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

岡 本 秀 輔† 曽 和 将 容†

命令フェッチをプログラム制御するための機構であるフェッチ分岐方式について述べる．この方式は，命令フェッチの動作を指定する命令の列をプログラム中に作り，実行時に演算命令などの命令と並列に処理を進めていく．基本的な性能評価の結果では，多くの分岐処理を他の命令実行とは並列に行えることが示され，1サイクルにフェッチする命令数を増やすことで，その性能を直接的に高められることが分かった．

Processor Architecture with Program Controlled Instruction Fetching

SHUSUKE OKAMOTO† and MASAHIRO SOWA†

This paper describes a program controlled instruction fetching mechanism, which is called the Fetch Instruction Method. In this mechanism, the special instructions to control instruction fetching are inserted into programs. They are processed in parallel with the other types of instructions. The results of performance evaluations show that many of branches are processed in parallel, and the performance is more improved by only increasing the number of fetched instructions per clock cycle.

1. はじめに

命令レベルの並列性の利用と動作周波数の高速化により，プロセッサが単位時間あたりに処理する命令の数は増加の一途をたどっている．その処理能力を有効に活用するためには，十分な数の命令を絶えず実行ユニットに供給する必要がある．しかし，キャッシュを含めたメモリ・システムのアクセス速度は，プロセッサの処理速度に比べて相対的に低速である．そのため安定的に多くの命令を供給するには，命令キューなどを用いて，命令供給の速度と実行の処理速度の差を吸収し，さらに，命令実行の進度の影響を命令供給側で受けないようにする必要がある．つまり，命令のフェッチと命令の実行の双方で，独立性が高くなるようにそれぞれの処理をする必要がある．

一般に，プログラム上で命令フェッチの動作が明示的に指定されることはない．そのため現状において命令フェッチの動作は，フェッチに影響を与える分岐命令の情報に基づいて行われている．ところが，現代のプロセッサで使用される分岐命令は，命令フェッチと実行を逐次的に行う初期のプロセッサのそれと原理的に変わっていない．分岐命令は，“次に実行すべき命令を”，デフォルトとして決められた隣接命令から他の命令に変更する目的で指定されるので，命令フェッチ動作用の情報としては十分ではなく，以下のような問題が生じる．

- 分岐先アドレスの情報が分岐処理の必要となる場所に置かれるため，分岐命令が実行されるまで次の分岐先アドレスは分からない．そのため実行が分岐命令に到達するまで，分岐先の命令フェッチを始められない．
- 1つの分岐命令からは，分岐後に命令フェッチをどこまで連続して行うべきかという情報が得られない．この情報を得ることなくプロセッサが実行と並列に命令フェッチを行うと，誤った命令をフェッチすることになる．その場合には当該命令を破棄し，新たに命令をフェッチしなければならない．

現代の多くのプロセッサではプログラムを変更せずに，Branch Target Buffer（BTB）またはそれを拡張したハードウェアにより，これらの問題に対処している[1),3),8)]．BTBは一種のキャッシュであり，過去に実行した分岐命令の情報を記憶することで，フェッチした時点での，1) 分岐命令の発見，2) 分岐先アドレスの提供，3) 分岐予測情報の提供という3つの主な役割を果たす．1) により，連続してフェッチする命令の境界を知ることができ，2) により，分岐先アドレスの値を分岐命令の実行前に知ることができる．3) に

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

対しては，分岐予測の方法により，条件分岐の履歴のみをBTBから独立させた別ユニットで行うことがあるので[2),9)]，1) および2) がこの機構の中心的役割となってきている．これらの中心的役割の意味するところは，すなわち，プログラム中に指定された命令実行のための情報と，過去の分岐命令の実行状況から，命令フェッチのために必要な情報をハードウェアで作り出していると考えることができる．

本論文では，このハードウェアのみの方法とは異なり，命令フェッチと命令実行を並列に処理するプロセッサに対して，命令フェッチをプログラム制御する方式について述べる．プログラム制御された命令フェッチ処理では，その一部として分岐処理を行うので，この方式をフェッチ分岐方式と呼ぶ[10)~13)]．

はじめにフェッチ分岐方式の原理および命令の詳細を説明し，命令パイプライン実行するプロセッサとして，このアーキテクチャを実現した場合の構成および基本性能について述べる．最後に，研究または実用化されている既存の技術における本方式の位置づけについて述べる．

2. フェッチ分岐方式

フェッチ分岐方式はプログラムによって命令フェッチを制御する方式である．この方式では命令フェッチの動作を制御するために，フェッチ命令と呼ぶ新しい種類の命令を導入する．フェッチ命令は，それだけで演算やロード/ストアなどの命令の列とは明確に区別できる命令列を構成する．1つのフェッチ命令には，いくつかの命令を束ねたブロックが処理すべき対象として割り当てられる．その情報として，次にフェッチすべきブロックの先頭アドレスと，そのブロックを構成する命令数の組が指定される．それらに加えて，ブロックが2つ指定された場合の選択方法である分岐処理の情報が指定される．したがって，次にフェッチすべきブロックの決定と，決定されたブロック内の命令をフェッチして実行側に渡すことが，1つのフェッチ命令の動作範囲となる．これによりプログラムの流れの制御はフェッチ命令の指定で行うこととなる．

命令をフェッチするための命令は，それ自身も処理前にフェッチされなければならない．そこでフェッチ命令をブロックの先頭に必ず1つ配置し，1つ前のフェッチ命令がブロックの先頭でそのフェッチを行う．これによりフェッチ命令に指定されるアドレスは，ブロックの先頭アドレスであるとともに，次のフェッチ命令が置かれるアドレスとなる．結果として，演算やロード/ストアといった実行のための命令は，ブロックごとに連続して配置されて列を成す．一方でフェッチ命令は空間的に不連続に配置されるが，次のフェッチ命令の置かれたアドレスによって列を成す．

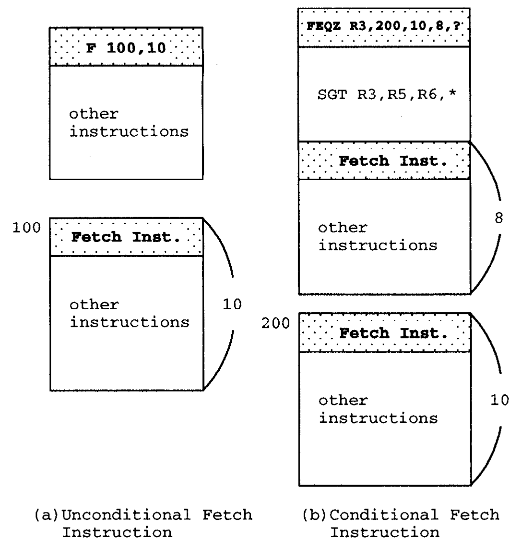


図1 基本フェッチ命令
Fig. 1 Basic fetch instructions.

1つのフェッチ命令は1つの分岐処理を含むことから，フェッチ命令の種類は，従来の分岐命令の種類と対応して，無条件フェッチ命令，条件フェッチ命令，手続き呼び出し用フェッチ命令，手続きからのリターン用フェッチ命令が主要なフェッチ命令となる．以下では各フェッチ命令の指定内容とその動作を述べる．

2.1 無条件フェッチ命令

無条件フェッチ命令は，無条件にブロック内の命令をフェッチして，実行側に渡すことを指定するための基本命令である．図1において，網かけ部分がフェッチ命令であり，それと続く四角の部分の命令とで1つのブロックを表す．図1(a) の上側のブロックの無条件フェッチ命令 “*F 100, 10*” は,「100番地から10命令で構成されるブロックの命令をフェッチして，実行側に渡す」という意味である．1つ前のフェッチ命令により，上側のブロックの命令がすべてフェッチされ，それが終わった時点でこの無条件フェッチ命令が開始される．フェッチ命令はブロックの先頭でフェッチされるため，1つ前のフェッチ命令の処理が終了したあとの最適なタイミングで，その処理を開始できる．

2.2 条件フェッチ命令

条件フェッチ命令は，2つのブロックから1つを選択してブロック内の命令をフェッチし，実行側に渡すことを指定する命令である．図1(b) は条件フェッチ命令の例である．この条件フェッチ命令 “*FEQZ R3,*

200, 10, 8,?” は,「分岐条件 R3 を設定する命令の処理と同期をとり，もし R3 がゼロならば 200 番地からの 10 命令をフェッチし，非ゼロならば隣接ブロックの 8 命令をフェッチして，実行側に渡す.」を意味する. “?” のオペランドが同期の指定である.

同図の比較命令 “*SGT R3, R5, R6,**” が，条件となる R3 を決定する条件設定命令となる．この命令は条件フェッチ命令の後にフェッチされて実行を開始し，実行終了時に合図を命令フェッチ側に送って同期をとる．条件設定の完了を保証するだけなので，同期によって条件設定命令が待つことはない．

2.3 手続き呼び出し/リターン用フェッチ命令

ロード/ストア方式を前提とした場合，手続き呼び出し用フェッチ命令は，手続きの先頭ブロックのフェッチを行うことに加えて，戻り先のブロックの情報をレジスタに待避する．リターン用のフェッチ命令は，待避された情報を用いて戻り先のブロックをフェッチする．しかし，待避すべきブロックの情報は，先頭アドレスと命令個数の 2 つになり，従来の分岐方式よりも扱うべきデータが増えてしまう．

この問題を回避する方法として，戻り先ブロックの命令数に制限を加える方法が考えられている．手続き呼び出し用フェッチ命令では，手続きの先頭ブロックのアドレスとその命令数のみを指定し，その処理では，隣接ブロックの先頭アドレスを予約されたレジスタに保存する．リターン用のフェッチ命令は，先頭アドレスが保存されたレジスタと，戻り先ブロックの命令数を指定する．この方法でレジスタに保存される情報は，戻り先の先頭アドレス 1 つになる．しかし，戻り先ブロックの命令数が固定となるので，命令数を調節するために，ブロックを分割するか，NOP 命令を挿入する必要が生じる．

どちらの手法が効率良く動作するかはプログラムによって異なる．以降では，仮に後者の方法を採用して議論を進めていく．また，レジスタに対して実行命令とのハザードが生じる場合には，条件フェッチ命令と同様に同期指定を行い，これを回避する．

リターン用のフェッチ命令は従来のレジスタ指定の分岐命令に相当し，複数方向の分岐に利用できる．また引数として渡された手続きの呼び出しには，リターン用の命令に隣接ブロックのアドレスを待避する機能を付加した命令を用いることができる．

3. 命令パイプラインの構成例

フェッチ分岐方式を，命令パイプライン動作のプロセッサに対して適用した場合の構成例について述べる．

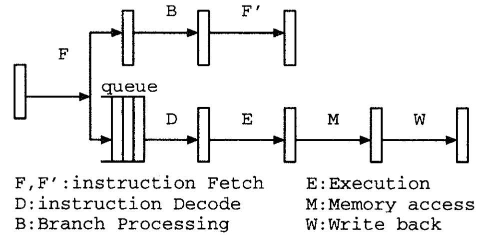


図 2 パイプライン構成
Fig. 2 A pipeline structure.

この構成は後の基本性能の評価で用いる．

命令パイプラインの構成を図 2 に示す．2 つの矩形部分を結ぶ矢印がパイプラインの各ステージを示している．全体として，フェッチ命令用の上段とその他の実行命令用の下段からなり，それらが命令フェッチ (F) ステージを共有した構成となっている．すべてのフェッチ命令は，ブロックの先頭に配置されているので，プロセッサはフェッチ命令をデコードすることなく，他の実行命令と区別できる．そのため (F) ステージにおいて，フェッチ命令は上段，その他の命令は下段に振り分けられる．

フェッチ命令用の上段にある分岐 (B) ステージでは，命令のデコード，アドレス計算，同期処理，先行するフェッチ命令終了のチェック，条件フェッチ命令のための処理が行われる．同期処理による待ち，または先行するフェッチ命令が終了していないことによる待ちが必要な場合には，このステージを繰り返す．続く，命令フェッチ (F') ステージでは，指定個数の命令をフェッチし，命令キューへ供給する．このステージの処理が，結果として先頭の (F) ステージに反映される．

実行命令用の下段では，指定された命令の種類によって，同期のための合図が最適なステージで送られる．たとえば，レジスタに結果を書き込む命令とレジスタを読むだけの命令では，合図のタイミングが異なり，レジスタ・フォワーディングなどの機構の有無によって，それぞれの最適なステージが決まる．

4. 処理効率に関する実現手法

フェッチ命令はそれらのみで列を構成し，図 2 の命令パイプラインの上段を通る．そのため，複数の命令を同時にフェッチすることで，フェッチ命令と他の実行命令とが並列に処理される可能性を高くする．また，待ちを必要とする条件フェッチ命令では，分岐処理の遅れが下段のパイプラインの停止につながる．そこで，この待ちの間に，選択すべき双方のブロックの命令を専用バッファにプリフェッチしておくことで，命

令フェッチの遅れを短縮できる．

このように，フェッチ命令の処理が全体の処理効率に影響を与える部分には，1 クロック・サイクルあたりにフェッチする命令数と分岐処理の 2 つがある．また，プリフェッチする命令にはつねにフェッチ命令が含まれるので，両手法ともフェッチ命令自身のフェッチと他の処理がオーバラップする頻度を高める効果を持つ．

5. 実行トレースの例

前述のパイプライン構成における 2 つの実行トレースの例を示す．フェッチ命令以外の命令は，ロード/ストア方式を採用する RISC 型のプロセッサ DLX [5] に基づく．以下では，1 クロック・サイクルに 2 命令をフェッチし，条件フェッチ命令の同期待ち時に，隣接ブロックと分岐先ブロック双方の先頭の 2 命令を，順にプリフェッチすることを仮定する．また，各パイプライン・ステージにおいて，レジスタ・フォワーディング機構などにより排除できるデータ・ハザードはすべて生じないものとする．

5.1 プログラム例 1

図 3 (a) は，“start”，“fall”，“target”，“end” というラベルがつけられた 4 つのブロックからなるプログラムである．“start” ブロック中の条件フェッチ命令が，“fall” または “target” のブロックを選択し，そのどちらのブロックも，次に “end” ブロックへと制御が移る．条件フェッチ命令である FEQZ 命令は，同期指定により，対応する sgt 命令の終了を待つ．“end” ブロックにある F 命令のオペランドを 0 にすることで，このプログラムはそれ以上の命令の実行を停止する．

同図 (b) は，(a) のプログラムを実行したときのトレース結果である．clk の欄は実行クロック，FC の欄はフェッチした命令アドレスの値，n の欄はフェッチすべき残りの命令の個数，q の欄は命令キューにある命令の個数である．FC，n，q はそれぞれそのクロックにおける処理前の状態を示している．F の欄はそのクロックでのフェッチの状態を示し，フェッチ命令を含んでフェッチしたときに “F”，実行命令をフェッチしたときには “*” となる．さらに，プリフェッチの状態として，分岐命令に続く命令のプリフェッチをした場合には “f”，分岐先命令をプリフェッチした場合には “t” となる．これは図 2 のパイプライン上段の F' ステージの内容でもある．B の欄はフェッチ命令の分岐ステージを示している．残りの欄は図 2 の下段のパイプラインの状態を示す．D の欄はフェッチ命令以外の命令のデコードを示す．E と M の欄では，処理さ

```
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       end,0
        add     r1,r0,r0

-------------------- (a)
```

clk	FC	n	q	F	B	D	EMW
1	0	5	0	F			
2	8	3	0	*		ADDI	
3	16	1	1	*		ADDI	a
4	20	0	1	f		ADDI	aa
5	20	0	0	t		SGT	aa1
6	20	0	0				aa2
7	20	0	0		FEQZ		a5
8	40	3	0	*		SLT	3
9	48	1	1	*	F	SUBI	a
10	52	2	1	F		AND	aa
11	60	0	1		F	SLE	aa6
12	52	0	0			ADD	aa3
13	52	0	0				aa4
14	52	0	0				a3
15	52	0	0				1

-------------------- (b)

図 3 プログラム例 1 とそのトレース
Fig. 3 Program example1 and its trace.

れた命令が算術・論理演算命令の場合には “a”，ロード/ストア命令の場合には “t” となる．W の欄は書き込むレジスタ番号を 16 進数で示す．

1 クロック目において，0 番地から残り 5 つの命令をフェッチすることが示され，命令キューが空という状態から，F 命令と addi 命令がフェッチされる．3 クロック目までに残りの 3 つの命令がフェッチされ，“start” ブロックのすべての命令がフェッチされる．分岐条件が決定していないために，4 クロック目で “fall” ブロックの 2 命令，5 クロック目で “target” ブロックの 2 命令がそれぞれプリフェッチされる．分岐条件を決める sgt 命令は，6 クロック目で計算を行い，次の 7 クロック目で条件フェッチ命令がその結果を利用する．“target” ブロックの先頭命令はすでにプリフェッチされているので，8 クロック目からデコードが開始される．また，“target” ブロックの無条件フェッチ命令は，9 クロック目で分岐ステージを終えており，他の実行命令とは並列に処理されている．

```
start:  FAL    func,6      ;; Call to ''_func''
        lhi    r14, ((memSize-4)>>16)&0xffff
        addui  r14, r14, ((memSize-4)&0xffff)
        sub    r14,r14,#4
        addi   r3,r0,#10
        sw     0(r14),r3

return: F      exit,1
        add    r14,r14,#4

func:   FR     r31,2 ;; Return
        sw     -4(r14),r2
        lw     r2,0(r14)
        addi   r1,r0,#3
        add    r1,r1,r2
        lw     r2,-4(r14)

exit:   F      exit,0
-------------------- (a)

   clk| FC  | n| q|F| B        D    |EMW|
   ---+-----+--+--+-+------+-----+---+
     1|    0| 6| 0|F|      |     |   |
     2|    8| 4| 0|*|      |  LHI|   |
     3|   16| 2| 1|*|FAL   |ADDUI|a  |
     4|   32| 6| 2|F|      | SUBI|aa |
     5|   40| 4| 2|*|      | ADDI|aaE|
     6|   48| 2| 3|*|FR    |   SW|aaE|
     7|   24| 2| 4|F|      |   SW|taE|
     8|   32| 0| 4| |F     |   LW|tt3|
     9|   56| 1| 3|F|      | ADDI|tt |
    10|   60| 0| 2| |F     |  ADD|at |
    11|   56| 0| 1| |      |   LW|aa2|
    12|   56| 0| 0| |      | ADDI|ta1|
    13|   56| 0| 0| |      |     |at1|
    14|   56| 0| 0| |      |     | a2|
    15|   56| 0| 0| |      |     |  E|
    16|   56| 0| 0| |      |     |   |
  --------------------- (b)
```

図4 プログラム例2とそのトレース
Fig. 4 Program example2 and its trace.

5.2 プログラム例2

図4(a)は手続き呼び出しを行うプログラムである．"start"ブロックのFAL命令は，戻り先である"return"ブロックの先頭番地をr31にセットした後に，"func"ブロックの命令をフェッチする．"func"ブロックでは，FR命令によりr31の持つ値をアドレスとしてリターンをする．

同図(b)は実行トレースである．FAL命令，FR命令は他の実行命令と完全にオーバラップして処理されているので，手続き呼び出しにより3つのブロックを制御が移動しているにもかかわらず，実行命令のデコード・ステージは1つのブロックのように処理している．

6. 基本性能の評価

これまでに述べたプロセッサに対する基本性能の評価について述べる．特に，フェッチ命令自身のフェッチおよびその実行が，前述の1クロックあたりの命令数および分岐先命令のプリフェッチによってどのように変化するかについて評価する．

6.1 評価方法

性能評価は命令パイプラインを備えたプロセッサの挙動をクロック・レベルで扱うソフトウエア・シミュレータを用いて行った．キャッシュによる影響はないものとする．評価対象のプロセッサは，フェッチ分岐方式を用いる以下の4種類のプロセッサとした．

- f1：1クロック・サイクルに1命令フェッチ
- f1p：f1に加えて分岐処理待ち時にプリフェッチ
- f2：1クロック・サイクルに2命令フェッチ
- f2p：f2に加えて分岐処理待ち時にプリフェッチ

プリフェッチを行うプロセッサは，先に示したパイプライン構成に，プリフェッチ用のバッファがついた構成である．また，命令キューはどれも8に設定した．

評価尺度として3種類のDLXプロセッサを用いた．

- db：遅延分岐を行う．
- pnt：静的分岐予測として分岐不成立予測を行う．
- pss：データ・ハザードのない分岐命令を他の種類の命令と並列に実行し，分岐後の命令フェッチのペナルティを理想的にゼロとする．

dbは分岐命令の次に分岐方向に関係しない1命令を実行する．pntは分岐命令の次に隣接する命令をフェッチして，分岐しない場合にはそれを実行し，分岐した場合には破棄する．pssは本評価における仮想的なプロセッサである．これら3つのプロセッサのパイプライン構成は図2の下段の構成と同一であり，すべてデコード・ステージまでで分岐命令を処理することとする．これらを評価尺度とすることで，本方式によるプロセッサの一般的な性能を知るとともに，命令フェッチを明示的に記述することの効果を明らかにする．

ベンチマーク・プログラムは以下の3つを用いた．

- btree：メモリ上のB木に対して挿入，削除，探索を行う．条件分岐が比較的多い．
- eval：再帰下降型構文解析法を用いて式評価を行う．条件分岐と手続き呼び出しの数がほぼ同数．
- tayl3：3次のテイラー展開を用いた常微分方程式の求解．条件分岐が少なく手続き呼び出しが多い．

これらのプログラムをC言語により記述し，DLXプロセッサの1つであるdb用のGNU Cコンパイラを用い，最適化オプションをつけてコンパイルした．さらに出力されたアセンブリ・プログラムを手で最適化し，結果をトランスレータでそれぞれのプロセッサ用に変換した．フェッチ分岐方式のプログラムへの変換は基本ブロック単位で行った．この変換で，変換前のプログラムの遅延スロットにある NOP 命令は削除されるが，前述の手続きからの戻りブロックの制限な

表1 プログラムごとの命令数
Table 1 Number of instructions.

	btree	eval	tayl3
db	1180	535	408
pnt, pss	1050	465	349
f1, f1p, f2, f2p	1101	510	360

どから，無条件フェッチ命令が増える場合がある．

6.2 プログラム・サイズ

表1 に，これらのプログラム・サイズをアセンブリ命令の数で示す．pnt と pss は同一のコードを実行し，フェッチ分岐方式を行うプロセッサは他の同一コードを実行する．この表から，これらのベンチマークにおいて，フェッチ分岐方式用のプログラムは，遅延分岐用のプログラムよりも命令数が少ないことが分かる．

6.3 実行時間

図5 にベンチマーク・プログラムの実行時間をクロック数で示す．3つのプログラムに共通して f2p と pss が他よりもクロック数が少ない．この2つは tayl3 において同一クロック数で実行し，btree と eval では約1%ほど pss の方が少ないクロック数で実行している．

また，btree と eval における f1p と f2 が f1 よりもクロック数を削減していることから，プリフェッチおよび2命令フェッチがどちらも効果があると考えられる．手続き呼び出しが大半を占める tayl3 では，プリフェッチがほとんど行われないために，f1 と f1p，f2 と f2p がそれぞれほぼ同数のクロック数である．

f1 による btree と eval の実行において，クロック数が他より多くなっている理由としては次のことが考えられる．f1 では条件分岐の処理を行った次のクロックにおいて，続くブロックのフェッチ命令のみをフェッチし，その次のクロックで演算などの実行命令をフェッチし，さらに次のクロックでその実行命令のデコードを開始する．この分岐後のフェッチの遅延が，結果として，実行時間を増加させている．

6.4 命令別の並列性

図6～8 のグラフは，フェッチ命令および分岐命令の種類ごとに，分岐処理が行われたタイミングおよびその頻度示す．各グラフは命令別にグループ分けされている．F(J) は無条件フェッチ（分岐）命令，FAL(JAL) は手続き呼び出し用のフェッチ（分岐）命令，FR(JR) は手続きからのリターン用のフェッチ（分岐）命令，FNEQ(BNEQ) は条件が非ゼロのときに分岐する条件フェッチ（分岐）命令，FEQZ(BEQZ) は条件がゼロのときに分岐する条件フェッチ（分岐）命令を，それぞれ示す．グラフの縦軸は各命令の処理された回数

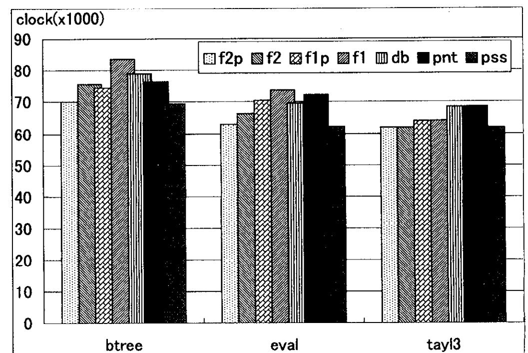


図5 ベンチマークプログラムの実行時間
Fig. 5 Execution time for benchmark programs.

であり，プロセッサの種類ごとに，その命令の処理されたタイミングが示されている．Zero はその命令が，全体の実行時間に対して，見かけ上ゼロ・クロックで処理されたことを示し，One はその命令が実行時間のなかで1クロックだけかかったことを示す．Over One は処理が1クロックを越えたことを示す．

すべてのグラフに共通して，無条件フェッチ命令の処理回数が同種の分岐命令の処理回数より多い．しかし f2 および f2p のプロセッサでは，どのプログラムでもほとんどの無条件フェッチ命令がゼロ・クロックで処理されており，付加的に挿入された命令による実行時間の増加はない．次にプログラムごとの特徴を見る．

6.4.1 B木ベンチマーク

図6はB木ベンチマーク btree プログラムの実行結果を示している．これは条件分岐の処理回数が手続き呼び出しよりも2倍以上多い．f2p と f2 では，無条件フェッチ命令，手続き呼び出し・リターン用フェッチ命令が，ほとんどがゼロ・クロックで処理されており，f2p と f1p で多くの条件フェッチ命令が1クロックのみで処理されている．つまり，2命令フェッチによる効果とプリフェッチによる効果がそれぞれ現れている．f2p と pss の比較では，手続き呼び出しに違いがある他は，ほぼ同じ処理となっている．

6.4.2 式評価ベンチマーク

図7は式評価を行うベンチマーク eval プログラムの実行結果を示している．これは手続き呼び出しと条件分岐の処理回数がほぼ同数である．f2p と f2 の無条件および手続き関連のフェッチ命令の多くが，ゼロ・クロックで処理されており，条件フェッチ命令でもゼロ・クロックの処理がある．逆に，f1p と f1 ではゼロ・クロックの処理が少なく，FAL 命令の処理では1クロック以上を示す黒い部分が半分以上を占めている．また，f2p と f1p の条件フェッチ命令における差から，

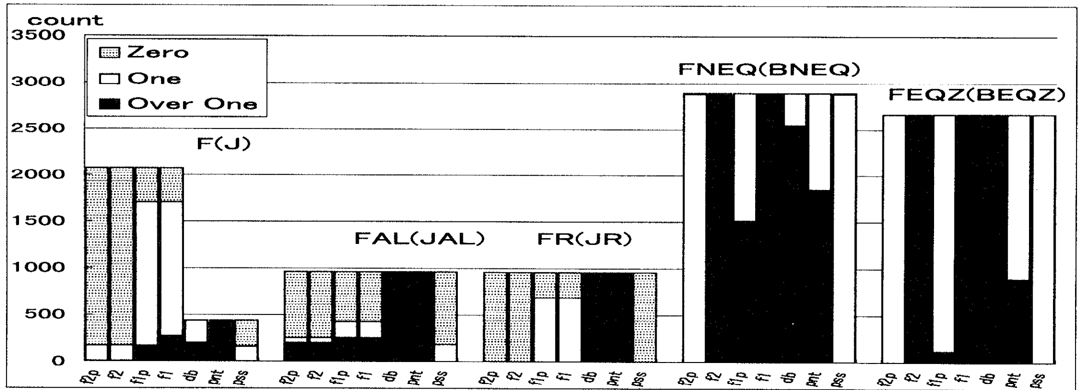


図 6　btree ベンチマークの命令配分
Fig. 6　Instruction mix on btree bechmark.

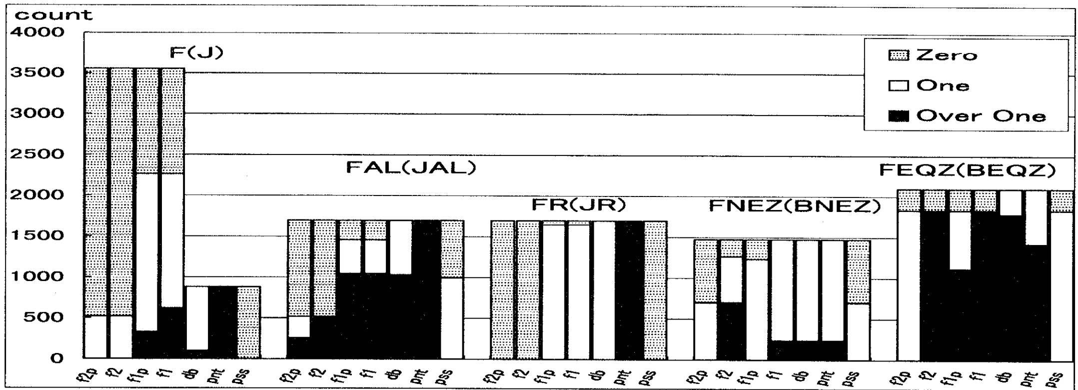


図 7　eval ベンチマークの命令配分
Fig. 7　Instruction mix on eval benchmark.

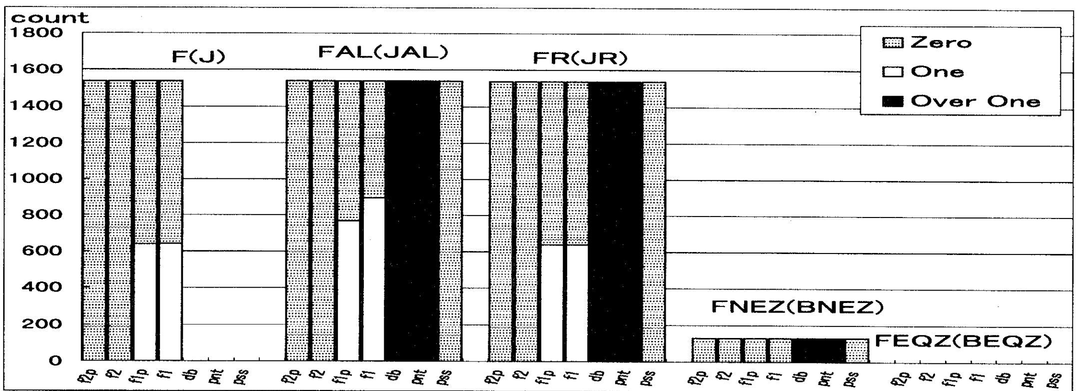


図 8　tayl3 ベンチマークの命令配分
Fig. 8　Instruction mix on tayl3 benchmarks.

2 命令フェッチは条件分岐の処理にも効果があることが分かる．f2p と pss の比較では，無条件フェッチ（分岐）命令で，pss の方がゼロ・クロックの処理が多く，手続き呼び出しのところで，f2p の方が多い．

6.4.3　テイラー・ベンチマーク

図 8 は 3 次のテイラー展開を用いるベンチマーク tayl3 プログラムの実行結果である．これは条件分岐の処理が少なく，手続き呼び出しが多い．f2p，f2 およ

表2 実行クロック数削減率
Table 2 Reduced clock ratio.

	btree			eval			tayl3		
	db	pnt	pss	db	pnt	pss	db	pnt	pss
f2p	11%	8%	−1%	9%	13%	−1%	9%	9%	0%
f2	4%	1%	−9%	5%	8%	−7%	9%	9%	0%
f1p	6%	3%	−7%	−1%	3%	−13%	6%	6%	−3%
f1	−6%	−9%	−21%	−6%	−2%	−19%	6%	6%	−4%

表3 分岐処理のタイミングの割合
Table 3 Timing ratio of branch processing.

	btree			eval			tayl3		
	Zero	One	Over One	Zero	One	Over One	Zero	One	Over One
f2p	38%	60%	2%	66%	31%	3%	100%	0%	0%
f2	37%	3%	60%	61%	10%	29%	100%	0%	0%
f1p	12%	66%	22%	20%	56%	24%	57%	43%	0%
f1	12%	24%	64%	17%	47%	36%	54%	46%	0%

び pss ではすべての分岐処理がゼロ・クロックでなされており，実行時間はすべて演算およびロード/ストア命令の処理時間という理想的な結果となった．一方 db と pnt ではほとんどすべての分岐処理が 1 クロック以上かかっている．また，f1p と f1 も多くのゼロ・クロックの処理がなされているが，f2p や f2 との差から，2 命令フェッチの効果が大きいと考えられる．

6.5 評価結果のまとめ

表2 に評価尺度のプロセッサ（db, pnt, pss）に対する実行クロック数削減率を示す．マイナスの数値は増加を示す．f2p は db，pnt に対して 8～13%の削減をした．pss に対しては −1～0% であり，ほぼ理想に近い分岐処理といえる．f1p と f2 の削減率は db，pnt に対して −1～9%であり，プリフェッチ/2 命令フェッチそれぞれの手法の効果は現れているが，プログラムの種類によってその効果の度合いが異なる．なお，f1 の削減率は −21～6%とマイナス値が多く，図 6～8 とあわせて考えると，条件分岐の処理回数が多いものほど性能が低い．これは条件分岐後のフェッチ命令自身のフェッチが，他の処理とオーバラップしないことによる性能の低下と考えられる．

次に，この 2 手法の具体的な効果の内容を**表3** により示す．これは各プログラムにおける全分岐処理のタイミングの割合であり，図 6～8 のデータのまとめである．ゼロ・クロック処理に着目し，f2p と f1p の差および f2 と f1 の差を見ると，btree では 26%と 25%，他の 2 プログラムでは 43～46%である．このタイミングではプリフェッチの効果も多少含まれるが，2 命令フェッチによる効果が主であるといえる．他方，1 クロック以上かかった処理に着目して，f2p と f2 の差および f1p と f1 の差をそれぞれ見ると，btree で 58%と 42%，eval で 26%と 12%の差である．ここでは 2 命令フェッチよりもプリフェッチの効果が大きい．このように 2 手法はそれぞれ異なった形で効果を持つ．

以上の結果から，フェッチ命令の導入により従来方式のプロセッサ（db, pnt）よりも良い性能が得られることが示された．なお，フェッチ命令を用いずに，プリフェッチや 2 命令フェッチのみで性能向上を行うことも考えられるが，その場合の効果は限定的である．たとえば，pnt にプリフェッチ/2 命令フェッチのみを導入したプロセッサに対する f2p のクロック数削減率を見ると，f2p の方が依然として高い（btree で 2%，eval と tayl3 でともに 5%程度）．また，pnt にプリフェッチ/2 命令フェッチを導入するには，命令フェッチと同時に分岐命令を判別し，分岐先アドレスを得る機構を必要とするので，ハードウェア量の観点からもフェッチ命令を採用した方がよい．

7. おわりに

動的分岐予測を行うために使われる Branch Target Buffer（BTB）は，キャッシュであるために，初めて処理される分岐命令は発見できず，そのサイズによって記憶できる分岐命令の数が制限される．そしてマルチタスク環境では，タスク切替えの際に蓄積した情報はクリアされうる．つまりすべての分岐命令をフェッチした時点で発見できるわけではない．

一方，フェッチ命令に指定されたアドレスは，次のフェッチ命令が置かれるアドレスでもあるために，フェッチした時点で必ずフェッチ命令が見つかる．また，ブロックの先頭命令のフェッチで次のアドレス計算

が開始できるので，1ブロックの命令数が同時にフェッチする命令数よりも多ければ，BTBの方法と同時期または早期に次のアドレスが得られると考えられる．さらに分岐予測に関しては，条件フェッチ命令の結果のみを保存することで，BTBとは分離した従来の予測技術を利用できると考えられる．以上のことから，本方式においてBTBを使わない分岐予測の機構を構築できると考えられる．

評価ではf1で条件分岐後のフェッチ命令自身のフェッチが単独で行われる場合でも，分岐命令は分岐条件設定の実行中にフェッチされるという違いにより性能差が現れた．しかし，スーパスカラ・プロセッサでは複数命令フェッチが前提であり，さらに，分岐予測や本評価のような分岐先命令のプリフェッチによってフェッチ命令がプリフェッチされることから，このマイナス面はなくなると考えられる．また，フェッチ命令のプリフェッチは上記のアドレス計算にさらに時間的な余裕を作る．分岐命令方式で分岐先命令をプリフェッチしても，それが分岐命令である可能性は低いので，これはフェッチ分岐方式の利点となる．

並列処理の可能性に関してもフェッチ分岐方式には利点がある．表3に示したゼロ・クロック処理の割合と図6～8の結果をあわせると，無条件フェッチ命令，手続きに関係するフェッチ命令，条件の決定した条件フェッチ命令は，1クロックにフェッチする命令数を複数にすることで，他の実行命令との並列処理の機会が増える．これはスーパスカラ・プロセッサを構築したときに，分岐命令を考慮しなくてよい分，命令発行の複雑さを減らすことにつながると考えられる．

この特徴はコンパイラが行う命令移動の最適化にも関連する．分岐命令方式は，分岐命令が基本ブロックの最後の命令であるため，先行する命令のストールによりデコードが中断された場合には，その分だけ処理開始が遅らされる可能性がある．これはスーパスカラ・プロセッサにおいても少なからず同様の影響がある．本方式ではデコード中断後も，命令キューが満杯にならない限りフェッチ命令の処理は続けられる．したがって，分岐条件を設定する命令の移動は，直接的にフェッチ命令の分岐処理を早期にする．従来方式では条件を設定する命令の後の命令のストール状況に，分岐命令の実行開始が左右される可能性がある．

その他に，フェッチ命令のみでプログラムの制御の流れをたどれるという特徴は，動的分岐予測の機構において，複数以上の分岐をたどることに利用できる可能性がある．また trace cache[2]のように，実行のトレースを命令キャッシュ上で検出する場合にも，同様にこの特徴の利用が考えられる．

他方でマイナス面もあり検討を必要とする．フェッチ命令はブロックの命令数を指定する．これは命令フィールドの数ビットを必要とし，結果として分岐先指定のビット数が減るためにコード配置で制限が生じる．同じく，命令数の指定に上限があるので，ループ・アンローリングなどの，基本ブロックの命令数を増やすような最適化が制限されうる．また，過去の資産を継承するためのバイナリ・コードの自動変換についても考慮を要する．コード変換は，基本ブロック単位の命令数を得てから，分岐命令をフェッチ命令に置き換え，分岐命令のない基本ブロックには無条件フェッチ命令を挿入すればよい．しかし，動的にリンクされるライブラリに対しては，それを利用するすべてのプログラムと，命令数に関して対応をとる必要があり，この処理が複雑になりうる．

参 考 文 献

1) Conte, T.M., Menezes, K.N., Mills, P.M. and Patel, B.A.: Optimization of Instruction Fetch Mechanisms for High Issue Rates, *Proc. 22nd Annual International Symposium on COMPUTER ARCHITECTURE* (1995).

2) Rotenberg, E., Bennett, S. and Smith, J.E.: Trace Cache: A Low Latency Approach to High Bandwidth Instruction Fetching, *Proc. 29th Annual IEEE/ACM International Symposium on MICROARCHITECTURE* (1996).

3) Uht, A.K., Sindagi, V. and Somanathan, S.: Branch Effect Reduction Techniques, *IEEE Computer*, Vol.30, No.5, pp.71–81 (1997).

4) Milligan, M.K. and Cragon, H.G.: Processor Implementations Using Queues, *IEEE Micro*, Vol.15, No.4, pp.58–66 (1995).

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

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

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

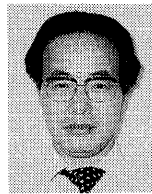
8) Perleberg, C.H. and Smith, A.J.: Branch Target Buffer Design and Optimization, *IEEE Trans. Compute.*, Vol.42, No.4, pp.396–412 (1993).

9) Advance Information PowerPC 620 RISC Microprocessor Technical Summary, MPR620TSU-01: IBM Microelectronics, MPC620/D: MOTOROLA (1994).
10) 岡本秀輔，曽和将容：フェッチ分岐方式の提案，情報処理学会研究報告，Vol.96, No.80, pp.79–82 (1996).
11) 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，情報処理学会研究報告，Vol.97, No.22, pp.49–54 (1997).
12) Okamoto, S. and Sowa, M.: Instruction Fetch Mechanism for PN-Superscalar, *Proc. International Conference on Parallel and Distributed Processing Techniques and Applications*, (*PDPTA'97*), Vol.III, pp.1406–1410, (1997).
13) 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).

(平成 9 年11月4日受付)
(平成10年 6 月5日採録)

岡本　秀輔（正会員）

昭和40年生．平成6年成蹊大学大学院工学研究科情報処理専攻博士後期課程修了．同年電気通信大学大学院情報システム学研究科助手．現在，同研究科講師．博士（工学）．並列分散処理プログラミング，命令レベル並列処理の研究・教育に従事．電子情報通信学会，日本ソフトウェア科学会，IEEE Computer Society 各会員．



曽和　将容（正会員）

昭和49年名古屋大学大学院博士課程（電気電子専攻）修了．同年群馬大学工学部情報工学科助手．昭和51年助教授．昭和62年名古屋工業大学教授．平成5年電気通信大学大学院情報システム学研究科教授．この間，並列処理，計算機アーキテクチャ，特にデータフロー計算機，コントロールフロー計算機など次世代コンピュータの研究に従事．工学博士．IEEE, ACM 各会員．