Vol. 40 No. 5 情報処理学会論文誌 May 1999

MULHI キャッシュ：VLIWプロセッサのための命令キャッシュ機構

仲池卓也[†] 阿部孝之[†] 大庭信之[††]
小林広明[†] 中村維男[†]

コンパイラによる高度な命令レベル並列性の抽出により高性能を達成するVLIWプロセッサが，次世代プロセッサアーキテクチャとして近年注目を集めている．VLIWプロセッサでは，並列実行可能な複数の演算操作からなる非常に長い命令を高速にフェッチするために，高ヒット率，高バンド幅の命令キャッシュが必要不可欠である．一般に，VLIW命令中には多くのnop（no operation）が含まれるために，nopを含んだVLIW命令を命令キャッシュに格納すると，命令キャッシュの使用効率が低下し，命令のキャッシュミス率が増加する．そこで，本論文では，VLIWプロセッサのための新たな命令キャッシュ機構としてMULHI（MULtiple HIt）キャッシュを提案し，SPEC95ベンチマーク中のいくつかのプログラムを用いて性能評価を行う．性能評価の結果，MULHIキャッシュは，nopを含んだVLIW命令をそのまま格納する従来の命令キャッシュ機構に比べて，最大1.68倍の性能向上を示した．

MULHI Cache: An Instruction Cache Mechanism for VLIW Processors

TAKUYA NAKAIKE,[†] TAKAYUKI ABE,[†] NOBUYUKI OOBA,[††]
HIROAKI KOBAYASHI[†] and TADAO NAKAMURA[†]

VLIW (Very Long Instruction Word) processors, which are expected to be a next generation high performance microprocessor architecture, need a high-bandwidth, high-hit-rate instruction cache to fetch VLIWs and issue operations of each VLIW to function units quickly. However, when VLIWs including many nops (no operations) are stored in a conventional instruction cache, the cache utilization is not high, resulting in the performance degradation of VLIW processors. In this paper, a new instruction cache mechanism for VLIW processors, named MULHI (MULtiple HIt) cache, is proposed and evaluated using several programs in the SPEC95 benchmark suite. The experimental results indicate that the MULHI cache achieves 1.68 times higher performance than a conventional instruction cache that stores VLIWs with nops.

1. はじめに

近年，ハードウェアの製造技術の発展により，多数の機能ユニットを備えるCPUを1チップで実現可能になった．それにともない，プログラム中の命令レベル並列性を利用したマイクロプロセッサの命令発行幅は，現在増加の傾向にある．ただし，命令発行幅を増加させることによって，アプリケーションプログラムの実行サイクル数を減少させるためには，単位時間あたりの命令供給量を増加させる必要がある．したがって，複数の命令を同時に実行することによって性能を向上させることを狙ったVLIW（Very Long Instrution Word）プロセッサ[1]やスーパスカラプロセッサ[2]では，多数の命令を高速に機能ユニットに供給する技術が非常に重要である．

VLIWプロセッサは，並列実行可能な複数の演算操作から構成されるVLIW命令を1命令とし，その命令を複数の機能ユニットで実行することによって，プログラム全体の実行サイクル数を短縮させることを狙っている．したがって，1 VLIW命令の長さは，1演算操作を1命令とした従来のマイクロプロセッサが使用する命令長の数倍になる．従来のマイクロプロセッサ同様，VLIWプロセッサにおいても，性能を向上させるために命令キャッシュが必要である．しかし，

† 東北大学大学院情報科学研究科
Graduate School of Information Science, Tohoku University
†† 日本アイ・ビー・エム株式会社東京基礎研究所
IBM Research, Tokyo Research Laboratory, IBM Japan Ltd.

1 VLIW命令の長さが長いため，命令キャッシュからVLIW命令を高速にフェッチするためには，従来のマイクロプロセッサが使用する命令キャッシュのバンド幅では不十分である．

一般に，現在のキャッシュは，キャッシュエントリの競合を避けるために，複数のキャッシュウェイを持つセットアソシアティブ構成となっている．セットアソシアティブキャッシュでは，複数のキャッシュウェイから複数のデータが出力され，マルチプレクサに送られる．そして，キャッシュがヒットした場合，マルチプレクサによって，1つのデータが選択されて出力される．VLIWプロセッサにおいて，セットアソシアティブキャッシュを命令キャッシュとして使用する場合，命令を1サイクルでフェッチするためには，各キャッシュウェイから出力されるビット数を1 VLIW命令の長さに合わせる必要がある．しかし，現在のVLSI技術では，以下の2つの理由により，各キャッシュウェイから出力されるビット数を大きくし，1 VLIW命令の長さに合わせることが困難である．

- キャッシュを構成するSRAMのデータポートを製造プロセスの最小配線ピッチで構成することが困難[3)]
- 「キャッシュのウェイ数 × 1 VLIW命令の長さ」の出力ビット数が必要

これらの理由により，複数のキャッシュウェイからマルチプレクサへの配線が複雑になり，配線面積，電力消費，発熱量の増加などの問題が生じる．その結果，4ウェイ以上のセットアソシアティブ構成では，現在の0.25 μmの製造プロセスの場合，各キャッシュウェイからの出力ビット数は128程度が現実的な設計と考えられ，その結果，バンド幅もプロセッササイクルあたり128ビット程度となる．たとえば，現在実用化されているスーパスカラプロセッサの命令キャッシュのバンド幅は128ビットである[4)]．

1 VLIW命令の長さは，1 VLIW命令に含まれる演算操作の数や，1演算操作に必要なビット数が増加すれば，128ビット以上になると考えられる．したがって，命令キャッシュの各ウェイからの出力ビット数を1 VLIW命令の長さに合わせることが困難となる．また，将来VLSI技術が向上し，各キャッシュウェイからの出力ビット数を増加させることが可能となった場合でも，命令レベル並列性の抽出技術の発展により1 VLIW命令の長さが増加すれば，同様の問題が起こると考えられる．この場合，1 VLIW命令のフェッチに数サイクルが必要になり，プログラムの実行サイクル数が増加し，VLIWプロセッサにおける性能向上の妨げになると考えられる．

VLIWプロセッサにおける命令キャッシュ機構のもう1つの問題点として，コード中に含まれるnop（no operation）の扱い方があげられる．1 VLIW命令は，複数の機能ユニットを制御するための複数のフィールドから構成され，コンパイラが並列実行可能な演算操作を各フィールドに割り当てる．並列実行可能な演算操作がないフィールドには，nopが割り当てられる．しかし，nopを含んだVLIW命令をメモリやキャッシュに格納すると，メモリやキャッシュにおける非nop命令の占める割合（キャッシュ使用効率）は他のアーキテクチャと比較して低くなる．特に，容量制限が厳しいオンチップキャッシュでは，このことはキャッシュミス率の増加につながる．この問題点に対し，VLIW命令の各フィールドに命令の境界を表すビットを付加し，キャッシュにnopを格納しない手法が，TMS320C62x/C67xアーキテクチャ[5)]で用いられている．しかし，このような場合，キャッシュ中の各VLIW命令の長さが異なるため，命令フェッチ機構が複雑になり，命令フェッチに要する時間が増加する可能性がある．

これら2つの問題点を解決するために，本論文では，VLIWプロセッサのための効果的な命令キャッシュ機構として，MULHI（MULtiple HIt）キャッシュを提案する．MULHIキャッシュは，従来のキャッシュのセットアソシアティブ構成を利用し，VLIW命令中の各演算命令を異なるウェイに格納することによって，キャッシュ中のnopを減少させる．また，MULHIキャッシュでは，1 VLIW命令が分割されて複数のキャッシュウェイに格納されるため，各キャッシュウェイからの出力を1 VLIW命令の長さに合わせる必要がない．さらに，並列にアクセス可能な各ウェイから，同時に並列実行可能な演算命令がフェッチされるため，各キャッシュウェイからの出力ビット数が小さい場合でも，1度に多くの演算命令を読み出すことが可能となる．本論文では，MULHIキャッシュの性能を，ミス（ヒット）率だけではなく，ヒット時の動作，およびキャッシュライン置き換えの動作の詳細なサイクル数を考慮し，SPEC95ベンチマーク中のいくつかのプログラムを用いて評価する．そして，従来の命令キャッシュ機構との比較，および検討を行う．

2章では，最初にVLIWプロセッサにおける命令キャッシュ機構の性能決定の要因について述べ，次に従来のVLIWプロセッサ用命令キャッシュ機構について述べる．3章では，本論文で提案する命令キャッシュ機構であるMULHIキャッシュについて述べる．4章

では，SPEC95 ベンチマークの *compress*, *li*, *mgrid*, *turb3d* を用いて性能評価を行い，ミス率，1 サイクルで実行された演算操作の数，およびハードウェア量に関して，従来の命令キャッシュ機構と MULHI キャッシュを比較する．5 章は結論である．

2. 従来の VLIW プロセッサ用命令キャッシュ機構

2.1 VLIW プロセッサ用命令キャッシュの性能決定要因

VLIW プロセッサの命令キャッシュ機構で特に重要視される点は，ミス率（ヒット率），およびヒットの際に命令を機能ユニットに発行するまでの時間である．

VLIW 命令は，コンパイラによって抽出された並列実行可能な複数の演算操作から構成される．これまでに，トレーススケジューリング[6] やスーパブロックスケジューリング[7] などのコンパイラ技術によって，抽出される並列実行可能な演算の最大数が増加している．この並列実行可能な演算の最大数の増加にともない，VLIW プロセッサにおける並列動作可能な機能ユニットの数が増加している．その結果，プログラム中の命令レベル並列性を最大限に利用するため，VLIW 命令の長さが長くなってきている．しかし，VLIW 命令の長さが長くなると，並列実行可能な演算数が少ない VLIW 命令中の nop の数が増加し，プログラム中の nop の割合が増加する可能性がある．プログラム中の nop の割合が増加すれば，命令キャッシュの使用効率が低下し，ミス率が高くなる．また，VLIW 命令が長くなると，命令キャッシュのバンド幅が小さい場合，命令フェッチに多くのサイクル数が必要になる．

一般に，命令キャッシュのミス率は，キャッシュサイズを増加させれば減少する．現在の商用のマイクロプロセッサにおけるオンチップの一次命令キャッシュサイズは 8〜32 KB であり，多くのアプリケーションプログラムにおいて，32 KB のキャッシュサイズで低いミス率を得ることが可能である[8]．しかしながら，VLIW 命令は，従来のマイクロプロセッサの命令よりも長く，かつ多くの nop を含む傾向にあるため，従来のマイクロプロセッサと同程度のキャッシュミス率を達成するためには，キャッシュのサイズをより大きくするか，もしくはキャッシュ中の nop の割合を減少させ，命令キャッシュの使用効率を向上させることが必要になる[9]．

従来のマイクロプロセッサでは，命令長が比較的短いために，命令キャッシュのバンド幅を命令長に合わせて設計することが可能である．その結果，命令キャッシュにヒットした場合は，命令を機能ユニットに発行するまでの時間を 1 サイクル程度にすることができる．しかしながら，VLIW プロセッサでは，VLIW 命令に適した広いキャッシュバンド幅を用意することが不可能な場合，命令キャッシュのヒット時においても，機能ユニットに命令を発行する時間が大きくなる可能性がある．本論文では，命令キャッシュから 1 VLIW 命令を読み出すまでに要するサイクル数を T_{fetch} と表す．

VLIW プロセッサでは，命令に機能ユニットに応じたフィールドを設定することが可能である場合，フェッチした命令をそのまま機能ユニットに発行することが可能である．しかし，TMS320C62x/C67x アーキテクチャのように，VLIW 命令中から nop を削除し，VLIW 命令の長さを可変にすると，命令キャッシュの使用効率は向上するが，VLIW 命令中の各演算操作命令が機能ユニットに 1 対 1 に対応しなくなる．そのため，VLIW 命令に含まれる各演算操作命令を適切な機能ユニットに分配する機構が必要になり，機能ユニットに命令を発行する時間が大きくなる可能性がある．本論文では，各演算操作命令を適切な機能ユニットに分配するために要するサイクル数を T_{expand} と表す．

以上をまとめると，VLIW プロセッサにおける命令キャッシュ機構では，以下の 3 つの点が性能を決定する要因となる．

- ミス率（ヒット率）
- T_{fetch}
- T_{expand}

本論文では，T_{fetch} と T_{expand} の値を使って，キャッシュにヒットした際に，命令を機能ユニットに発行するまでのサイクル数 T_{issue} を以下の式で定義する．

$$T_{issue} = T_{fetch} + T_{expand} \quad (1)$$

ここで，T_{fetch} は以下の式で表される．

$$T_{fetch} = \frac{L}{W} \quad (2)$$

式 (2) で，L は 1 VLIW 命令の長さであり，W は命令キャッシュのバンド幅である．

本節では，以上で述べた点に注目して，従来の命令キャッシュ機構（以下，NOP キャッシュと呼ぶ），TMS320C62x/C67x アーキテクチャで使用されている nop を格納しない命令キャッシュ機構（以下，COMPRESS キャッシュと呼ぶ）の構成について述べ，これらの命令キャッシュ機構の問題点を明らかにする．

本論文では，個々の演算操作命令を Op（Operation），1 つ以上の並列実行可能な Op によって構成される命

令を MultOp（Multiple Operations）と記述する．また，MultOp はメモリや 2 次キャッシュ中で nop を含まないと仮定する[10]．メモリや 2 次キャッシュ中の各 MultOp の境界は，MultOp に含まれる Op に *separation bit* を付加することによって区別され，MultOp 中の最後の Op の *separation bit* が 1 になる．さらに，各 Op 中には，使用する機能ユニットを表すビット（*futype*）が付加されているとする．各命令キャッシュ機構には，以下にあげる仮定を適用する．

- 命令キャッシュはオンチップキャッシュであり，LRU 制御のセットアソシアティブ構成である．
- 2 次キャッシュのヒット率は 100%である．
- 命令キャッシュでミスが発生した場合，並列動作可能な機能ユニットと同数の Op が 2 次キャッシュから読み出される．
- 各 Op を適切な機能ユニットに分配するための処理はパイプライン化される．

2.2 NOP キャッシュ

NOP キャッシュは，MultOp の最大長と同じ長さのキャッシュラインに，機能ユニットに 1 対 1 に対応したフィールドを設定し，適切なフィールドに MultOp 中の Op を格納する．MultOp が使用しない機能ユニットに対応するフィールドには，nop が格納される．したがって，NOP キャッシュでは，nop を含んだ固定長の VLIW 命令が各キャッシュラインに格納される．**図 1** に NOP キャッシュの構成を示す．同図において，INT は整数演算ユニット，MEM はロード/ストアユニット，BR は分岐ユニット，FP は浮動小数点演算ユニットを示す．また，アルファベットは Op を表し，同じアルファベットで表される Op は同じ MultOp に含まれる．Expander は，*futype* に応じて，Op を適切なフィールドに分配するクロスバスイッチである．図 1 から分かるように，MultOp 中に nop が多く存在する場合の NOP キャッシュの使用効率は低い．そのため，低いミス率を達成するためには，キャッシュサイズを大きくする必要がある．

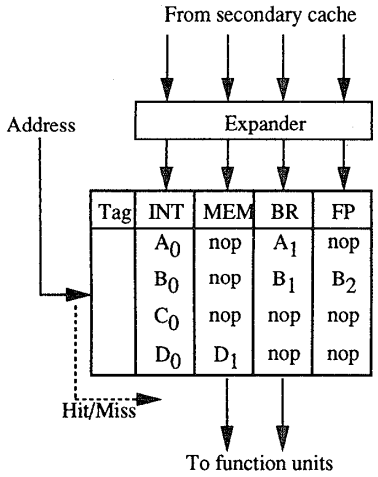


図 1 NOP キャッシュの構成
Fig. 1 NOP cache configuration.

NOP キャッシュのミス/ヒットのアルゴリズムを以下に示す．

- ミス時
  (1) LRU 制御によって置き換えするラインを決定する．
  (2) 2 次キャッシュから読み出された Op の *separation bit* を見て，1 つの MultOp を取り出す．
  (3) Expander は，*futype* に応じて各 Op をキャッシュラインの適切なフィールドに格納する．同時に，MultOp が使用しないフィールドに nop を格納する．

- ヒット時
  (1) ヒットしたラインに格納されている Op や nop をフェッチする．
  (2) フェッチされた Op や nop を各フィールドに対応する機能ユニットに発行する．

NOP キャッシュでは，2 次キャッシュと命令キャッシュの間に Expander が設置されるため，キャッシュミスの際のハードウェアは複雑になり，キャッシュのミスペナルティが増加する．それに対して，ヒットの際の命令発行機構は単純になり，T_{expand} は 0 になる．NOP キャッシュでは，1 ラインを 1 VLIW 命令としているため，ヒットの際に 1 ラインをすべてフェッチする必要がある．そのため，キャッシュバンド幅がラインサイズよりも小さい場合，T_{fetch} が 2 以上になる．

2.3 COMPRESS キャッシュ

COMPRESS キャッシュは，2 次キャッシュ中で nop を含まない VLIW 命令を，そのまま命令キャッシュに格納する．**図 2** に COMPRESS キャッシュの構成を示す．同図に示されるように，COMPRESS キャッシュ中には nop が格納されないため，キャッシュの使用効率は NOP キャッシュよりも高く，ミス率は低くなると考えられる．

COMPRESS キャッシュは，ミスの際に 2 次キャッシュから読み出された Op をそのままキャッシュに格納するため，キャッシュラインの置き換えのアルゴリズムは，通常のキャッシュと同様である．以下にヒットの際の COMPRESS キャッシュのアルゴリズムを示す．

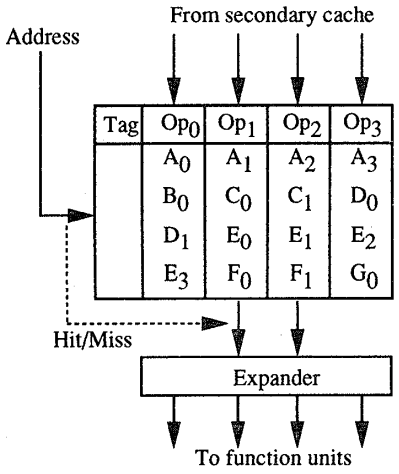


図 2 COMPRESS キャッシュの構成
Fig. 2 COMPRESS cache configuration.

- ヒット時
  (1) キャッシュバンド幅の分の Op をフェッチする.
  (2) フェッチされた Op の *separation bit* が 0 ならば (1) から繰り返す. 1 ならば (3) を行う.
  (3) Expander は, *futype* に応じてフェッチされた各 Op を適切な機能ユニットに発行する.

COMPRESS キャッシュでフェッチされる Op の数は, MultOp ごとに異なる. また, nop が格納されていないため, 1 つの MultOp が複数のラインにわたって格納される可能性がある. したがって, COMPRESS キャッシュでは, T_{fetch} が, MultOp に含まれる Op の数と MultOp が格納されている位置によって異なる.

ここで, 図 2 に示した COMPRESS キャッシュを例にとって, ヒットの際の T_{fetch} を考える. 図 2 では, キャッシュバンド幅を 2 Ops/cycle と仮定する. 最初に, MultOp A をフェッチする場合を考える. MultOp A は 4 つの Op で構成されるため, T_{fetch} は 2 である. MultOp C は 2 つの Op で構成されるが, T_{fetch} は 2 となる. この理由は, 通常のキャッシュでは, データや命令をラインの先頭から, バンド幅の増分でフェッチを行うためである. したがって, 図 2 中の Op C_0 は Op B_0 と一緒にフェッチされる. ただし, MultOp C のフェッチの際には, Op B_0 は使用しない. また, Op C_1 は Op D_0 と一緒にフェッチされる. ゆえに, MultOp C のフェッチには, 2 サイクルを要する. MultOp D をフェッチする場合には, Op D_0 と Op D_1 が異なるラインに格納されているため, フェッチに 2 サイクルを要する. MultOp E の T_{fetch} は, E_0 のフェッチで 1 サイクル, Op E_1, E_2 のフェッチで 1 サイクル, Op E_3 のフェッチで 1 サイクルであり, 計 3 サイクルである. したがって, COMPERSS キャッシュの T_{fetch} は, キャッシュ中の格納場所, および 1 つの MultOp に含まれる Op 数に応じて 1〜3 サイクルとなる.

COMPRESS キャッシュでは, NOP キャッシュとは異なり, キャッシュと機能ユニットの間に Expander を設置する必要がある. この理由は, COMPRESS キャッシュには, 機能ユニットに応じたフィールドを設定することができないためである. したがって, 命令発行機構は NOP キャッシュよりも複雑になり, T_{expand} が 1 以上になる. ただし, T_{expand} は, パイプライン化可能であるため, 分岐予測が誤りであった際にのみペナルティとなる. よって, T_{expand} による実行サイクル数の増加は少ないと考えられる.

3. MULHI (MULtiple HIt) キャッシュ

本論文では, VLIW プロセッサのための命令キャッシュ機構として, 既存のキャッシュのセットアソシアティブ構成を利用することによって, キャッシュ中の nop の割合を減少させ, 複数の Op を並列に読み出すことが可能な MULHI キャッシュを提案する. MULHI キャッシュは, MultOp に含まれる Op をセットアソシアティブキャッシュの同一セット上の別のウェイに格納する. 同一の MultOp に含まれる Op は同じタグを持ち, それらの Op を格納するウェイはキャッシュアクセス時に同時にヒットする (Multiple Hit). また, あるセットに格納されている MultOp に含まれる Op の数がウェイ数よりも小さい場合, そのセットの残りのウェイに他の MultOp を格納することが可能である.

図 3 に, 4 ウェイセットアソシアティブ方式の MULHI キャッシュの構成を示す. 図 3 のセット 0 には, 3 つの MultOp A, E, H が格納されている. このように, MULHI キャッシュでは, MultOp に含まれる Op の数に応じて, 複数の MultOp を柔軟に同一のセットに格納することが可能である. したがって, キャッシュの使用効率が向上し, ミス率が低下すると考えられる.

しかし, 図 3 では, キャッシュラインサイズを 1 つの Op の長さとしているため, NOP キャッシュに比べてタグサイズが大きくなる. このタグサイズの増加は, キャッシュラインサイズを大きくすることで緩和することが可能である. しかしながら, キャッシュのラインサイズを大きくした場合, MultOp に含まれる複数の Op が 1 キャッシュラインに格納される. したがって, もし MultOp の長さが, キャッシュラインサイズの整数倍でない場合, キャッシュには Op ととも

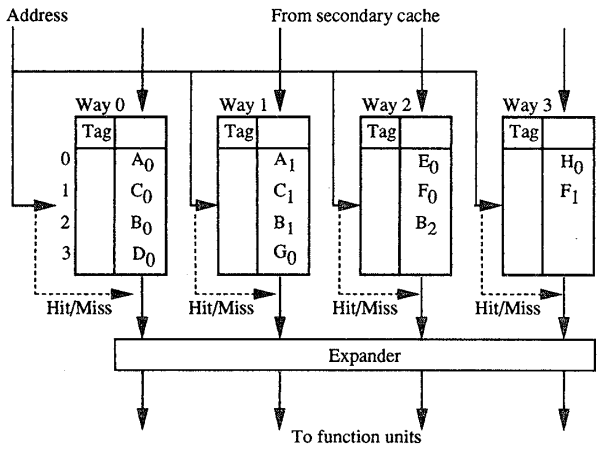


図 3 MULHI キャッシュの構成
Fig. 3 MULHI cache configuration.

に nop が格納される．たとえば，キャッシュラインサイズを 2 つの Op の長さとした場合，1 つの Op しか含まない MultOp を格納すると nop も同時に格納される．また，MultOp が 3 つの Op を含んでいる場合は，あるラインには 2 つの Op が格納されるが，もう一方のラインには 1 つの Op と nop が格納される．

以下に MULHI キャッシュのミス/ヒットのアルゴリズムを示す．

- ミス時
  - (1) 選択されたセット中の invalid なウェイと，LRU 制御により最も過去に参照されたウェイを探す．MultOp 中の Op を格納するウェイの優先順位は，invalid なウェイ，最も過去に参照されたウェイ，その他のウェイである．
  - (2) 2 次キャッシュから読み出された Op から，*separator bit* によって，1 つの MultOp を取り出す．
  - (3) 選択されたセットに MultOp を格納する．(1) で探したウェイだけで MultOp を格納できない場合は，セット全体を置き換え対象にする
- ヒット時
  - (1) ヒットしたすべてのウェイから Op をフェッチする．
  - (2) Expander は，*futype* に応じてフェッチされた各 Op を適切な機能ユニットに発行する．

ミスの際の (1) で行うウェイの選択は，2 次キャッシュからの Op の読み出しと同時に行うことができるため，キャッシュ置換えの時間に影響しない．(3) の動作時間は，(1) によって格納すべきセットとウェイが決定されているため，キャッシュへの書き込みに要する時間とほぼ等しい．したがって，MULHI キャッシュのミスペナルティは，NOP キャッシュよりも小さいと考えられる．

MULHI キャッシュでは，ヒットの際，1 つの MultOp のフェッチに複数のメモリリードポートを使用することが可能である．この理由は，キャッシュの各ウェイが，同時に読み出すことができるように構成されているためである．したがって，異なるウェイに同時にヒットする MULHI キャッシュでは，各ウェイから同時に命令をフェッチすることが可能である．ゆえに，MULHI キャッシュでは，各ウェイからの出力ビット数が 1 ラインに格納されている Op の長さと等しければ，T_{fetch} が 1 になる．また，1 MultOp を 1 サイクルで読み出すことが可能なように通常のセットアソシアティブキャッシュを NOP キャッシュとして設計した場合，Op の長さを X ビット，MultOp に含まれる最大の Op の数を N とすると，各ウェイすべてに X × N ビットのリードポートを用意する必要がある．それに対して，MULHI キャッシュでは，各ラインに格納されている Op の数を M とすると，各ウェイすべてに X × M ビットのリードポートを用意するだけでよい．通常，MULHI キャッシュを有効に使用するためには，M < N とする必要があるため，MULHI キャッシュを実現するために必要な配線量は，他の命令キャッシュよりも少なくなる．

現在，マイクロプロセッサのオンチップキャッシュのサイズが増加するにつれて，ウェイ数も増加している．この理由は，現在のマイクロプロセッサのオンチップキャッシュが，物理アドレスでアクセスできる必要があるためである．オンチップキャッシュにおいて，仮想アドレスの物理アドレス部分のみによって，キャッシュエントリの選択が可能であれば，キャッシュエントリの選択と仮想アドレスから物理アドレスへの変換を同時に実行できるため，キャッシュアクセス速度が向上する．しかし，ウェイ数が小さいキャッシュのサイズを増加させると，アドレス中のキャッシュエントリの選択に使用されるビット数が増加し，ページサイズを超える可能性がある．この場合，仮想アドレスから物理アドレスへの変換とキャッシュエントリの選択を逐次的に実行しなければならないため，キャッシュアクセス速度が低下する．したがって，キャッシュのアクセス速度を向上させるために，現在キャッシュサイズの増加にともなってウェイ数を増加させ，キャッシュインデクシング動作を物理アドレス部分で行えるようにしている．たとえば，PentiumPro[4] の命令キャッシュでは，16 KB のサイズで 4 ウェイのセットアソシ

表 1 各命令キャッシュ機構の特徴
Table 1 Charactristics of each instruction cache mechanism.

Cache mechanism	Cache utilization	T_{fetch}	T_{expand}
NOP Cache	low	medium	unnecessary
COMPRESS Cache	high	long	necessary
MULHI Cache	high	short	necessary

アティブ構成をとっている．また，PowerPC750[11] では，32 KB のサイズで 8 ウェイのセットアソシアティブ構成をとっている．したがって，MULHI キャッシュは，既存の大きなウェイ数を備えるキャッシュによって容易に実現される考えられ，なおかつ大きなウェイを持つ VLIW 用命令キャッシュ機構として，配線量においても有利であると考えられる．

MULHI キャッシュでは，COMPRESS キャッシュと同様，各ウェイに格納される Op の種類は任意であるため，キャッシュと機能ユニットの間に，Op を適切な機能ユニットに分配する Expander が必要になる．したがって，命令発行機構は複雑になり，T_{expand} が 1 以上になる．

本章で述べた各命令キャッシュ機構の特徴を**表 1** に示す．表 1 中で，Cache utilization は命令キャッシュの使用効率である．

4. 性能評価

本章では，最初に，SPEC95 ベンチマーク中のいくつかのプログラムを用いたシミュレーションによって得られた NOP キャッシュ，COMPRESS キャッシュ，MULHI キャッシュのミス率，1 サイクルで実行された Op の数（OPC: Operations Per Cycle）を用いて，各命令キャッシュ機構を評価する．次に，各命令キャッシュ機構をハードウェア量の面から評価する．

4.1 シミュレーションによる性能評価

4.1.1 実験方法

実験は，MIPS R4000 の命令セット[12] を基本としたソフトウェアシミュレータ上で行った．シミュレータは，整数演算命令，ロード/ストア命令，分岐命令を実行可能な機能ユニットを 2 つと，それらの命令に加えて浮動小数点演算命令を実行可能な機能ユニットを 2 つ備えている．したがって，並列実行可能な最大の Op 数は 4 である．MIPS R4000 における命令の長さは 4 バイトであるため，Op の長さは 4 バイト，MultOp の最大長は 16 バイトである．

本論文で想定している VLIW アーキテクチャでは，MultOp の境界や Op が使用する機能ユニットを示すビットなどが必要である．また，レジスタ数の増加やプレディケートのためにより大きな命令長が要求される可能性がある．そこで，本論文では，VLIW プロセッサのための制御ビットを付加した TINKER VLIW testbed の命令フォーマット[9),10] の演算操作部分に，MIPS R4000 の命令を用い，Op の長さを 8 バイト，MultOp の最大長を 32 バイトとした場合の実験も行った．

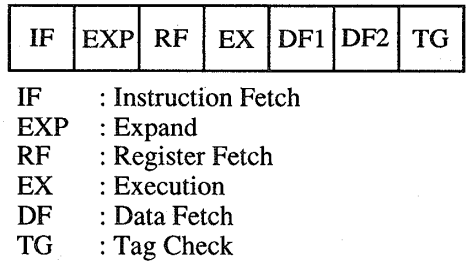


図 4 Op を機能ユニットに分配するステージを含んだパイプライン構成
Fig. 4 Pipeline configuration including the expand stage.

NOP キャッシュと COMPRESS キャッシュのラインサイズは，1 MultOp の最大長と等しい大きさとした．MULHI キャッシュでは，ラインサイズを 1 つの Op の長さにした場合と，2 つの Op の長さにした場合の実験を行った．

各命令キャッシュ機構は，すべて 8 ウェイのセットアソシアティブ構成とした．シミュレーションでは，2 次キャッシュから 1 つの Op を読み出すのに 3 サイクル，その他の 3 つの Op をバースト転送するのに 3 サイクルがそれぞれ必要であると仮定した．NOP キャッシュでは，キャッシュミスの際に Op や nop を適切なフィールドに分配する必要があるため，キャッシュミスペナルティを他の命令キャッシュ機構よりも 1 サイクル大きく，7 サイクルとした．

命令キャッシュのバンド幅は 16 バイト/サイクルとした．したがって，Op の長さが 4 バイトの場合，すべての命令キャッシュ機構で T_{fetch} が 1 になる．ただし，COMPRESS キャッシュでは，MultOp が異なるラインに格納されている場合，T_{fetch} が 1 増加する．T_{expand} はパイプライン化され，分岐予測ミスの際にのみペナルティとなる．T_{expand} を含んだパイプラインの構成を図 4 に示す．実験で使用したシミュレータでは，2 ビットで予測を行う BTB（Branch Target

表 2 各命令キャッシュ機構における T_{fetch}（Op の長さが 8 バイト）
Table 2 Parameters of each instruction cache mechanism.

Cache mechanism	T_{fetch}				
	# of Ops	1	2	3	4
NOP Cache		2	2	2	2
COMPRESS Cache		1	1~2	1~2	2~3
MULHI Cache		1	1	1	1

表 3 使用したベンチマークプログラムの特徴
Table 3 Charactristics of benchmark programs.

Benchmark	Code size (KB)		Static instruction length (ave.)	Dynamic instruction length (ave.)
	4-byte Op	8-byte Op		
compress	24	48	1.26	1.36
li	207	414	1.36	1.20
mgrid	59	118	1.67	2.31
turb3d	253	507	1.22	1.34

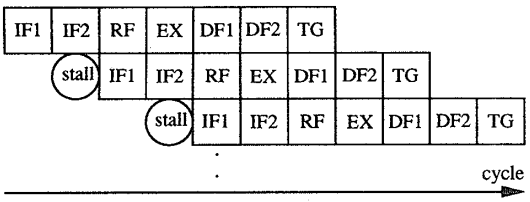


図 5 NOP キャッシュを用いた際のパイプライン構成
Fig. 5 Pipeline configuration for the NOP cache.

Buffer）を用い，分岐ペナルティを 1 サイクルとした．

Op の長さが 8 バイトの場合の T_{fetch} を**表 2** に示す．表 2 中の T_{fetch} は，MultOp に含まれる Op の数の違いに応じて示してある．NOP キャッシュでは，T_{fetch} がつねに 2 であるため，使用されるパイプラインの構成は**図 5** のようになる．ただし，図 5 中の IF1 と IF2 は同時に実行できないため，つねに 1 サイクルのストールが生じる．COMPRESS キャッシュのパイプライン構成は図 4 と同じであるが，T_{fetch} の値に応じて，ストールが生じる．

実験に使用したベンチマークは，SPECint95 の *compress*, *li* と SPECfp95 の *mgrid*, *turb3d* である．使用したベンチマークの特徴を**表 3** に示す．これらのベンチマークは，*gcc* のフロントエンドによって逐次処理用のアセンブリコードに変換される．生成されたアセンブリコードは，本研究で作成したスケジューラによって，基本ブロック内でリストスケジューリングされる．スケジューリングされたコードは，カラーリングアルゴリズム[13] に基づき本研究で作成したレジスタアロケータによって，レジスタ割当てが行われ，シミュレータへの入力となる．表 3 中のコードサイズは，Op の長さが 4 バイトと 8 バイトの場合の nop を含んだプログラムのサイズを示している．静的平均命令長は，コンパイラによってスケジューリングされた後の 1 つの MultOp に含まれる平均の Op の数を示す．動的平均命令長は，実際に実行した結果，実行された 1 つの MultOp に含まれる Op の平均の数を示す．

4.1.2 実験結果

図 6～**図 13** に，16 KB，32 KB のキャッシュサイズにおけるミス率と OPC の関係を示す．図 6～図 9 は Op の長さが 4 バイトの場合の結果であり，図 10～図 13 は Op の長さが 8 バイトの場合の結果である．以下では，Op の長さが 4 バイトの場合と 8 バイトの場合の結果について考察する．

・4 バイト Op

図 6～図 9 から，16 KB のキャッシュサイズにおける *li* と *turb3d* を除いて，すべての命令キャッシュのミス率が非常に低いことが分かる．この理由は，キャッシュサイズに対してプログラムのコードサイズが小さいためである．したがって，16 KB のキャッシュサイズにおける *li* と *turb3d* 以外の場合，キャッシュの使用効率によって生じるミス率の差は，性能に大きく影響していない．これは，実行サイクル中でミスペナルティの占める割合が小さいことを意味している．その結果，COMPRESS キャッシュは，ミス率が最も低いにもかかわらず，T_{issue} が大きいために他の命令キャッシュよりも OPC が小さくなっている．それに対して，MULHI キャッシュは，ミス率が NOP キャッシュよりも多少低いため，NOP キャッシュと同等か，それ以上の OPC を示している．このことから，キャッシュサイズを増加させる，もしくは小さいコードサイズのアプリケーションプログラムを使用するといった，NOP キャッシュが有利なミス率の低い状況になって

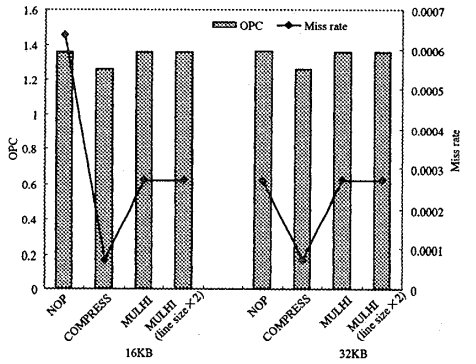


図 6 *compress* における OPC とミス率の関係（4 バイト Op）
Fig. 6 OPC and miss rate of *compress* (4-byte Op).

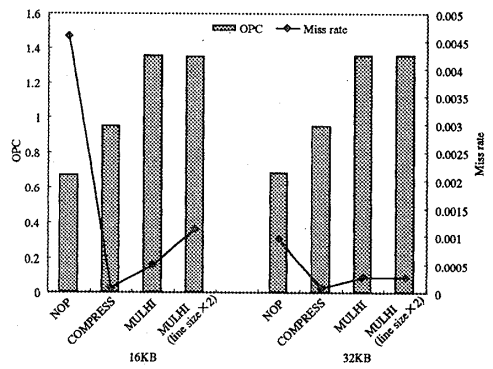


図 10 *compress* における OPC とミス率の関係（8 バイト Op）
Fig. 10 OPC and miss rate of *compress* (8-byte Op).

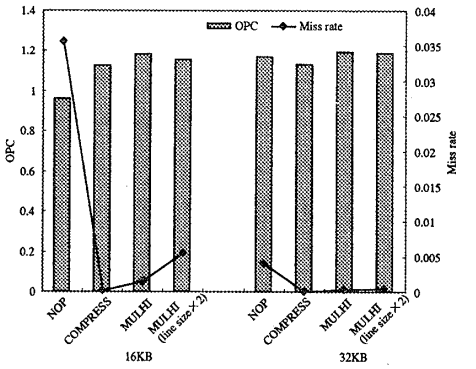


図 7 *li* における OPC とミス率の関係（4 バイト Op）
Fig. 7 OPC and miss rate of *li* (4-byte Op).

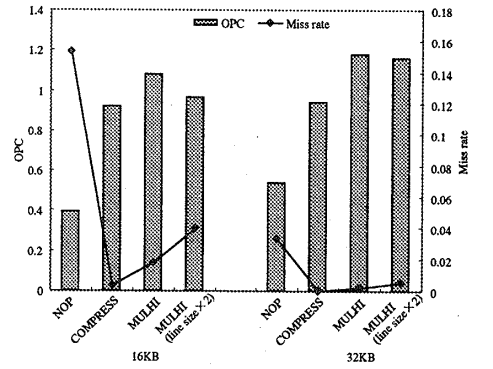


図 11 *li* における OPC とミス率の関係（8 バイト Op）
Fig. 11 OPC and miss rate of *li* (8-byte Op).

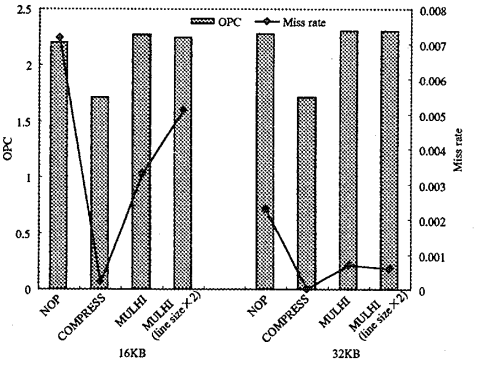


図 8 *mgrid* における OPC とミス率の関係（4 バイト Op）
Fig. 8 OPC and miss rate of *mgrid* (4-byte Op).

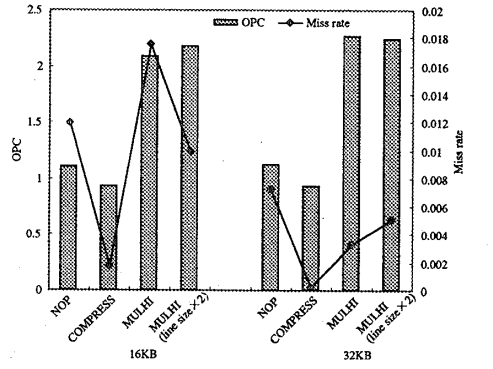


図 12 *mgrid* における OPC とミス率の関係（8 バイト Op）
Fig. 12 OPC and miss rate of *mgrid* (8-byte Op).

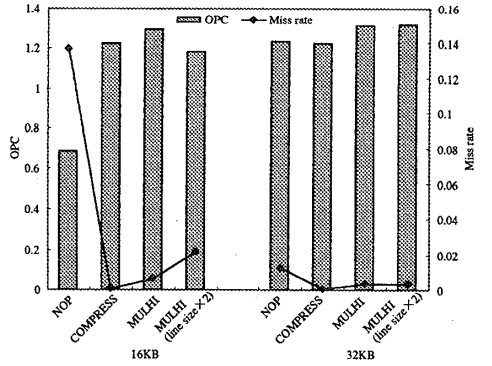


図 9 *turb3d* における OPC とミス率の関係（4 バイト Op）
Fig. 9 OPC and miss rate of *turb3d* (4-byte Op).

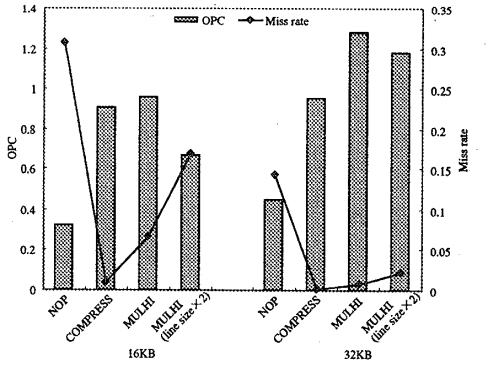


図 13 *turb3d* における OPC とミス率の関係（8 バイト Op）
Fig. 13 OPC and miss rate of *turb3d* (8-byte Op).

表4 各命令キャッシュのハードウェア量（8-byte Op, 32 KB）
Table 4 Amount of hardware of each instruction cache mechanism (32 KB).

Cache mechanism	Tag + Data (bytes)	Increase (%)
NOP cache	34944	–
COMPRESS cache	34944	0.0
MULHI cache	41472	18.7
MULHI cache (line size × 2)	37120	6.2

も，MULHIキャッシュは，NOPキャッシュ以上の性能を達成できると結論付けることができる．

16 KBのキャッシュサイズにおける*li*と*turb3d*では，コードサイズに対して，キャッシュサイズが小さいため，NOPキャッシュのミス率が高い．その結果，NOPキャッシュのOPCは最も小さくなっている．COMPRESSキャッシュとMULHIキャッシュを比較すると，ミス率はCOMPRESSキャッシュの方が低いが，OPCはMULHIキャッシュの方が大きい．この理由は，MULHIキャッシュのT_{fetch}がCOMPRESSキャッシュよりも小さいためである．実験の結果，*li*, *turb3d*におけるCOMPRESSキャッシュの平均のT_{fetch}は，それぞれ1.05, 1.09であった．それに対して，MULHIキャッシュのT_{fetch}はつねに1である．以上のことから，キャッシュサイズが小さい場合でも，MULHIキャッシュはCOMPRESSキャッシュよりも有効であるということができる．

・8バイトOp

図10～図13から，すべての場合において，COMPRESSキャッシュのミス率が最も低いことが分かる．しかし，COMPRESSキャッシュのOPCは，それほど大きくない．特に，*mgrid*では，NOPキャッシュよりも小さいOPCを示している．この理由は，キャッシュのバンド幅が不十分である場合，COMPRESSキャッシュのT_{fetch}が大きくなるためである．それに対して，MULHIキャッシュは，すべての場合において最も良いOPCを示している．この理由は，MULHIキャッシュのT_{fetch}が小さいためである．その結果，MULHIキャッシュは，NOPキャッシュに対して最大1.68倍の性能向上を示した．

16 KBのキャッシュサイズにおける*mgrid*を除いて，1ラインに1つのOpを格納するMULHIキャッシュのOPCが，2つのOpを格納するMULHIキャッシュよりも大きい．この理由は，2つのOpを格納するMULHIキャッシュには，nopが格納される場合があり，ミス率が高くなるためである．しかし，16 KBのキャッシュサイズにおける*mgrid*では，1ラインに2つのOpを格納するMULHIキャッシュのOPCが，1つのOpを格納するMULHIキャッシュよりも大きい．この理由は，*mgrid*のように動的平均命令長が長い場合，1つのセットに格納可能なMultOpの数が少ないと，1つのセット内で競合が頻繁に起こるためである．1ラインに2つのOpを格納するMULHIキャッシュは，1つのOpを格納するMULHIキャッシュと比較して，2倍のMultOpを同一セットに格納できるため，セット内での競合が減少し，性能が向上している．その結果，1ラインに2つのOpを格納するMULHIのミス率が低くなり，OPCが大きくなっている．

16 KBのキャッシュサイズにおける*turb3d*では，1ラインに2つのOpを格納するMULHIキャッシュのOPCが，COMPRESSキャッシュよりも小さくなっている．この理由は，*turb3d*の動的平均命令長が短いため，2つのOpを格納するMULHIキャッシュに多くのnopが格納されるためである．したがって，他のベンチマークに比べて，1ラインに2つのOpを格納するMULHIキャッシュとCOMPRESSキャッシュのミス率の差が大きくなり，1ラインに2つのOpを格納するMULHIキャッシュのOPCが小さくなる．以上から，MULHIキャッシュでは，1ラインに格納するOpの数を増加させると，プログラムの動的平均命令長によって性能が大きく左右されるようになるということが分かる．

4.2 ハードウェア量評価

本節では，4.1節で示した性能に加えて，ハードウェア量を考慮に入れた各命令キャッシュ機構の評価を行う．**表4**に，Opの長さが8バイト，キャッシュサイズが32 KBの場合の各命令キャッシュ機構のタグとデータの総容量と，NOPキャッシュの全ハードウェア量に対する各命令キャッシュ機構のハードウェア量の増加割合を示す．

COMPRESSキャッシュは，NOPキャッシュと同じハードウェア量で実現可能である．しかし，4.1節で述べたように，NOPキャッシュのミス率が低い場合，COMPRESSキャッシュは，大きなT_{issue}のためにNOPキャッシュよりも性能が低い．したがって，COMPRESSキャッシュは，NOPキャッシュで低いミス率を得ることが困難な場合有効である．たとえば，使用するアプリケーションプログラムのコードサイズ

が大きい場合や，オンチップの命令キャッシュのサイズを大きくできない場合，COMPRESS キャッシュはミス率を減少させ，性能を向上させることが可能である．

表 4 から分かるように，ラインサイズが Op のサイズと一致する MULHI キャッシュは，性能の向上は大きいが，ハードウェア量の増加も 18.7%と大きい．しかし，図 6〜図 13 に示されるように，MULHI キャッシュの OPC は，2 倍のサイズの COMPRESS キャッシュより大きい．したがって，COMPRESS キャッシュを用いるよりも，1/2 のサイズの MULHI キャッシュを用いた方が有効であるといえる．また，図 10〜図 13 から分かるように，プログラムのコードサイズが大きいと，NOP キャッシュのミス率が高くなるため，MULHI キャッシュの性能が 2 倍のサイズの NOP キャッシュよりも良い．したがって，現在使用されている多くのアプリケーションプログラムのコードサイズが大きいことを考慮すると，NOP キャッシュのサイズを 2 倍にするよりも，タグサイズを増加させて MULHI キャッシュを採用する方が効果的であるといえる．

ラインサイズが 2 つの Op のサイズと一致する MULHI キャッシュは，ラインサイズが Op のサイズと一致する MULHI キャッシュに比べて，ハードウェア量の増加は小さいが，性能向上も小さい．しかしながら，同じキャッシュサイズで他の命令キャッシュ機構と比較すると，ラインサイズを 2 倍にしても，ほとんどの場合，MULHI キャッシュの性能が良い．したがって，命令キャッシュのハードウェア量を小さくしたい場合は，ラインサイズを 2 倍にした MULHI キャッシュが効果的であると考えられる．

MULHI キャッシュは，各 VLIW 命令を複数の Op に分割してキャッシュに格納する点で，文献 9) に示されている Silo キャッシュと類似する．Silo キャッシュでは，複数のキャッシュメモリが使用され，各キャッシュメモリに特定の機能ユニットを使用する Op が格納される．その結果，T_{expand} が 0 になり，MULHI キャッシュよりも T_{issue} が小さい．しかし，Silo キャッシュでは，各キャッシュに格納する Op の種類を限定しているため，Silo キャッシュのミス率は MULHI キャッシュよりも高いと予想される．また，Silo キャッシュは複数のキャッシュメモリを使用するため，セットアソシアティブキャッシュのキャッシュウェイを効果的に利用する MULHI キャッシュよりもハードウェア量が大きいと考えられる．本研究では，これらの点に基づき，今後 Silo キャッシュと MULHI キャッシュを比較する予定である．

5. まとめ

本論文では，VLIW プロセッサのための新たな命令キャッシュ機構である MULHI キャッシュを提案し，評価を行った．その結果，MULHI キャッシュのミス率は，最もミス率が低い COMPRESS キャッシュに近い値であった．その効果に加えて，MULHI キャッシュは，機能ユニットへの発行サイクルが小さいため，NOP キャッシュと比べて最大 1.68 倍の性能向上を示した．また，MULHI キャッシュは，2 倍のサイズの COMPRESS キャッシュよりも高性能であり，コストパフォーマンスが良いことも分かった．

今後の課題として，本論文で提案した命令キャッシュ機構を適切なデバイステクノロジデータを用いて設計し，キャッシュミスペナルティや命令発行サイクルなどを正確に測定することがあげられる．また，文献 9) に示されている Silo キャッシュなどの他の命令キャッシュ機構との詳細な比較も今後の課題である．

参考文献

1) Fisher, J.A.: Very long instruction word architectures and the ELI-512, *Proc. 10th Ann. Symposium on Computer Architectures*, pp.140–150 (1983).
2) マイクジョンソン：スーパースカラプロセッサ，日経 BP 出版センター (1994).
3) SEMICONDUCTOR INDUSTRY ASSOCIATION: *The National Technology Roadmap for Semiconductors* (1998).
4) Intel: *PentiumPro processor developer's manual* (1996).
5) Texas Instruments Incorporated: *TMS320C62x /67x CPU and instruction set reference guide* (1998).
6) Fisher, J.A.: Trace scheduling: A technique for global microcode compaction, *IEEE Trans. Comput*, Vol.C-30, pp.478–490 (1981).
7) Mahlke, S.A., Chen, W.Y., Bringmann, R.A., Hank, R.E., Hwu, W. M.W., Rau, B.R. and Schlansker, M.S.: Sentinel Scheduling: A Model for Compiler – Controlled Speculative Execution, *ACM Trans. Computer Syst.*, Vol.11, No.4, pp.376–408 (1993).
8) Yung, R.: Design Decisions Influencing the UltraSPARC's Instruction Fetch Architecture, *Proc. 29th Ann. IEEE/ACM International Symposium on Microarchitecture*, pp.178–190 (1996).
9) Conte, T.M., Banerjia, S., Larin, S.Y. and Menezes, K.N.: Instruction Fetch Mechanisms

for VLIW Architectures with Compressed Encodings, *Proc. 29th Ann. IEEE/ACM International Symposium on Microarchitecture*, pp.201–213 (1996).
10) Conte, T.M. and Sathaye, S.W.: Dynamic rescheduling: A technique for object code compatibility in VLIW architectures, *Proc. 28th Ann. International Symposium on Microarchitecture*, pp.208–218 (1995).
11) Motorola: *MPC750 RISC Microprocessor Hardware Specifications* (1997).
12) MIPS: *R4000/R4400 Microprocessor User's Manual* (1994).
13) Chow, F. and Hennessy, J.: Register Allocation by Priority – based Coloring, *The ACM SIGPLAN '84 Symposium on Compiler Construction SIGPLAN Notices*, Vol.19, No.6 (1984).

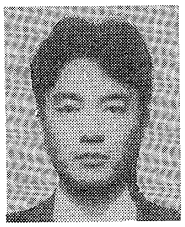
(平成 10 年 8 月 31 日受付)
(平成 11 年 3 月 5 日採録)

仲池 卓也（学生会員）

昭和 47 年生．平成 7 年東北大学工学部機械知能工学科卒業．平成 9 年同大学大学院情報科学研究科情報基礎科学専攻博士課程前期修了．現在，同大学大学院情報科学研究科情報基礎科学専攻博士課程後期に在学．計算機アーキテクチャの研究に従事．

阿部 孝之

昭和 49 年生．平成 9 年東北大学工学部機械知能工学科卒業．同年，同大学大学院情報科学研究科情報基礎科学専攻博士課程前期入学，現在に至る．計算機アーキテクチャの研究に従事．



大庭 信之（正会員）

昭和 33 年生．昭和 56 年東北大学工学部通信工学科卒業．昭和 61 年同大学大学院工学研究科電気および通信工学専攻博士課程修了．同年，日本アイ・ビー・エム（株）東京基礎研究所入社，現在に至る．工学博士．計算機アーキテクチャの研究に従事．

小林 広明（正会員）

昭和 36 年生．昭和 58 年東北大学工学部通信工学科卒業．昭和 63 年同大学大学院工学研究科情報工学専攻博士課程修了．工学博士．現在，東北大学大学院情報科学研究科助教授．平成 7 年スタンフォード大学客員助教授併任．並列計算機アーキテクチャ，コンピュータグラフィックスの研究に従事．電子情報通信学会，ACM，IEEE 各会員．

中村 維男

昭和 19 年生．昭和 47 年東北大学大学院工学研究科博士課程修了．工学博士．同年同大学工学部助手．昭和 53 年同大学助教授．昭和 63 年同大学大学院情報科学研究科教授，現在に至る．平成 6 年よりスタンフォード大学客員教授併任．研究分野は，計算機アーキテクチャ．日本機械学会，電子情報通信学会，IEEE 各会員．