計算機アーキテクチャ 139－17
（2000．8．4）

# 命令フェッチ制御命令による命令キャッシュミスペナルティの削減

岡本秀輔†

命令フェッチ制御命令は従来から使われてきた分岐命令の置き換えとなるべく導入された命令である．分岐命令が次に実行すべき命令を指定するのに対して，命令フェッチ制御命令は次にフェッチすべき命令群を指定する．さらに，この命令がメモリからフェッチされる際には，デコードすることなく，他の通常の命令と区別されて，命令群のプリフェッチに利用される．本報告では，この命令を用いた命令プリフェッチにより，いかに命令キャッシュのミスペナルティが削減されるかについて述べる．

# Reducing I-Cache Miss Penalty by I-Fetch Instruction

SHUSUKE OKAMOTO†

Instructions for fetching instruction is the replacement of conventional branch instructions, which we have recently introduced. An instruction of this type gives information of next basic block to be fetched. It is recognized by processor without decoding, and is processed in parallel with the other types of instructions. It is also used as the special instruction to prefech instructions. This report shows that the processor using instructions of this type can reduce the instruction miss penalty by prefetching.

## 1. はじめに

命令キャッシュのプリフェッチは，命令キャッシュのミス率を抑えるとともに，ミス・ペナルティを減らす重要な技術である．その重要性は，プロセッサ性能の向上とともに高まっている．ギガヘルツの周波数で動作するプロセッサや命令レベルで高度に並列化されたスーパスカラ・プロセッサは，単位時間当たりに非常に多く命令を消費する．しかし，一般に知られているように，プロセッサとメモリとの性能差は広がるばかりであり，両者間の性能差を隠すためには，命令キャッシュのミス率を抑え，かつ，ミス時のペナルティを減らすことが必要である．これらの対応なくして，今日のプロセッサの性能を高く維持することは難しい．

従来のプロセッサにおいて行われて来た命令フェッチは，プログラム・カウンタに密接に関係した，プロセッサのデフォルトの動作である．物理的に隣接して配置された命令に対して連続したアドレスを振り，そのアドレスを数えるカウンタによって，次に実行すべき命令が自動的に決まるというものである．ブランチやジャンプといった分岐命令が，プログラム・カウンタの数え上げの動作に変更を与えて，制御の流れを変更することができる．したがって，プロセッサは分岐命令を見つけるまでは，連続アドレス上の命令をフェッチするのみである．分岐命令を見つけると，分岐先アドレスを計算し，そして，分岐先アドレスにある命令を次に実行すべき命令としてフェッチする．条件分岐の場合には分岐条件が満たされているかを判定するという動作が加わる．このように，分岐命令は「次に実行すべき命令」を他に変更する命令であるために，プロセッサが分岐命令を見つけてから次の命令をフェッチするまでに時間的な余裕が少ない．そのため，命令キャッシュ・ミスが起こった場合にはそのペナルティを直接に受けてしまう．これに加えて，隣接命令のフェッチに比べて，分岐先の命令のそれはキャッシュ・ミスを起こしやすいという点もまた，キャッシュ・ミスによる性能低下をもたらす要因となっている．

しかしながら，基本ブロックを単位としたプログラムの制御の流れの解析から明らかなように，あるブロックからの分岐先のアドレスは固定している場合が多い．また，分岐先アドレスが固定しているならば，そのアドレスにある基本ブロックのサイズもまた定まっている．もし，プロセッサが基本ブロックの先頭の命令でこれら次の分岐先ブロックの情報を得たならば，その時点から実際に分岐を必要とする時点までがプロセッサにとって時間的な余裕となる．つまり，この間にプロセッサはキャッシュへの命令のプリフェッチを行ったり，分岐条件が定まるタイミングを正確にとらえる

† 茨城大学 工学部情報工学科
Department of Computer and Information Sciences, Faculty of Engineering, Ibaraki University.

といったことが可能となる．そして，プロセッサが次のブロックのサイズを知ることにより，不要な命令のフェッチを確実になくすことができる．

現在の分岐命令は，分岐先アドレスの情報を保持しているが，この命令は基本ブロックの最後に置かれるので，上述の意味の時間的な余裕はない．BTB(Branch Target Buffer) などの技術により，分岐命令をフェッチし，デコードして，アドレス計算を行う期間分だけ早期に分岐先アドレスを得ることは可能ではあるが，基本ブロックの先頭で分岐先アドレスを得るほどには早期に行うことができない．そして，次の基本ブロックのサイズは，次の分岐命令を見つけるまでは分からないために，命令のプリフェッチを行った場合に不要な命令フェッチを行ってしまう可能性がある．

命令フェッチを制御する命令は，「次にフェッチすべき命令群」を指定する．この種の命令のどれもが基本ブロックの先頭に置かれる．プログラムは基本ブロック単位で解析されるので，その制御の流れを変更する命令もブロック単位で指定し，かつ，基本ブロックの先頭でその情報をプロセッサに与えるべきという上記の考えを実現したものになっている．そして，これは従来の分岐命令で指定される「次に実行する命令」と異なった情報をプロセッサに与える．

本報告では，次のブロックの情報を従来よりも早期に得ることができるプロセッサが，情報を得てから分岐するまでに命令キャッシュへのプリフェッチをどの程度行うことができるかについて述べる．このための評価は命令パイプライン・レベルの動作をシミュレートするソフトウエアを用いて行った．命令キャッシュへのプリフェッチは，他のプロセッサの動作と並行して行うことでその効果が現われる．したがって，評価の方法として，プリフェッチにより避けられない命令キャッシュのミスを早期に起こし，どれだけ他の処理と並行してミスの処理が行えるか，つまり，ミス・ペナルティをどれだけ減らすことができるかを調べた．

## 2. 命令フェッチ制御命令

命令フェッチ制御命令は次にフェッチすべき命令群を指定するものであるが，その役割は従来の分岐命令から変更を加えた形になっている．したがって，その種類も分岐命令と同じものである．すなわち，無条件に命令フェッチを行うように指定する場合，条件付きで分岐先の命令フェッチ行うように指定する場合，手続き呼び出しの場合，そして，フェッチする命令群が動的に決まる場合，の4種類である．

### 2.1 無条件命令フェッチ

無条件命令フェッチは，無条件ジャンプに対応した命令である．これは

```
f L1, 10
```

という形式で指定する．この命令の意味は，L1 というラベルから始まる 10 個の命令からなるブロックを，現在のブロックの後に，無条件にフェッチする という意味になる．現在のブロックとは，この f 命令を含んだ命令ブロックであり，このブロックのサイズは，1つ前のフェッチ制御命令によって指定されている．図1(a) は基本ブロック中でどこにこの命令が配置されるかを示している．上側の基本ブロックの先頭で次のブロックの情報をプロセッサは得る．下の基本ブロックがキャッシュになければ，上のブロックを処理している間にメモリからキャッシュへプリフェッチできる可能性がある．

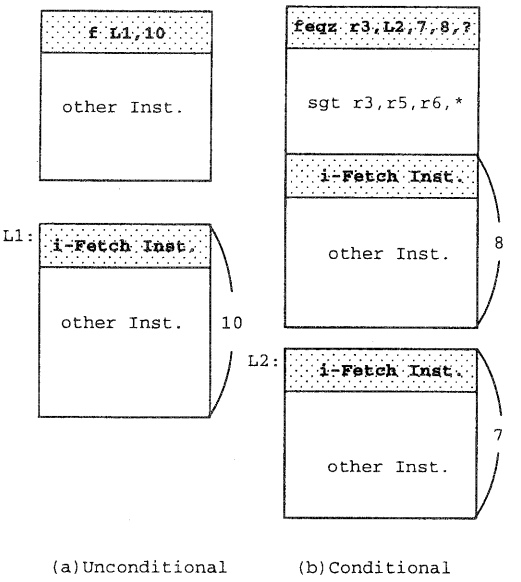


図1 I-Fetch Instructions

### 2.2 条件命令フェッチ

条件命令フェッチは，選択すべき2つのブロックの情報と選択条件を指定する．分岐命令と同様に選択のうちの片方は隣接ブロックとして，先頭アドレスを省略してサイズのみを指定する．図1(b) は条件命令フェッチの例を示している．

```
feqz r3,L2,7,8,?
```

は，レジスタ r3 がゼロの場合には，L2 から始まる7個の命令からなるブロックを，非ゼロならば隣接で8個の命令からなるブロックを次にフェッチすべき命令として選択する という意味になる．レジスタ r3 に値が正しく設定されることを保証するためには，r3 を設定する命令と同期をとる必要がある．例えば命令 sgt がレジスタ r3 を設定する命令ならば，これに ''*'' というオペランドを付加し，feqz 命令に ''?'' を付加することで，両者の命令に同期が必要であるということを指定する．この同期のためのオペランド指定は，他種のフェッチ制御命令につけても良い．

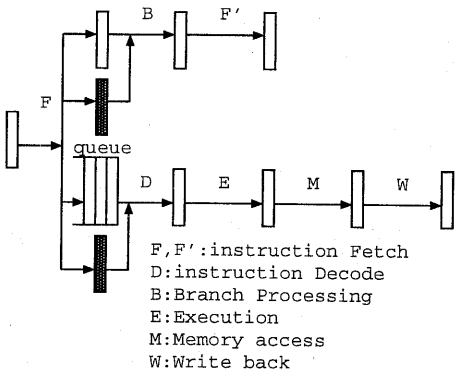


図2 Pipeline Structure

### 2.3 手続き呼び出しとリターン

手続き呼び出しのためのフェッチ制御命令 **fal** は，無条件フェッチと同じ指定の仕方をするが，実行時には次のブロックのフェッチ行うとともに，この命令を含むブロックの隣接ブロックの先頭アドレスをレジスタに保存する．隣接ブロックは手続きからの戻りブロックを意味する．これは，MIPS プロセッサなどのジャンプ・アンド・リンク **jal** 命令と似た効果をもたらす．

**fal** 命令が戻り先のアドレスのみでそのサイズを保存しない代わりに，**fr** 命令が戻り先のアドレスを含んだレジスタとそのブロックのサイズを指定する．**fal** 命令と **fr** の関係は命令設計上の 1 つの選択であり，これにより既存のコードからの変換をやさしくするという効果を持つ．その半面において，ある手続きを呼び出す命令を含んだブロックの隣接ブロックは必ず同じサイズでなければならないという制約がある．

### 2.4 既存のコードからの変換

分岐命令を含んだ既存のプログラム・コードからフェッチ制御命令を用いたコードに変換するには，まず，基本ブロック単位の解析を行い，各ブロックのサイズを得る．次に，各基本ブロックごとに分岐命令を削除して，削除した分岐命令が指していたラベルの基本ブロックの情報を持つようなフェッチ制御命令をブロックの先頭に挿入する．ただし，分岐命令を持たない基本ブロックも存在するので，そのようなブロックには無条件命令フェッチを加えて，ブロックサイズを調整する．上述の手続き呼び出しの戻りブロックの問題に対しては，パラメータを置いたスタックを調整するためのコードとして，固定サイズのブロックを戻りブロックに常に置くようにする．手続き呼び出しの隣接コードには，つねにこの固定サイズのブロックが置かれるので，**fr** 命令のサイズも固定にすることができる．以上のような変換により，既存のコンパイラの出力を使うことが可能となる．

## 3. 命令パイプライン構成

後の実行例および評価における前提となるプロセッサの命令パイプライン構成を図 2に示す．四角がパイプラインバッファを表し，矢印が命令の流れを表す．2 つの四角の間がパイプラインのステージを表す．プログラム中の基本ブロックの先頭には常にフェッチ制御命令が置かれている．したがって，他種の命令との区別にデコードを必要としない．そこで，フェッチ制御命令はフェッチステージ **F** から直接に上側のラインを進み，その他の命令は下側のラインを進む．上側のライン の **F'** ステージはフェッチ制御命令の実行が行われる場所をしてしており，この実行が左の **F** ステージの動作となる．網かけのついた四角は，分岐条件の設定待ちの間に，キャッシュからプロセッサへプリフェッチした命令を保持するために使用される．

## 4. 実行トレースの例

簡単なプログラム・コードの例とその実行トレースについて説明する．以下で述べるコードにおいてフェッチ制御命令以外の命令は，DLX プロセッサ [2] という，MIPS プロセッサに類似したロード／ストア型の RISC プロセッサのアセンブリ命令を用いている．また，命令パイプライン実行において，1 サイクルに 2 命令の命令をフェッチすることを仮定する．そして，分岐条件の設定待ちの間にいくつかの命令をプロセッサ内にプリフェッチする．その方法は，単純に，隣接ブロックの 2 命令をプリフェッチした後に，分岐先ブロックの 2 命令をプリフェッチする．

図 3(a) のコードは 4 つのブロックからなり，それぞれ先頭にフェッチ制御命令が置かれている．**start** ラベルのブロックには，条件命令フェッチが指定されており，**target** ラベルのブロックか隣接となる **fall** ラベルのブロックをレジスタ **r3** の値により選択する．

図 3(b) はコードの実行トレースを示している．"clk" の列は実行クロックサイクルを，"n" の列はブロック内のフェッチすべき残りの命令数を示している．"F" の列はフェッチステージの動作であり，"*" が通常の命令フェッチを，"p" が条件待ちのプリフェッチを示す．"B" および"D" の列は処理された命令を示し，"E", "M" そして"W" の列は，"*" によりそのステージで処理が行われていることを示す．

**FEQZ** 命令はクロック 0 でフェッチされて，**sgt** 命令の設定したレジスタ **r3** の値が有効になるクロック 7 で，その分岐判定のために処理が行われる．つまり，クロック 1 ～ 6 の間に分岐先アドレスを計算し，分岐先ブロックのいくつかの命令をキャッシュへプリフェッチする時間的余裕があることが分かる．この例では，クロック 4 で **fall** ラベルのブロック中の **F** 命令と **sle** 命令がプリフェッチされ，クロック 5 で分岐先である

```
start:  FEQZ    r3,target,5,3,?
        addi    r1,r0,#3
        addi    r2,r0,#2
        addi    r5,r0,#5
        sgt     r3,r2,r1,*

fall:   F       end,2
        sle     r3,r2,r1
        and     r4,r3,r1

target: F       end,2
        slt     r6,r2,r1
        subi    r3,r5,2
        and     r4,r3,r1
        sle     r3,r2,r1

end:    F       nowhere,0 ;halt
        add     r1,r0,r0
         ( a )
```

```
 clk| n| F| B  :  D  |EMW|
 ---+--+--+---+-----+---+
   1| 5| *|    :     |   |
   2| 3| *|    : ADDI|   |
   3| 1| *|    : ADDI|*  |
   4| 0| p|    : ADDI|** |
   5| 0| p|    :  SGT|***|
   6| 0|  |    :     |***|
   7| 0|  |FEQZ:     | **|
   8| 3| *|    :  SLT|  *|
   9| 1| *|F   : SUBI|*  |
  10| 2| *|    :  AND|** |
  11| 0|  |F   :  SLE|***|
  12| 0|  |    :  ADD|***|
  13| 0|  |    :     |***|
  14| 0|  |    :     | **|
  15| 0|  |    :     |  *|
        ( b )
```

**図3** Sample Code and its Trace

**target** ラベルのブロックにある **F** 命令と **slt** 命令がプリフェッチされる．そして，分岐判定の次のサイクルのクロック8で **slt** 命令のデコードが行われている．もし，クロック5でのプリフェッチがキャッシュミスを起こしたとしても，2クロック分のペナルティは隠蔽できる．また，**target** ブロックと **end** ブロックは2つのブロックが一つのブロックであるかのように連続して処理されている．

## 5. プリフェッチによる効果

命令パイプラインレベルの動作を検証できるソフトウエアシミュレータを用いることで，分岐条件設定待ちにおける命令プリフェッチが，命令キャッシュの振舞いにどのような影響を与えるかを調べた．

このシミュレータでは，表1に示すような機能ユニットを仮定して，DLX プロセッサおよび DLX の分岐命令をフェッチ制御命令に置き換えたプロセッサの実行をシミュレートする．これに加えて，命令とデータにそれぞれ 16KB のレベル1キャッシュを備えている．このキャッシュは，1ラインが32バイト (命令とデータはともに4バイト)，直接マップであり，データキャッシュに関しては，ライトスルー，書き込みミス時に割当をしない方針をとっている．

表1　機能ユニットの特性

| 機能ユニット | レイテンシ | 発行可能間隔 |
|---|---|---|
| integer ALU | 0 | 1 |
| Load/Store | 1 | 1 |
| FP add | 3 | 1 |
| FP multiply | 6 | 1 |
| FP divide | 24 | 25 |

シミュレーションでは6つの種類のプロセッサについて調べた．そのうち4つはフェッチ制御命令を用いるものである．"f1" は1サイクルに1命令フェッチを行い，"f2" は1サイクルに2命令フェッチを行う．"f1p" と"f2p" はそれぞれの命令フェッチ数に加えて，条件設定待ちの間にプリフェッチを試みる．残りの2つは DLX プロセッサであり，dlx が1サイクルに1命令フェッチを行い，dlx2 が1サイクルに2命令フェッチを行う．

ベンチマークプログラムとして，DLX プロセッサ用の GNU C Compiler 1.37 の中の cc1 をシミュレータ上で実行させた．コンパイル後の命令数は，f1,f1p,f2,f2p 用のコードが 247,200 命令．dlx 用のコードが 265,870 命令であった．dlx 用のコードは遅延分岐のために遅延スロットの多くが **nop** 命令である．この **nop** 命令を除いた命令数は，232,239 命令であった．この結果から，前述の，付加的に挿入されたフェッチ制御命令の数はそれほど多くないことが分かった．

図4は，各プロセッサにおけるプログラム実行時間を，dlx に対する相対時間で示したものである．横軸のキャッシュミスペナルティを変えて測定した．命令キャッシュのミス率は，2.3%〜2.9%であった．この図の f1 と f1p および f2 と f2p の差からプリフェッチにより性能が向上していることが分かる．f2p の dlx に対する最大の性能差は 15.5%である．f1 が dlx より遅くなっている部分があるが，この理由として次のことが考えられる．実行しているコードのパス上に小さいサイズの基本ブロックが連続して現われた場合には，f1 の命令キューは空になっている場合が多くなる．基本ブロックの先頭は常にフェッチ制御命令であるために，1サイクルに1命令のフェッチを行っただけでは，デコーダに送る通常の命令は命令キューに入らない．したがって，ブロックごとで dlx に比べて常に1クロックサイクルだけ送れて通常命令の処理が始まることになる．そのために，分岐条件が定まる時期も dlx より1クロックサイクル遅くなり，そのために，dlx よりも性能が悪くなっている．

図5は命令キャッシュのミスペナルティの削減を示したものである．図4の実行において，命令キャッシュミスが起きている間にデコードすべき命令が命令キューの中に存在している場合を数えたものである．つまり，ミスペナルティは他の処理が行えることによって，見

かけ上隠蔽されている．この図からプリフェッチを行っている f1p と f2p が高い確率でペナルティを隠す効果があることが分かる．

## 6. おわりに

本報告では，命令フェッチ制御命令を用いた命令プリフェッチが命令キャッシュの動作に対してどのような影響を与えるかについて述べた．命令フェッチ制御命令は，次にフェッチすべき命令群の情報として，ブロックの先頭アドレスとブロックのサイズをプロセッサに与える．そして，すべての命令フェッチ制御命令をブロックの先頭に配置することで，他の命令とのデコードなしの分離を可能とし，この命令がフェッチされてから実際に分岐処理が起こるまでの間に時間的余裕を与える．プロセッサは 1 サイクルに複数命令をフェッチし，さらに，分岐処理までの間に命令のプリフェッチを行うことで，命令キャッシュミスの間に他の命令の処理を行うことができる．シミュレータの用いた評価では，約 50%の割合で命令キャッシュのミスペナルティを，見かけ上，削減されることが示された．

今回の評価では，分岐予測の技術についての考慮がなされていない．シミュレートされたプロセッサがプリフェッチを行う場合には，まず，隣接ブロックの命令をプリフェッチして，その次に分岐先ブロックの命令をプリフェッチする．したがって，この逐次的なプリフェッチはサイズの小さいブロックが連続するような場合には，うまく働かない．プリフェッチに，分岐予測を導入することで，無駄な命令フェッチ動作を予測の確率に応じて減らすことができ，さらなるプリフェッチの効果を性能向上に役立てることができると考えられる．

またこれとは別に，フェッチ制御命令が指し示している次のブロックの先頭アドレスには，次のフェッチ制御命令が存在している．この事実を利用することで，フェッチ制御命令だけをたどって，実行時に，ループを検出することができる．これにより，ループとして検出された命令を命令キャッシュ上にとどめておくことができるかもしれない．これに対する具体的な方法は現在検討中である．

**謝　辞**

本研究の一部は文部省科学研究費補助金 (課題番号 11780199) の補助による．

## 参考文献

1) Conte, T., Menezes, K., Mills, P., and Patel, B. Optimization of instruction fetch mechanisms for high issue rates. *Proceedings of the 22nd Annual International Symposium on COMPUTER ARCHITECTURE*(1995).
2) Hennessy, J. L. and Patterson, D. A. *Computer Architecture – A Quantitative Approach, Second Edition.* Morgan Kaufmann(1996).
3) Pierce, J. and Mudge, T. Wrong-path prefetching. *Proceedings of the 29th Annual IEEE/ACM International Symposium on MICROARCHITECTURE*, pages 264 – 273(1996).
4) Rotenberg, E., Bennett, S., and Smith, J.E. Trace cache: a low latency approach to high bandwidth instruction fetching. *Proceedings of the 29th Annual IEEE/ACM International Symposium on MICROARCHITECTURE*(1996).
5) Luk, C. and Mowry, T. C. Cooperative prefetching: compiler and hardware support for effective instruction prefetching in modern processors. *Proceedings of the 31th Annual ACM/IEEE International Symposium on MICROARCHITECTURE*, pages 182 – 194(1997).
6) Reinman, G., Calder, B., and Austin, T. Fetch directed instruction prefetching. *Proceedings of the 32th Annual ACM/IEEE International Symposium on MICROARCHITECTURE*(1999).
7) Okamoto, S., and Sowa, M. Instruction Set Architecture to Control Instruction Fetch on Pipelined Processors. *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM'97)*, Vol. 1 of 2, pp.121-124(1997).
8) 岡本秀輔, 曽和将容 命令フェッチを制御する命令を持ったアーキテクチャの定性的な評価. 情報研報, Vol.97, No.22, pp49 ～ 54(1997).
9) 岡本秀輔, 曽和将容 命令フェッチをプログラム制御するプロセッサ・アーキテクチャ. 情報処理学会論文誌, Vol.39, No.8, pp.2509-2518(1998).
10) Okamoto, S., and Takata K. Reducing Cache Miss Penalties Using I-Fetch Instructions. *14th Annual International Symposium on High Performance Computing Systems and Applications(HPCS2000)*, to be published in a book by Kluwer Academic Publishers.

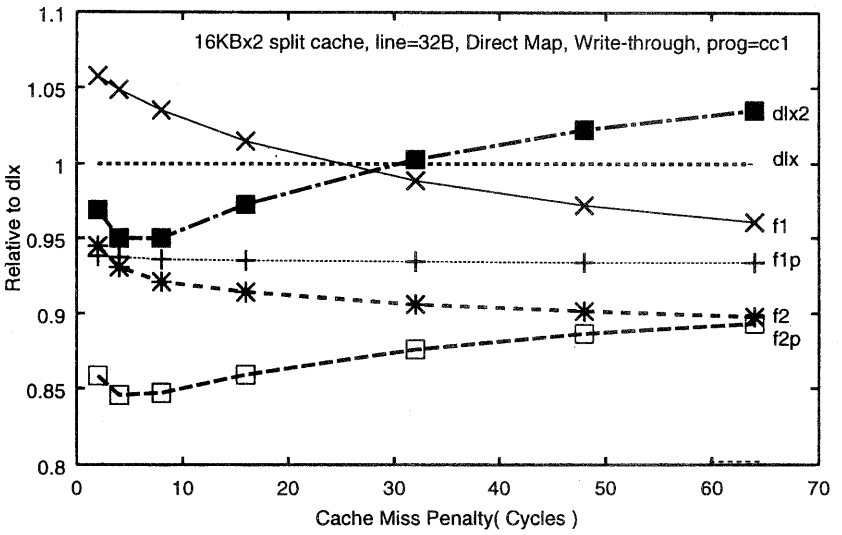


図 4 Relative Execution Time

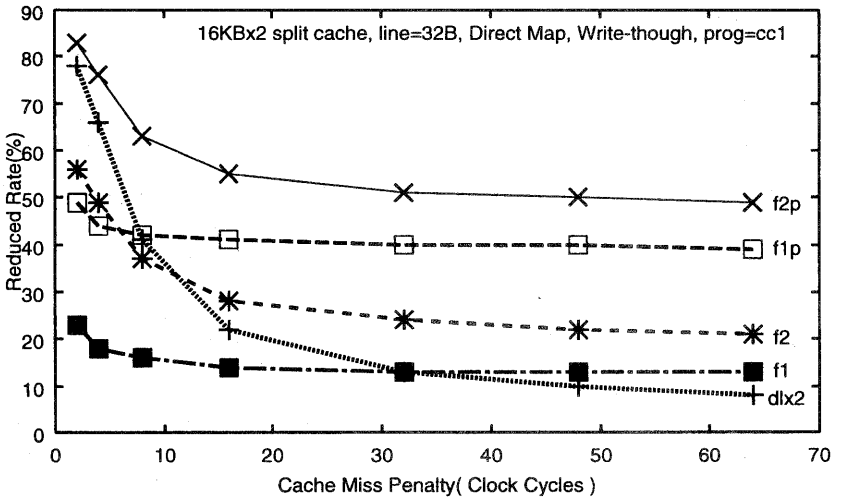


図 5 I-cache Miss Penalty Reduction