ数とBTBのヒット率、IPCの評価結果

| BTBエントリ | BTBヒット率 | IPC | IPC/上限値 |
|---|---|---|---|
| 512 | 90.275 | 3.002 | 0.817 |
| 1024 | 94.336 | 3.115 | 0.847 |
| 2048 | 96.643 | 3.184 | 0.866 |

表2より、今回提案した方法では、上限値の約85%の性能が得られることができた。

#### 4.2.2 Offset of Next Branchの効果

本稿で提案したOffset of Next Branch(3.1節)は、分岐を越えて命令をフェッチするために必要なBTBの情報である。

そこでこのOffset of Next Branch (BTB1エントリ当り4ビット)の効果を調べるために、Offset of Next Branchがない場合、つまり分岐命令までしかフェッチしない場合のIPCを調べた(表3)。

表3　分岐命令までフェッチする方法のIPCの評価結果

| BTBエントリ | IPC | 表2のIPCとの比 |
|---|---|---|
| 512 | 2.821 | 0.939 |
| 1024 | 2.914 | 0.935 |
| 2048 | 2.968 | 0.932 |

表3のOffset of Next Branchを用いない分岐までしかフェッチしない方法は、Offset of Next Branchを用いて分岐を越えて命令をフェッチする方法(表2)に比べて約6%から7%の性能低下が見られた。

### 4.3 2-port命令キャッシュの命令フェッチ幅の評価

2-port命令キャッシュは3.4節で述べたように、4命令内に分岐が含まれ、その命令列がライン境界を越える場合は4命令を同時にフェッチできない。しかし、その頻度が少なければ性能の低下は小さい。またラインサイズが大きければ、ライン境界を越える頻度は少なくなると考えられる。

そこで、2-port命令キャッシュが、理想的な命令フェッチ幅(ライン境界をまたぐことによる制限をうけない)に比べてどれくらいの効率で命令をフェッチできるか、ラインサイズを替えて調べた(図8)。また1-portの評価も行った。ここではキャッシュヒット率は100%と仮定した。

図8の横軸はキャッシュのラインサイズ(ワード)、縦軸は命令フェッチ幅を命令フェッチ幅の上限3.677(4.2.1節)で割ったものである。

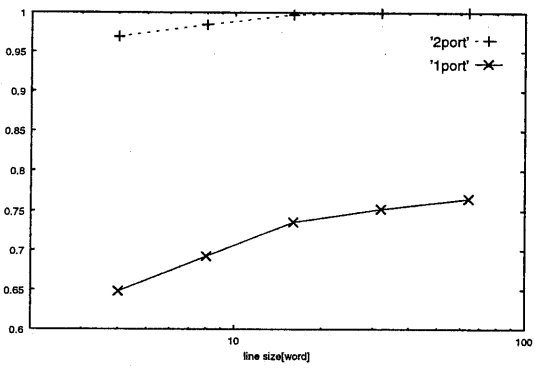


図8　2-portと1-port命令キャッシュの命令フェッチ効率

図8より2-portの場合、ラインサイズが4ワードでも理想的な命令フェッチ幅に対して95%以上の効率で命令フェッチ幅が得られることがわかる。したがって、2-portで十分な命令フェッチ幅が得られるといえる。

## 5. ま と め

- 非同期式プロセッサ TITAC-3 の命令供給機構の構成方法を示した。
  - 分岐でフェッチ幅を区切るのではなく、分岐先の命令も同時にフェッチする方式を示し、それを実現するためのBTBの構成方法を示した。
  - BTBと並列に参照することができる分岐予測機構の構成方法を示した。
  - 命令キャッシュの適切な方式を決定した。
- ダイナミックトレースを用いてBTB、分岐予測のヒット率を調べ、それらを用いた命令供給機構の評価を行った。

本研究の一部は科学研究費補助金(基盤研究(B)09480049)、及び(株)半導体理工学研究センターとの共同研究によるものである。

## 参 考 文 献

1) Akihiro Takamura, M. Kuwako, M. Imai, T. Fujii, M. Ozawa, I. Fukasaku, Y. Ueno and T. Nanya, "TITAC-2: An asynchronous 32-bit microprocessor based on Scalable-Delay-Insensitive model," *Proc. of ICCD*, Oct. 1997, pp.288-294.

2) 小沢基一, 中村宏, 南谷崇, "非同期式プロセッサTITAC-3の命令実行機構," 第58回 情報処理学会 全国大会 2H-4, Mar. 1999.

3) J. K. F. Lee, and A. J. Smith, "Branch Prediction Strategies and Branch Target Buffer Design," *IEEE Transactions on Computer*, Volume 17, Number 1, Jan. 1984, pp.6-22.

4) T. Conte, K. N. Menezes, P. M. Mills, and B. A. Patel, "Optimization of Instruction Fetch Mechanisms for High Issue Rate," *Proc. of ISCA-22*, June 1995.

5) E. Rotenberg, S. Bennett, and J. E. Smith, "Trace Cache:A Low Latency Approach to High Bandwidth Instruction Fetching," *Proc. of MICRO-29*, 1996.

6) Scott Mcfarling, "Combining Branch Predictors," *WRL Technical Note TN-36, Digital Equipment Corporation*, June 1993.

7) Po-Yung Chang, Eric Hao and Yale N. Patt, "Alternative Implementations of Hybrid Branch Pedictors," *Proc. of MICRO-28*, Nov. 1995.

8) Po-Yung Chang, Eric Hao and Yale N. Patt, "Target Prediction for Indirect Jump," *in Proc. of MICRO-24*, pp.274-283, June 1997.