計算機アーキテクチャ 119－9
（1996．8．27）

ロードアドレス予測による命令並列度の向上

西本晴子† 勝野 昭† 木村康則†

総実行サイクル数を低減し命令レベル並列度を向上させるための，ロード命令のアドレス予測手法を提案する．データキャッシュアクセスレイテンシを低減するロードアドレス予測について，トレースベースのシミュレータで評価を行なった．その結果，in-order実行のプロセッサにおいて，総実行サイクル数は最大25% 減少し，Instruction Per Cycle(IPC) は33% 向上した．また，out-of-order実行のプロセッサにおいて，総実行サイクル数が最大5% 減少し，IPCは5% 向上した．

Improving Instruction Level Parallelism using Load Address Prediction Techniques

HARUKO NISHIMOTO,† AKIRA KATSUNO† and YASUNORI KIMURA†

In this paper, we propose a load address prediction technique to reduce total execution cycles, and to improve Instruction Level Parallelism(ILP). We evaluated an effect of load prediction techniques to reduce data cache latency on a trace-based simulator. Our evaluations showed that the total execution cycles were reduced by 25% and Instruction Per Cycle(IPC) was improved by 33% on an in-order execution processor model, and the total execution cycles were reduced by 5% and IPC was improved by 5% on an out-of-order execution processor model.

1. はじめに

一般にプロセッサでは，メモリからのデータの読み出しや書き込みが非常に頻繁に行なわれる[1)~4)]．従って，プロセッサの高速化のためには，プログラム実行時にプロセッサにデータをいかにうまく供給するかが重要である．近年，プロセッサのクロック周波数は急激に向上する傾向にあるが，データキャッシュのアクセスレイテンシはあまり減少していない．よって，プロセッサの性能が向上するにつれて，データキャッシュレイテンシの影響が相対的に大きくなっている．

そこで，データキャッシュレイテンシを低減し，メモリアクセス命令と後続の命令の投機的実行を行なう手法の一つとして，ロードアドレス予測の手法を提案する．本稿では，3種類のロードアドレス予測アルゴリズムを示し[2)5)]，総実行サイクル数とIPCに対するロードアドレス予測の効果と，それに関する考察について述べる．

2. ロードアドレス予測メカニズム

2.1 アドレス予測と命令間の依存関係

ロードアドレス予測はプロセッサの最適化手法の一つと考えられている[5)]．それは，データアドレスをあらかじめ予測することで，そのロード命令のデータアドレスを計算する命令と，そのアドレスを使用するロード命令との依存関係をなくし，データキャッシュレイテンシを低減するものである．

まず，ロードアドレス予測を用いた場合の命令間の依存関係について説明する．例としてSPARCの命令セットを用いる．命令の最後のオペランドはデスティネーション (出力) レジスタで，その他がソース (入力) レジスタである．以下のような命令列を考える．

```
(1)  sethi  1dd,      R1
(2)  or     R1,       2de,  R1
(3)  ld     [R1+8],   R3
(4)  ld     [R4+8],   R6
(5)  sub    R5,       R3,   R5
(6)  sub    R6,       R7,   R8
```

これらの命令列のデータ依存は以下のように示される．

TM case A:

(1) → (2) → (3) → (5)
(4) → (6)

ロードアドレス予測を用い，この命令列のロードアドレスが正しく予測されたとすると，(3) のアドレス計算がロードアドレス予測用のハードウェアによって先行実行されるため，(2) と (3) の依存関係は解消される．

† (株) 富士通研究所 ネットワークコンピューティング研究部
Network Computing Laboratory, Fujitsu Laboratories LTD.

よって，ロードアドレス予測を行なった場合，この命令列間のデータ依存は以下のようになる．

case B:
(1) → (2)
(3) → (5)
(4) → (6)

case B の場合ロードアドレス予測を用いることにより，依存関係の深さは case A の半分になる．依存関係の解消された命令列は並列に実行できるので，命令並列度の向上が期待でき，総実行サイクル数を低減できる．

2.2 予測アルゴリズム

高いロードアドレス予測正解率を得るために，以下の3つのアルゴリズムを提案する．これらのアルゴリズムの目的は，規則的に定数値または変数値でのアドレスインクリメントを行なうような[7]，ストライドアクセスロード命令の実効アドレスを正確に予測することである．図1はこのようなロード命令の例である．このようなコードは，数値計算アプリケーションなどで行列データを参照する時によく見られる．

```
L1:  add R1, 64, R1
     ld   [R1], R2
          :
     b    L1
```

図1 ストライドロード命令の例
Fig. 1 An example of stride load instruction.

これらロード命令のデータアドレスを予測するために，**ロードアドレスキャッシュ** (LAC) と呼ぶ一種のハードウェアキャッシュを用意する．LAC はロード命令のプログラムカウンタ (PC)，そのロード命令のデータアドレス，前回のデータアドレスと前々回のデータアドレス差，LAC 情報更新のための状態ビットを保持する．図2に今回の実験で仮定した LAC の構成を示す．

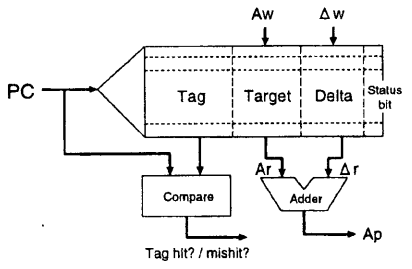

図2 ロードアドレスキャッシュ (LAC) の構成
Fig. 2 Block diagram of a Load Address Cache(LAC).

Tag はロード命令のプログラムカウンタ，**Target** はそのロード命令が前回に用いたデータアドレス，**Delta** は 前回と前々回のデータアドレスの差，また **Aw** は LAC に書き込むデータアドレス，**Ar** は LAC から読み出されたデータアドレス，**Ap** は予測アドレス，**Δw** は LAC に書き込む Delta の値，**Δr** は LAC から読み出された Delta の値をそれぞれ表す．

以下に，今回実験を行なった3つの予測アルゴリズムについて説明する．

Algorithm 0

Algorithm 0 で LAC の各エントリは，Tag，ターゲットアドレスのみで構成し，Delta と status bit は使用しない．アドレス予測の際，ロード命令の PC を LAC 検索のタグとして用い，タグがヒットすれば，ターゲットアドレスがそのまま予測アドレスとしてロード命令に渡され，そのアドレスでデータキャッシュアクセスを行なう．予測アドレスが外れた場合，ターゲットアドレスは毎回更新される．

このアルゴリズムを以下に示す．この中の，**Ac** は実際のデータアドレスを表す．

```
Ap = Ar ;     /* 予測アドレス */
if (予測が正解)
     then Aw = Ap ;
else  (予測が不正解)
     then Aw = Ac ;
```

Algorithm 0 は，毎回同じデータアドレスにアクセスするようなロード命令に対して効果があると予想される．

Algorithm 1

Algorithm 1 で LAC の各エントリは，Tag，ターゲットアドレス，Delta フィールドで構成し，status bit は使用しない．予測アドレスは，ターゲットアドレスと Delta を足して求められ，ロード命令に渡される．もし予測が正しければ，ターゲットアドレスは現在のデータアドレスに更新されるが，Delta フィールドは変更されない．予測が外れた場合は，ターゲットアドレスと Delta の両方を予測が外れる度に更新する．

このアルゴリズムを以下に示す．

```
Ap = Ar + Δr ;     /* 予測アドレス */
if (予測が正解)
     then Aw = Ap ;
else (予測が不正解)
     then Aw = Ac, Δw = Ac − Ar ;
```

Algorithm 1 は，データアドレスの差が常に一定であるようなロード命令に対して効果があると予想される．

Algorithm 2

Algorithm 2 で LAC の各エントリは，Tag，ターゲットアドレス，Delta フィールド，状態ビット，の4つで構成する．状態ビットは Delta を更新するための情報を保持する．

タグがヒットすれば，ターゲットアドレスと Delta を足して予測アドレスとする．予測が成功すると状態ビッ

トを1とし，予測が失敗すると0とする．0の時にさらに予測が失敗すると，Deltaが新しい値に更新される．すなわち，2回連続して予測が失敗した時にのみDeltaを更新するというところがAlgorithm 1と違う点である．図3に状態ビットの遷移の様子を示す．

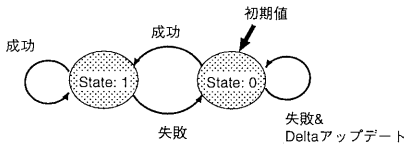

図3 状態ビットの遷移
Fig. 3 Transition of status bit.

このアルゴリズムを以下に示す．

```
Ap = Ar + Δr;     /* 予測アドレス */
if (予測が正解)
     then Aw = Ap, status_bit = 1 ;
else (予測が不正解)
     if (status_bit == 0)    /* Δ の更新 */
          then Aw = Ac, Δw = Ac − Ar ;
     else
          then Aw = Ac, status_bit = 0 ;
```

Algorithm 2は，ロードアドレスのアクセスパターンが一定値のストライドを持つが，時々突発的に変化するようなロード命令に対して，効果があると予想される．

3. 実験方法

3.1 パイプライン

図4(1)はLACのない場合のシミュレーションで用いている，通常のロード命令の実行パイプラインを表している．そのパイプラインは命令フェッチ，命令デコード，アドレス計算，メモリアクセス，書き戻し，の5ステージである．

ロードやストアのようなメモリアクセス命令が実行される時，データのデータアドレスがアドレス計算ステージで求められ，メモリアクセスステージでそのアドレスを用いてデータキャッシュにアクセスされ，最終的に書き戻しのステージでロードされたデータはレジスタファイルに書き込まれる．

一方，図4(2)は，LACを用いてアドレス予測を行なう場合の実行パイプラインを表している．最初のステージでロード命令のフェッチとLACのアクセスを行なってデータアドレスの予測を行ない，次のステージではそのアドレスに基づいてデータキャッシュにアクセスを行なう．そして次のステージで，ロードされたデータは一時レジスタに書き込みされる．よって，予測が当たれば，データを得るのに3サイクルで済むと仮定する．予測が外れた場合は，LACがなかった時と同じサイクルだけかかると仮定する．

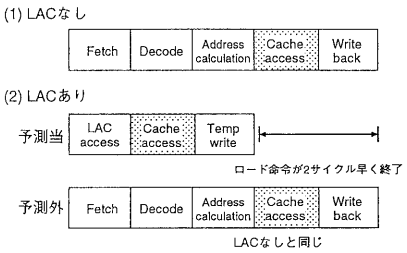

図4 パイプライン機構
Fig. 4 Pipeline mechanism.

3.2 評価用シミュレータ

LACの性能を評価するために，シミュレータ(LACsim)を開発した．LACsimは，SPARC V8アーキテクチャに準拠した命令レベル解析ツールであるSHADOWのモジュールである[6]．図5に，LACsimの構成を示す．

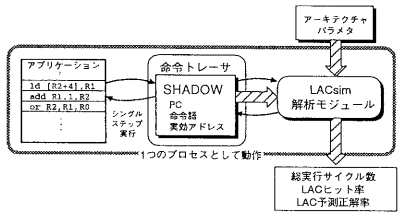

図5 評価用シミュレータ(LACsim)
Fig. 5 Evaluation simulator (LACsim).

LACsimは，SHADOWから各ロード命令のプログラムカウンタと実効アドレスの履歴を受け取る．それからLACsimモジュール内で，総実行命令数やLACのタグヒット率，LACの予測正解確率などを算出する．

図6は，LACsimで仮定しているロードユニットモデルを表している．実アドレスと予測アドレスの両方が同時にデータキャッシュにアクセスでき，実アドレスによってロードされたデータは，常にレジスタファイルに書き込まれ，予測アドレスによってロードされたデータは常に一時レジスタに書き込まれる．予測データは実データの流れには影響を与えず，予測が間違っていた場合にのみ，予測データによって先行実行された命令をキャンセルし，実アドレスによってロードされたデータを使用して再実行を行なう．

よって，これらの条件から今回の評価での仮定をまとめる．

- LACを用いることにより，アドレス計算命令とそのアドレスを使用するロード命令との依存関係がなくなる．
- ロードアドレスが正しく予測された場合，データキャッシュレンテンシが2サイクル低減できる．
- ロードアドレスが間違って予測された場合，予測ミ

スによるペナルティはない．

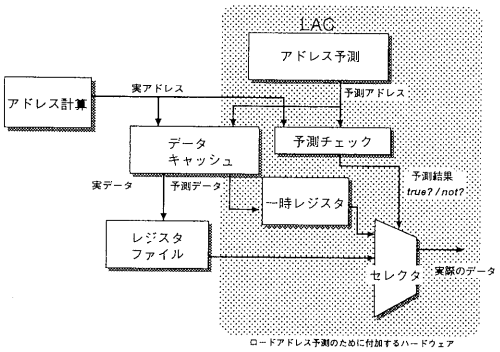

図6 ロードユニットモデル
Fig. 6 Load unit model.

3.3 LACの構成と評価プロセッサモデル

今回我々は，LACのサイズを一定にして一連の実験を行なった．512エントリで4ウェイ(全エントリ数2048エントリ)の連想方式LACである．また，データキャッシュも16KBに固定した．シミュレーションで用いたプロセッサモデルは，4命令発行でin-order実行またはout-of-order実行のモデルである．表1に今回の評価モデルを示す．

表1 評価プロセッサモデル
Table 1 Processor model on this evaluation.

実行形式	in-order / out-of-order
命令発行 / 命令フェッチ	4命令 / 4命令
分岐先バッファ	128エントリ
キャッシュ	命令: 16 KB　データ: 16 KB
キャッシュミスペナルティ	3サイクル
ストアバッファ	8エントリ
待機ステーション	LD/ST：17，others：32エントリ
機能ユニット	ALU ×2　(latency 1)
	LDST　(latency 2)
	SFT　(latency 1)
	MUL　(latency 16)
	DIV　(latency 32)
	FADD　(latency 3)
	FMUL　(latency 3)
LAC	**512エントリ 4ウェイ**
	total：2048エントリ

3.4 ベンチマークプログラムとコンパイラ

評価用プログラムとして，SPECint92から6つの整数系プログラム，SPECfp92から6つの浮動小数点系プログラムを抜粋して用いた．全てのベンチマークプログラムはGNU gcc version 2.4.5でコンパイルされ，最適化オプションはO2である．

4. 実験結果

LACsimを用い，3つのアルゴリズムの総実行サイクル数と予測正解率の比較を行なった．

表2はLACのタグヒット率(TH)と各アルゴリズムの予測正解率を示す．タグヒット率の平均は98%で，Algorithm 0 (A0)での予測正解率は約44%，Algorithm 1 (A1)では79%，Algorithm 2 (A2)では83%であった．全てのベンチマークプログラムに対して，Algorithm 2の性能が最も良かった．特にそれは浮動小数点系のプログラムに顕著に見られ，Algorithm 1と2の正解確率の違いから，LACの各々のエントリに1ビット付け加えるだけでも，予測正解率は4%程度向上することが分かった．

表2 各アルゴリズムに対するLACタグヒット率と予測正解率
Table 2 LAC correct prediction ratio for each algorithm.

program	TH	A0(%)	A1(%)	A2(%)
espresso	99.66	57.04	67.94	72.77
xlisp	99.99	46.24	54.18	63.58
eqntott	99.79	44.92	81.15	85.63
compress	99.98	63.80	79.36	80.86
sc	99.37	71.58	85.67	89.85
gcc	97.26	37.86	47.18	53.88
average(int)	99.34	53.53	69.24	74.42
tomcatv	99.99	19.55	99.35	99.67
alvinn	99.99	77.25	83.83	88.08
ear	99.99	14.11	97.74	98.84
swm256	99.99	27.76	99.45	99.72
hydro2d	99.99	20.57	86.12	91.89
nasa7	78.96	46.89	70.51	75.48
average(fp)	96.49	34.36	89.50	92.28
average(all)	**97.92**	**43.95**	**79.37**	**83.35**

次に，このようなロードアドレス予測が，全体性能にどの程度影響を及ぼすかを，総実行サイクル数を評価基準にして比較を行なった．以下のグラフ中で，**Base**はLACを用いないベースモデルでの総実行サイクル数，**Algo0**は，Algorithm 0でのサイクル数，**Algo1**は，Algorithm 1でのサイクル数，**Algo2**は，Algorithm 2でのサイクル数，**Ideal**は，LACが100%予測できると仮定した理想値を，それぞれ表している．今回の実験結果は，各々のアルゴリズムでの総実行サイクル数を，**Base**モデルを1とした時の比で表した．

図7はin-order実行モデルでの，各アルゴリズムごとの総実行サイクル数比を示している．整数系プログラムでは，LAC付きの総実行サイクル数はLACなしの場合に比べて，9%から25%程度減少していることがわかる．特に，**compress**, **eqntott**, **sc**においては，LAC付きの総実行サイクル数は理想値に近くなっている．これは表2で見られるように，Algorithm 2での予

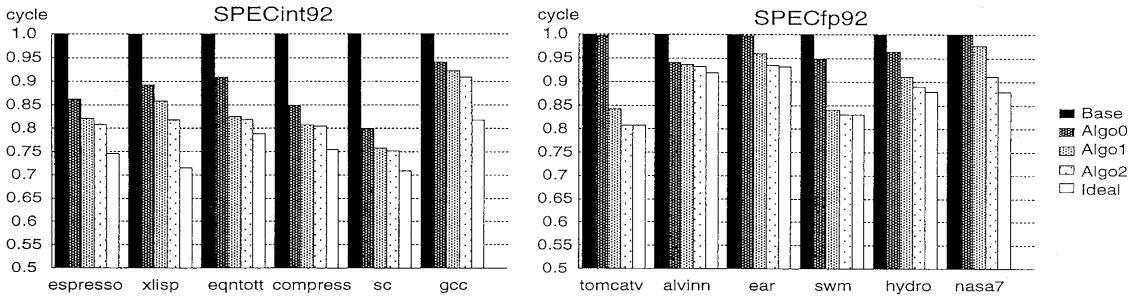

図7 各アルゴリズムでの総実行サイクル数比較(in-order 実行)
Fig. 7 Total execution cycle on in-order processor model.

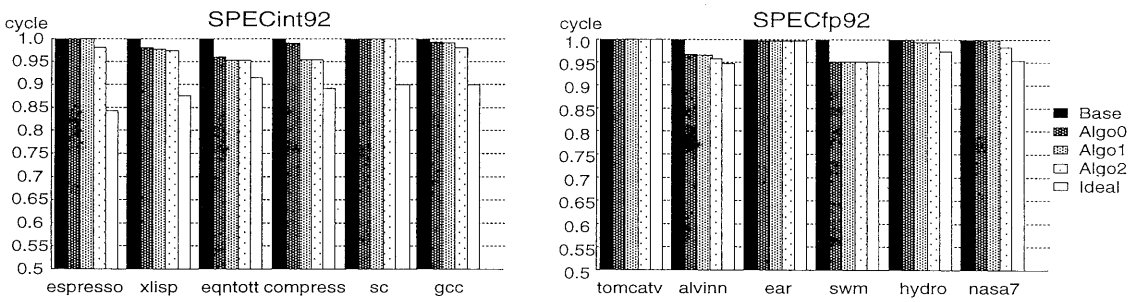

図8 各アルゴリズムでの総実行サイクル数比較(out-of-order 実行)
Fig. 8 Total execution cycle on out-of-order processor model.

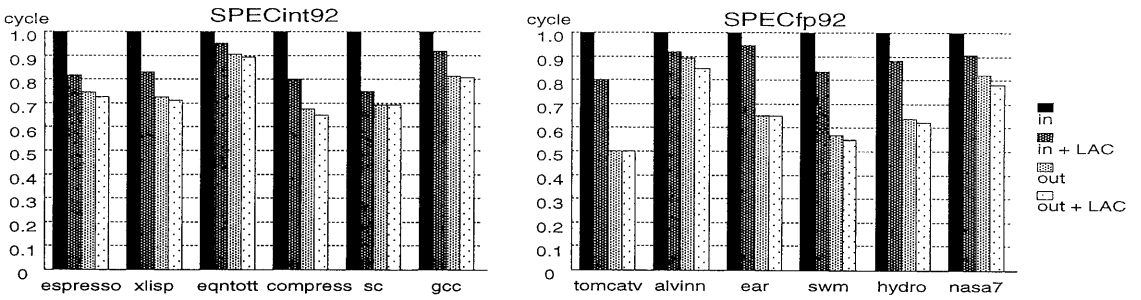

図9 In-order 実行と out-of-order 実行での LAC 効果
Fig. 9 In-order vs out-of-order processor model.

測正解率が80%を超え，非常に高いためである．これらのプログラムに対して，Instruction Per Cycle(IPC)の値は23%から33%の向上が見られた．

浮動小数点プログラムでもまた，LAC付きの総実行サイクル数は7%から20%程度減少している．特にAlgorithm 2での総実行サイクル数は理想値に近く，予測正解率もほぼ100%である(表2参照)．これは，ほとんどのロード命令の実行が数値計算ループ内でおこなわれ，そこでは定数でアドレスインクリメントを行なうような，規則的で反復的なストライドアクセスロード命令が多く実行されるからである[3]．

図8は out-of-order 実行モデルに対する総実行サイクル数を表している．ベースモデルに対して総実行サイクル数は0.03%から5%の減少が見られ，またIPCは0.03%から5%の向上が見られた．

図9はin-order実行とout-of-order実行におけるLAC効果の違いを表している．図中in-orderベースモデルでの総実行サイクル数を1とした時の各総実

行サイクル数を比で表す．**in** は in-order ベースモデル，**in+LAC** は Algorithm 2 の LAC を用いた in-order プロセッサモデル，**out** は out-of-order ベースモデル，**out+LAC** は out-of-order ベースモデルに同 LAC を用いたモデルでの総実行サイクル数比をそれぞれ表す．

in-order 実行の場合の総実行サイクル数は LAC を用いることにより最大約 25% 減少することが分かる．それに対して out-of-order の場合では，ほとんど効果が見られない．また，**espresso**，**sc** においては，**in+LAC** と **out** の値は非常に近いことが言える．LAC の機能を付加した in-order 実行のハードウエアは out-of-order 実行のハードウエアに比べて単純な機構になると思われる．従ってこれらのプログラムに対して全体性能を向上させるために LAC の機能を付け加えることは，比較的シンプルで経済的な手法であるということが言える．

5. LAC 正解率についての考察

この節では，ストライドロード命令と LAC の予測正解率との関係を考察する．

図 10(a) は動的に測定した全ロード命令の実行に占めるストライドロード命令の実行の割合である．整数系のプログラムにおいては，ストライドロード命令の実行の割合はあまり多くなく，一方浮動小数点系プログラムにおいては，半数以上のロード命令がストライドロード命令であるということが判明した．

図 10(b) はストライドロード命令についての LAC 予測正解率を示す．浮動小数点系プログラムに対しては，ストライドロード命令の LAC 予測正解確率は非常に高いことが分かる．

これらのデータから，全ロード命令中，動的に実行されたストライドロード命令の予測正解率は浮動小数点系プログラムに対しては 92% から 99% となり，非常に高いことが判明した．従って，浮動小数点系プログラムに関しては，今回提案したアルゴリズムでの，ロードアドレス予測の効果があると言える．

6. ま と め

プロセッサ高速化に向けて，総実行サイクル数を低減し命令並列度を向上させるために，ロードアドレス予測のアルゴリズム手法を提案し，その効果について実験を行なった．

その結果，ロードアドレス予測の手法はデータキャッシュレイテンシを大きく低減することができ，全体性能を向上させることができた．in-order 実行の場合，総実行サイクル数は最大約 25% 減少し，IPC は 33% 向上した．out-of-order 実行の場合は 総実行サイクル数は約 5% 減少し，IPC は 5% 程度の向上が見られた．

また本稿で提案しているロードアドレス予測の手法

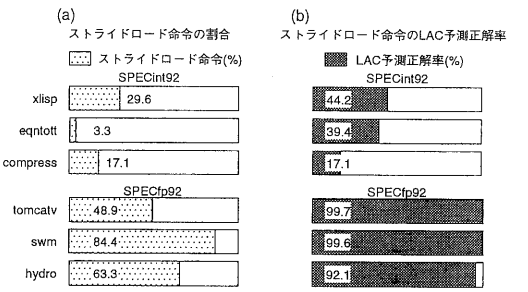

図 10 ストライドロード命令の実行割合と，LAC 予測正解確率
Fig. 10 Stride load execution ratio and LAC accuracy.

は，in-order 実行方式のプロセッサで，ストライドロード命令が多く実行されるアプリケーションプログラムに対して，全体性能を向上させるのに非常に効果的な手段であることが判明した．

今後の課題として，予測ミス時にペナルティがないようなハードウエア構成を考えることや，整数系プログラムにおいて，ロードアドレス予測の効果があるロード命令の特徴を調査する予定である．

謝 辞

本研究を遂行するにあたり，御指導御支援して下さった，津田所長代理，ならびに，小沢年弘，志村浩也 両氏に感謝いたします．また，討論していただいた研究部の皆様に感謝致します．

参 考 文 献

1) Mike Johnson, "Superscalar Microprocessor Design" 1990.
2) 勝野，木村，"ロードアドレス予測方式の検討" 情報処理学会研究会報告，96-ARC-117，March, 1996.
3) 小沢，西崎，木村，"ロード命令の先行実行とその評価" 情報処理学会研究会報告，94-ARC-109-1, December, 1994.
4) 小沢，西崎，木村，"ロード命令の先行実行の科学技術計算プログラムによる評価" 情報処理学会研究会報告，95-ARC-112-4，June, 1995.
5) Eickemeyer, R. J., et al., "A load-instruction unit for pipelined processors," IBM J. RES. DEVELOP., pp.547-564, Vol. 37 July 1993.
6) 志村，西本，江口，木村，"スーパスカラプロセッサの性能評価 -Paratool- " 情報処理学会研究会報告，93-ARC-102-1，Oct.,1993.
7) T.Ozawa, et al., "Cache miss Heuristics and Preloading Techniques for General-Purpose Programs". In Proceedings of IEEE Micro-28 NOV., 1995.