計算機アーキテクチャ 119－14
（1996. 8. 27）

フェッチ分岐方式の提案

岡本秀輔† 曽和将容†

パイプラインプロセッサにおける分岐ハザードを削減するための一つのアプローチとして、フェッチ分岐方式を提案する。この方式では、プロセッサは分岐命令をフェッチステージで識別して、他の命令の実行とともに、並列に分岐条件決定の監視を始める。そして、分岐条件の決定の直後に分岐処理を行う。分岐処理を可能な限り早期に行うので、データハザードとの組合せによりゼロサイクル分岐が起こり得る。本報告では、このフェッチ分岐方式を実現するための命令セットアーキテクチャと、2つのプログラム例を用いた本方式の効果について述べる。

Branch on Fetch Method for Pipelined Processors

SHUSUKE OKAMOTO† and MASAHIRO SOWA†

We propose a new branch method which is an approach to reduce the branch hazard in the pipelined processors. In this method, the processor can identify a branch instruction on fetch stage of pipeline and it begins to watch the determination of the branch condition. So it can begin to process the branch as soon as the branch condition is set. Because of this early branching method, the zero cycle branch may occur. This paper describes the instruction set architecture as well as the effect of this method with showing two sample programs.

1. はじめに

プロセッサの高速化の技術として、命令のパイプライン処理は今やなくてはならないものとなっている。ところが、この命令パイプラインでは、一般に、フェッチした命令が適切だったか否かの判定が遅れる分岐ハザード[1]が生じ、これがパイプラインのストールの原因となって、プロセッサの性能を下げる。

分岐ハザードによるパイプラインのストールを減らす方法としては、遅延分岐, 分岐予測, 命令バッファ(多重プリフェッチ方式, 分岐の折り込み) など[2)~7)]があげられるが、それぞれに共通の原因から生じる問題を持っている。

例えば、遅延分岐では、分岐条件の決定とは無関係に遅延スロットが設定されている。そして、遅延スロットに入れることのできる命令には制限があるために[3)]、適当な命令が見つからずにNOPを実行しなければならない状況が多くある。

また、分岐予測では、予測の当たる確立が実行しているアプリケーションに左右されやすく、予測ミスは大きなペナルティを被るが、一般に、早期に分岐条件が決定されても予測の段階ではそれを使用せずに、後の分岐命令の実行によって始めて結果を知るために、予測ミスが増えてしまう。

このようにこれらの方式では、分岐条件が早期に決定し、実際に分岐するまでに、演算命令等を実行などの、時間的余裕がある場合においても、分岐のための処理は分岐の直前まで始まらない。これは分岐指定の分岐元が、分岐命令の場所 (またはその次) にデフォルトで固定されていることに起因する。分岐命令に対するフェッチ／デコードを含めた処理は、分岐の直前に行なわれるので、短い時間内で分岐に関する全ての処理を行なわねばならなず、結果として、分岐ハザードは依然として残ってしまっていると思われる。

このように考えると、分岐条件決定から実際に分岐するまでの時間的余裕を有効利用することで、分岐ハザードを解消する方法が考えられる。そして、それを可能にする方法として、分岐を指定する命令を他の命令とは並列に実行する方法が考えられる。

そこで、分岐条件の決定の直後に分岐処理を開始できる方法の一つとして、分岐元を指定できる命令を用意して、その命令を早期に開始するフェッチ分岐方式を提案する。

提案するフェッチ分岐方式の特徴は以下の様である。

- 分岐を指定する命令とその他の命令との分離を、命令パイプラインのフェッチステージで行い、フェッチ命令とその他の命令とを並列に処理する。
- 分岐条件決定の後すぐに分岐処理の開始を可能に

† 電気通信大学 大学院 情報システム学研究科
The Graduate School of Information Systems,
The University of Electro-Communications

する。

- 分岐元と分岐先の指定情報をコンパイラが扱う基本ブロックとし、基本ブロック内の命令を、可能な限りプリフェッチする。
- 状況によりゼロサイクル分岐が可能。

本報告では、フェッチ分岐方式の提案とともに、それを実現するための命令セットアーキテクチャについて述べる。そして、フェッチ分岐方式によって得られるいくつかの利点と解決すべき問題について考察を行なう。

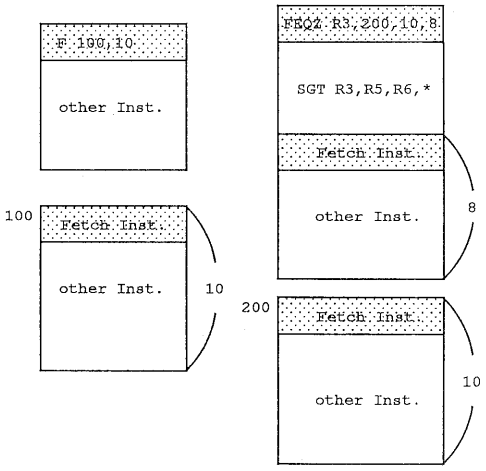
2. フェッチ分岐方式

命令のパイプライン処理により、命令のフェッチとその実行とは区別され、分岐命令の実行は命令のフェッチに影響を与えるので、今や、分岐命令は「次に実行すべき命令をデフォルトから変更する」命令から「次にフェッチすべき命令をデフォルトから変更する」命令へとその役割が変わったと見ることができる。

この部分に着目し、前述の分岐元は分岐指定をする命令からというデフォルトルールを変えることを目的として、分岐命令の代わりに、どこからどこまでフェッチするかを指定するフェッチ命令を導入する。このフェッチ命令の指定では、フェッチすべき命令をコンパイラが扱う基本ブロックとする。つまり、分岐処理も基本ブロックを単位となる。基本ブロック内の命令は必ず実行されるとし、命令キューといった中間バッファを用いて、可能な限り命令のプリフェッチを行なう。

次に、このフェッチ命令とその他の命令の並列処理を考える。考慮すべき点は、フェッチ命令とその他の命令をどの段階で区別して、並列実行を開始するかである。一般に、分岐ハザードが生じる原因の一つとして、分岐命令の識別の遅れあげられる。言い替えるとデコードステージまで分岐命令を識別できず、この遅れが分岐ハザードとなる。パイプライン実行では分岐命令の識別が遅れることで、分岐命令以降に配置された命令の処理をどうするかといったこと考えなければならない。したがって、分岐を指定する命令をフェッチした時点で識別することができれば、このような問題は生じない。

これと類似する目的を達成する方式には Branch Target Buffer(BTB) がある[7]。BTB は分岐命令をキャッシュすることで、パイプラインのフェッチステージにおいて分岐命令を識別する。しかし、BTB は過去に分岐した命令の情報を残しているが、ここで実現しようとしているフェッチ命令の並列実行のためには、すべてのフェッチ命令の情報が必要であり、それを達成するには単純に考えてもバッファのサイズをかなり増大させねばならない。また、あくまでも BTB はキャッシュであるので、フェッチの 1 回目は情報が



(a) Unconditional Fetch (b) Conditional Fetch

図 1 program samples using branch on fetch method

得られない。したがって、他の方法を用いてフェッチ命令を識別しなくてはならない。

そこで、本方式では、フェッチ命令は各基本ブロックの先頭に配置することで、場所によるフェッチ命令の識別を行なうことにした。これにより、分岐の後にフェッチした命令は必ずフェッチ命令となるので、フェッチ命令と他の命令との区別はフェッチステージにおいて可能となる。

図 1 はフェッチ命令による (a) 無条件フェッチと (b) 条件フェッチの例である。網かけ部分がフェッチ命令であり、フェッチ命令とそれに続く四角の部分が基本ブロックを表している。

(a) の上の基本ブロックのフェッチ命令"*F 100, 10*"は、「次にフェッチする命令は 100 番地から 10 命令で構成される基本ブロック」という意味である。すべてのフェッチ命令は、現在フェッチしている基本ブロックがフェッチし終った後、次にフェッチすべき基本ブロックを指定する。この例のような無条件フェッチ命令では、フェッチ命令置かれている基本ブロック (上側のブロック) が命令キューに全てフェッチされた後、すぐに指定されたブロックのフェッチを開始できることを意味する。

(b) の"*FEQZ R3, 200, 10, 8*"は、「次にフェッチすべきは、もし、R3 が 0 ならば 200 番地から 10 命令で構成される基本ブロックであり、そうでないならば、現在フェッチしている基本ブロックの次の番地から 8 命令で構成される基本ブロックである」という意味である。条件となる R3 を決定する命令"*SGT R3,R5,R6,**"とこの FEQZ 命令とで同期がとられる。同期をとる必要があるかどうかは、現段階では、フェッチ命令と対応する命令の最後のオペランドに"*"を示すことで明示的に指定する。

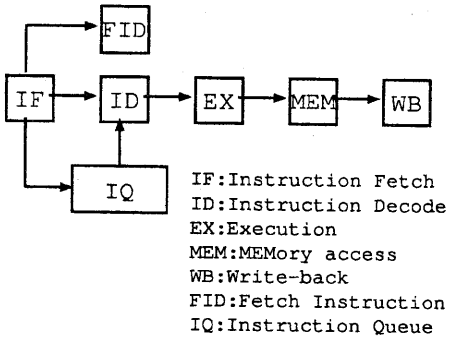


図2　pipeline structure

まとめると、フェッチ分岐方式では、フェッチ命令を各基本ブロックの先頭に配置する。その指定では、現在の基本ブロックのフェッチが終った後に、次にフェッチすべき基本ブロックを指定する。フェッチ命令はパイプラインのフェッチステージでその他の命令と分離され、そして並列実行される。そのために、分岐条件が決定した後すぐに次のブロックのフェッチの準備をすることができる。

3. プログラム例

2つの特徴的な例を示すことにより、フェッチ分岐方式の利点を示す。前提として、フェッチ分岐方式では図2の様なパイプライン処理がなされるとする。フェッチ命令は、IF でフェッチされた後、フェッチ命令を専用にデコードする FID へと流れる。その他の命令は、基本的に ID へ流れるが、パイプラインストール等で IF から ID へ進められない場合には、フェッチした命令は命令キューである IQ へと流れ、その後フェッチした順番を保つように処理が進められることとする。例では、比較のために遅延分岐を行なう DLX プロセッサ[3]を用いるが、このパイプライン構成は、図2から FID, IQ を除いたもので、分岐処理は ID で行なわれると仮定している。

3.1　ゼロサイクル分岐

基本ブロックの実行中には、ロード命令の実行、データ・キャッシュミス、浮動小数点演算のマルチサイクル実行などによりデータハザードが生じてパイプラインがストールする状況が生じる。フェッチ分岐方式では、このような状況でも連続してフェッチを続けるために、状況によっては、フェッチ命令の処理をこのストール中に行い、結果としてゼロサイクル分岐となる。

例えば、

```
for(i=0; i<N; i++) A[i] += 1;
```

のような、配列の全ての要素に1を加算する処理を考える。r3 を配列の各要素の番地 (A+4*i), r5 を変数 i、r6 を N、r7 を配列 A の先頭番地とすると、DLX ではこのループは図3のようになる。

```
                1   2   3   4   5   6   7   8   9   10
slli   r3,r5,#2 IF  ID  EX  MEM WB
add    r3,r7,r3     IF  ID  EX  MEM WB
add    r5,r5,#1         IF  ID  EX  MEM WB
sle    r1,r5,r6             IF  ID  EX  MEM WB
lw     r4,0(r3)                 IF  ID  EX  MEM WB
addi   r4,r4,#1                 IF  ID  st  EX  MEM
bnez   r1,L5                        IF  st  ID
sw     0(r3),r4                         IF  ID
-------------------------------------------------------
slli   r3,r5,#2                                 IF
```

図3　loop iteration on DLX

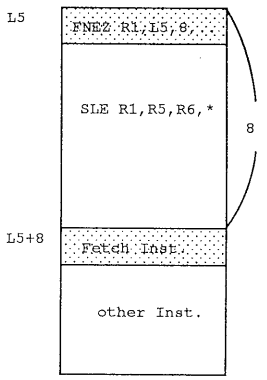


図4　simple loop

先頭の *slli* 命令から *sw* 命令までがこのループの1イタレーションであり、先頭の *slli* 命令が L5 番地にあるとしている。この例では8サイクル目に *lw* 命令の結果待ちのために、1サイクルのストールが生じている。結果として、次のイタレーションの開始は10サイクル目からとなっている。*sle* 命令の実行を次のフェッチ分岐の比較のために、早めに実行しているがそのことによる問題はない。

フェッチ分岐でこのような単純なループを構成するには、フェッチ命令が配置されている番地をフェッチ変更先の指定とすれば良い。図4はこの単純なループを示したものである。これは DLX コードの *bnez* 命令をフェッチ命令として置き換え、ループを構成する基本ブロックの先頭に配置したものと同じである。ループ後の処理を決めていないので、このフェッチ命令の最後のオペランドは仮に？としている。そして、このフェッチ命令が検査する分岐条件の r1 は、*sle* 命令が計算するために、同期指定の"*"を追加している。

このループの実行は図5のようになる。この例では、*fnez* 命令は、FID において分岐条件が決定するまで、ストールしている。7サイクル目で *sle* 命令が分岐条件を計算し、8サイクル目で *fnez* 命令の分岐処理が行なわれる。8サイクル目では同時に、基本ブロック内のすべての命令のフェッチが終了しているために、9サイクル目から次のイタレーションのフェッチが開始されている。この9サイクル目では、*lw* 命令によるストールが DLX のコードと同様に生じているが、フェッ

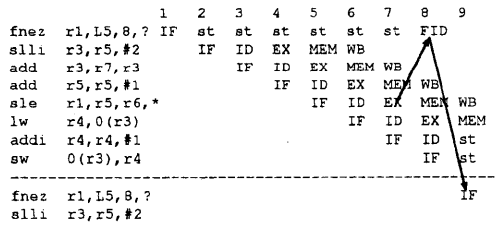


図5 loop iteration with branch on fetch

```
                      1   2   3   4   5   6   7   8
    fnez r3,L1,1,2    IF  st  st  st [FID]
    subi r3,r1,#2,*       IF  ID  EX  MEM WB
    subi r4,r2,#2,*           IF  ID  EX  MEM WB
L1: fnez r4,L2,??,2                   [IF] FID
    addi r1,r0,#1                         IF  ID  E
    f    L2,??                            IF  F
    add  r2,r0,r0                             I
L2:
```

図6 branch to select size

チの操作とは無関係である。そして、１０サイクル目に *slli* 命令のフェッチが開始され、それはストールすることなく、処理が進められる。

まとめると、*fnez* 命令は、*lw* 命令によるストール中にフェッチされ、その後の実行は他の命令と並列に行なわれるために、ループのイタレーションとしてみると、分岐処理のための時間が完全に隠れてしまっており、ゼロサイクル分岐と同じ効果をもたらしている。

3.2 命令数選択分岐

条件フェッチ命令の指定の仕方によって、分岐先が同じであるがフェッチすべき命令数が異なるような指定をすることができる。このような指定を特に「命令数選択分岐」と呼ぶが、この場合、分岐条件が決定する前に、次のブロックのいくつかの命令をフェッチすることができる。

例えば、

```
if ( aa == 2 ) aa = 1;
if ( bb == 2 ) bb = 0;
```

といった、２つの無関係な if 文の処理を考える。r1 を aa、r2 を bb とし、aa,bb ともに２の場合、フェッチ分岐方式では、図6のようになる。

この例では、先頭の *fnez r3,L1,1,2* 命令が、命令選択分岐である。この命令は分岐条件に関わらず、次に L1 からフェッチを開始する。条件によって変わるのは、ブロックを構成する命令の個数である。したがって、実行では、５サイクル目で L1 の *fnez* 命令を先頭のフェッチ命令の分岐処理と同時にフェッチでき、分岐条件に無関係な処理を進めることができる。

4. ま と め

本報告では、パイプラインプロセッサにおける分岐ハザードを削減する一つのアプローチとして、分岐条件の決定の直後に分岐処理を開始することを可能とした、フェッチ分岐方式を提案した。２つのプログラム例をあげることにより、この方式の効果の可能性を示した。

基本ブロックによるフェッチ指定と命令キューへのプリフェッチの効果により、これまでにないさまざまなプログラムの最適化技法が考えられる。現在、実際の高級言語の手によるコンパイルを通して、これらの調査を行なっている。

参 考 文 献

1) Patterson, D. A. and Hennessy, J. L. , 成田光彰 訳:「コンピュータの構成と設計」, 日経BP社 (1996).

2) Wilkinson, B. , 高橋義造 監訳, 渡辺尚, 小林真也 訳:「計算機設計技法 マルチプロセッサシステム論」, プレンティスホール／トッパン (1994).

3) Hennessy, J. L. and Patterson, D. A. : "Computer Architecture – A Quantitative Approach", Morgan Kaufmann(1990).

4) Johnson,M.: "Superscalar Microprocessor Design" , Prentice Hall(1991).

5) Ditzel, D. R. and McLellan, H. R.: "Branch Folding in the CRISP Microprocessor: Reducing Branch Delay to Zero", Proceeding of the 14 th Annual International Symposium on Computer Architecture, ACM, pp.2–9(1987).

6) Intrater, G. D. and Spillinger, I. Y. : "Performance Evaluation of a Decoded Instruction Cache for Variable Instruction Length Computers", IEEE Transaction on Computers, Vol.43, No.10, pp.1140–1150(1994).

7) Perleberg, C. H. and Smith, A. J. : "Branch Target Buffer Design and Optimization", IEEE Transaction on Computers, Vol.42, No.4, pp.396–412(1993).