マイクロコンピュータ 44－1
（１９８７．３．１３）

パイプライン処理とブランチ命令

吉田 豊彦　松尾 雅仁　上田 達也　清水 徹

三菱電機（株）ＬＳＩ研究所

我々はオリジナル32ビットマイクロプロセッサの開発にあたり新しいマイクロプロセッサのアーキテクチャに適する各種のパイプライン方式を検討した。

パイプライン処理は汎用計算機の歴史の中で高速化技術として最も成功したものの１つである。しかし、パイプライン処理も処理段数が増大するにつれ各種のオーバーヘッドのため処理速度の向上に飽和傾向が現れる。パイプライン処理のオーバーヘッドのなかで最も問題となるのはブランチ命令実行によるパイプラインの乱れである。我々はパイプライン処理におけるブランチ命令のオーバーヘッドをなるべく少なくするため、ブランチ命令の履歴に従ってブランチするかどうかを判断する動的ブランチ予測処理を採用した。

本報告ではパイプライン処理方式の例として８種類のパイプラインモデルを考え、「エラトステネスのふるい」のベンチマークプログラムに対して各種モデルにおける動的ブランチ予測処理の効果をシミュレーションにより検討した結果について報告する。

本報告のシミュレーションではパイプライン段数が４段以上の場合に動的ブランチ処理により10%前後の性能向上が見られた。

BRANCH PREDICTION IN A PIPELINED MICROPROCESSOR

Toyohiko YOSHIDA, Masahito MATSUO, Tatsuya UEDA and Toru SHIMIZU

MISTUBISHI ELECTRIC Corporation LSI Research and Development Laboratory
4-1, Mizuhara, Itami-city, Hyougo-prefecture, 664, JAPAN

Pipelining is one of the most efficient techniques to reach higher performance. A fundamental disadvantage of pipelining is the performance degradation from branches in the instruction stream.

The architecture of our microprocessor is newly developed and it is different from old pipelined computers. So, we designed several types of pipelining models for our microprocessor and examined their performance. We designed a branch prediction mechanism based on branch history to overcome this problem.

Improvements of 5 to 14 percent can be expected in our microprocessor performance when we install this branch prediction mechanism.

1．はじめに

電子計算機の中央処理装置(CPU)は約40年の歴史の中で論理回路自身の高速化と処理方式の並列化により処理能力を大幅に向上させてきた。種々の並列処理方式の中で汎用計算機において最も効果があったものはパイプライン処理である。マイクロプロセッサも1971年に登場して以来、LSIの微細加工技術の進歩に支えられ、動作周波数の高速化と処理ビット幅の拡大により処理能力を急速に向上させてきた。近年のマイクロプロセッサでは１つのLSI内での使用可能ハードウエア量の増大にともない、処理ビット幅の拡大だけではなく汎用計算機を手本にパイプライン処理も取り入れられるようになった。

パイプライン処理は汎用計算機の高速化に有効な技術であるが、パイプライン処理段数が増大するにつれ各種のオーバーヘッドのため処理速度の向上に飽和傾向が現れる。パイプライン処理のオーバーヘッドのなかで最も問題となるのはブランチ命令実行によるパイプラインの乱れである。過去の汎用計算機ではこのブランチ命令によるパイプラインの乱れを少なくするため多くの研究が行われてきた[1]。

我々はオリジナル32ビットマイクロプロセッサ（本報告では以下「ATOM」と呼ぶ）の開発にあたり各種のパイプライン方式とブランチ命令の処理方法について検討した。ATOMでは動的ブランチ予測処理を行うことによりブランチ命令実行によるパイプライン処理の乱れを少なくする見通しを得た。

instruction	高級言語		
	COBOL	Fortran	Pascal
Branch	24.2%	18.0%	18.4%
Logical Operation	14.6%	8.1%	9.9%
Load/Store	40.2%	48.7%	54.0%
Storage-Storage Move	12.4%	2.1%	3.8%
Integer Math	6.4%	11.0%	7.0%
Floating-Point Math	0.0%	11.9%	6.8%
Decimal Math	1.6%	0.0%	0.0%
Other	0.6%	0.2%	0.1%

表2.1　IBM 370 におけるブランチ命令の実行頻度

ブランチ命令の実行頻度は約20%である。

	エラトステネスのふるい		Dhrystone	
	初期値40H	初期値10H	1回目	2回目以降
BRA	33	12	10	10
Bcc(taken)	234	50	9	9
Bcc(not taken)	66	24	18	18
JMP	0	0	1	1
BSR	0	0	15	15
EXIT	0	0	15	15
ブランチ命令計	323 (22%)	86 (22%)	68 (16%)	68 (16%)
全実行命令数	1441	389	420	420

表2.2　ベンチマークにおけるブランチ命令の実行頻度

両ベンチマークプログラムともブランチ命令の実行頻度は実際のアプリケーションを代表しているといえる。

本報告ではATOMのパイプライン処理方式の例として８種類のパイプラインモデルを考え、各種モデルにおける動的ブランチ予測処理の効果をシミュレーションにより検討した結果について報告する。

2．ブランチ命令の実行頻度

命令の実行頻度に関する研究はすでに各種のアーキテクチャの計算機についてさかんに行われてきた。表2.1にHP Precisionアーキテクチャ決定の際調査されたIBM 370の命令実行頻度を示す[2]。もとになる高級言語により多少の差はあるがブランチ命令の実行頻度は約20%あることが示されている。VAX-11やIntel8086シリーズについてもブランチ命令の実行頻度は20%前後と報告されている[3][4]。

本報告で用いたATOMに対する「エラトステネスのふるい」[5]のベンチマークプログラムにおけるブランチ命令の実行頻度を表2.2に示す。過去の統計にしたがって高級言語における実行文のタイプやデータの配置を決定したベンチマークプログラムである

「Dhrystone」ベンチマーク[6]でのブランチ命令の実行頻度も参考のため表2.2に示す。ブランチ命令の実行頻度は「エラトステネスのふるい」では22%、「Dhrystone」では16%であり、両ベンチマークプログラムともブランチ命令の実行頻度について実際のアプリケーションを代表していると言える。「エラトステネスのふるい」は初期値によってプログラム中のループの実行回数が異なり、実行命令の頻度も若干の差があるが、「Dhrystone」は同じプログラムを繰り返し実行するだけであるため実行回数が変わっても実行命令の頻度は同じである。ただし、本報告で行った動的ブランチ予測のヒット率はプログラムの履歴に依存するため後に表4.1で示すように1回目と2回目以降でヒット率が異なる。

3．パイプライン処理

パイプライン処理は計算機の高速化の歴史の中で最も成功した手法の1つである。今後も多くの高速計算機にパイプライン処理は用いられるであろう。しかし、パイプライン処理においても単純にハードウエアをつぎ込んでパイプライン段数を増やせばその分だけ処理速度が増すわけではない。一般にはパイプライン段数が増えるほどオーバーヘッドも増し、パイプライン段数が増すにつれて処理速度が飽和する傾向がある。従って、パイプライン段数を決定する場合には具体的実現手段を考慮して注意深く検討する必要がある。

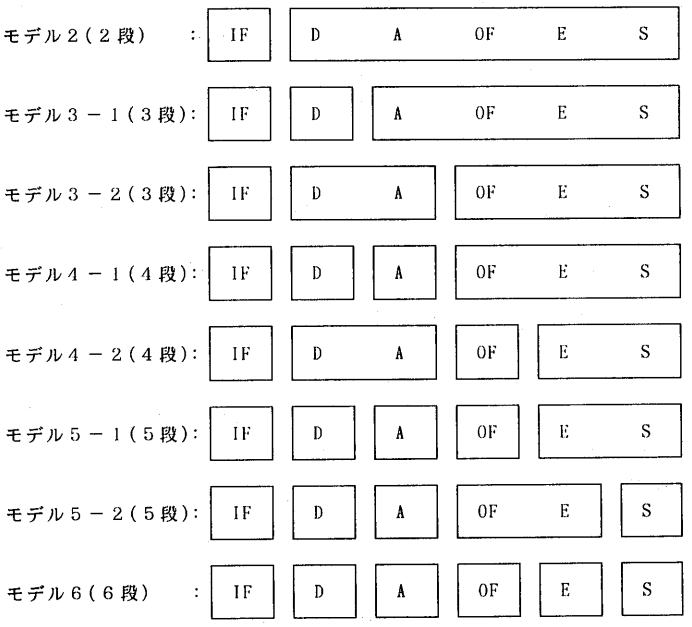

図3.1　パイプライン処理モデル

パイプライン処理方法の違いにより8種類のモデルを考えてシミュレーションによる評価を行った。

本報告ではATOMの具体的パイプラインモデルとして図3.1に示す8種類のパイプラインを考え、その効率を調べた結果について述べる。1つの命令の処理はその命令をメモリからフェッチして実行を完了するまでにいくつかの処理に分けることができる。図3.1のモデル6は処理を6つに分けたもので1つの命令の処理を命令フェッチ(IF)、命令デコード(D)、オペランドアドレス計算(A)、オペランドフェッチ(OF)、実行(E)、オペランドストア(S)に分けたものである。図3.1のその他のモデルはモデル6の各処理を組み合わせてパイプラインステージを2段から5段にしたものである。

図3.1に示した各パイプラインモデルはIBM360シリーズのように汎用アドレシングモードで指定できるオペランドの数が1命令中に1個である命令セットを持つ計算機に対してよく用いられたパイプラインであり[1]、単純にATOMに適合しても効率が良いとは限らない。ATOMでは汎用

アドレッシングモードで指定できるオペランドを１命令中に２つ持つフォーマットの命令があり、このフォーマットの命令を図3.1に示すモデルのパイプラインで単純に処理しても効率が良くない場合がある。われわれはこの問題を解決するため汎用アドレッシングモードで指定できるオペランドが２つある命令は２つの処理単位に分けてパイプラインを流すことにした。１つの命令を２つのパイプライン処理単位に分解する作業はデコーダが行う。

図3.1に示す各パイプラインステージの処理は最小１クロックを要する。オペランドストアステージ以外はたとえそのステージの処理が不要でもパイプラインステージが独立に存在すればそのステージを命令が通過するのに１クロック必要である。またｎ個の処理がまとめられたステージではそのステージ内で必要とされる処理数により処理に必要なクロック数が定まり、最大ｎクロック最小１クロックを必要とする。また命令フェッチステージと命令デコードステージの間には16バイトの命令プリフェッチキューをおく。

図3.1のモデル６を例に各ステージの機能をつぎに説明する。その他のモデルで,複数の処理が１つにまとめられたステージではまとめられた処理のうち必要な処理が１つのステージ内でシリアルに行われる。

(1)命令フェッチステージは１クロックの間に４バイトの命令コードをメモリよりフェッチする。

(2)デコードステージはデコードに必要な命令コードを命令キューより取り出し、１クロックの間の１回の処理で１つの命令の全部あるいは一部をデコードする。ATOMでは命令は可変長であり、ほとんどの基本命令は１回の処理で１つの命令をデコードできるが、命令コードの長い命令では１つの命令をデコードするのに複数回の処理が必要になる。本報告のハードウエアモデルでは１回のデコード処理に最大６バイトの命令コードを命令キューより取り出してデコードすることができる。パイプラインモデル６のようにアドレス計算ステージなど実行までにさらに２つのステージが存在する場合は１回の処理を１クロックで行うが、実行に移るまでの間のステージが１個または０個の場合にはデコードにさらに時間を必要とする。

(3)アドレス計算ステージは１クロックで１回のアドレス計算を行う。メモリ間接アドレッシングやインデックスアドレッシングではさらに多くの処理時間を必要とする。

(4)オペランドフェッチステージは１度に最大４バイトのオペランドをメモリよりフェッチする。オペランドが即値であった場合やアドレス計算結果そのものであった場合にはその値を次のステージに転送するのみである。

(5)実行ステージはALUを１回使用して行う処理につき１クロックを要する。ALUをｍ回使用する処理はｍクロックを要する。

(6)オペランドストアステージは１度に最大４バイトのオペランドをメモリにストアする。ストア先がレジスタの場合にはオペランドストアは実行ステージでの処理時間内に行われるためストアステージの動作は不要である。ストアステージでの処理が不要な命令に対しては処理を実行ステージで終了するため、ストアステージがあってもその処理時間はゼロとなる。

今日の商用の汎用マイクロプロセッサではCPUとメモリを結ぶデータバスは１本しかない。ATOMでもCPUとメモリを結ぶデータバスは４バイトのバス１本のハードウエアモデルを基本とする。この場合、複数のパイプラインステージが同時にメモリに対してアクセスを必要としてもCPUとメモリを結ぶバスは１つしかなく、メモリに対する読み書き処理は１つずつシリアルにしかできない。例えばモデル６ではオペランドストア、オペランドフェッチ、間接アドレスフェッチ、命令フェッチの最大４種類の処理が同時に要求される可能性があるがこれらの要求のうちパイプラインの最も後ろのステージ要求１つしか受け付けられない。命令フェッチはもっとも優先順位が低く他のステージからのバスアクセス要求がないときしか受け付けられない。

このようにCPUとメモリを結ぶバスが１本しかないときはパイプライン処理段数が多くなりCPUの処理能力が増すにつれ、データバスの性能が全体の処理速度に大きく影響を与えるようになる。そこで本報告ではデータバスの処理速度の影響を小さくしてCPU内部処理速度をより明確にするため、データバスが２本あり命令フェッチとデータアクセスが別々に独立して行える場合のハードウエアモデルについても考察した。

4．動的ブランチ予測処理

命令フェッチ、命令デコード、実行はすべての命令に対して必要であるが、オペランドアドレス計算、オペランドフェッチ、オペランドストアは命令によって必要な場合と必要でない場合がある。このためパイプラインステージが多くても計算機の処理速度がその分だけ速いとは限らない。１つの命令が処理を開始されてから終わるまでについてだけ見ればパイプライン段数が少ないほど処理速度の点で有利であり、多段のパイプラインは履歴依存性の強いプログラムには必ずしも良い方法とは言えない。例えばブランチ命令やリターン命令がいくつも連続する場合のパイプライン動作を考えればこのことは明白である。

パイプライン処理にとって不幸なことはこの履歴依存処理が一般のプログラムに多く存在することである。第２章で述べたようにブランチ命令の出現頻度は約20％と高く、しかも全ブランチ命令の平均を取ればブランチする確率の方がブランチしない確率よりやや多いのである。もちろんブランチ以外にも履歴依存処理はあり、そのためにパイプラインが乱れることもある。例えばすでにフェッチしたオペランドと新たにストアするオペランドのコンフリクトやアドレス計算を行った後にアドレスレジスタへの値が書き込まれるのを防ぐためのアドレスレジスタの読み込み待ちがこれにあたるが、これらによりパイプラインが乱れる確率はブランチによってパイプラインが乱れる確率より遙かに小さい。またこれらの原因によるパイプラインの乱れはハードウエアによって確実に防げる場合もある。

本報告ではパイプラインを乱す最大要因であるブランチとパイプライン処理の関係について調べ、ブランチによるパイプラインの乱れを防止する１方法として動的ブランチ予測処理を取り上げ、ATOMでの動的ブランチ予測の効果について報告する。

すべてのブランチ命令について動的ブランチ処理を行うことは難しく、ハードウエア負担を考えると動的ブランチ予測を行う命令の種類を限定した方がよい。

動的ブランチ予測の容易さから分類するとブランチ命令はブランチ条件がスタティックかダイナミックかとブランチ先がスタティックかダイナミックかにより次に示す４つのタイプに分けられる。ここでスタティックとはオブジェクトコード作成時にブランチするかどうかやブランチ先アドレスが決定できることを意味し、ダイナミックとはプログラム実行時になって初めてこれらが決定できることを意味するとする。

(1)第１のタイプのブランチ命令はブランチ条件ブランチ先ともスタティックな命令である。ATOMでは無条件ブランチ命令(BRA)、サブルーチンコール命令(BSR)などがこれに当たる。

(2)第２のタイプのブランチ命令はブランチ条件はダイナミックであるがブランチ先がスタティックな命令である。ATOMでは条件ブランチ命令(Bcc)、ループ制御命令(ACB)などがこれに当たる。

(3)第３のタイプのブランチ命令はブランチ条件はスタティックであるがブランチ先がダイナミックな命令である。ATOMではサブルーチンからのリターン命令(RET)、レジスタ間接アドレスに対するジャンプ命令(JMP)などがこれに当たる。

(4)第４のタイプのブランチ命令はブランチ条件ブランチ先ともダイナミックな命令である。ATOMでは条件トラップ命令(TRAP)がこれに当たる。

本報告ではデコード段階でタイプ１とタイプ２のブランチ命令に対してブランチ予測処理を行うハードウエアモデルを考えた。タイプ１のブランチ命令に対しては無条件にブランチすると判断する。タイプ２のブランチ命令に対しては命令により処理が異なる。ループ制御命令に対しては無条件にブランチすると判断する。条件ブランチ命令に対してはその条件ブランチ命令のアドレスがもつブランチ履歴に従いブランチするかどうかを判断する。あるアドレスの条件ブランチ命令がブランチを起こすとそのアドレスと下位８ビットが一致する条件ブランチ命令がつぎに実行される際ブランチを起こすと予測する。履歴は過去１回の実行結果についてとり、予測が誤ったときは履歴を反転する。なお履歴の初期値は「ブランチしない」とした。

	エラトステネスのふるい				Dhrystone			
	初期値40H		初期値10H		1回目		2回目以降	
	実行	的中	実行	的中	実行	的中	実行	的中
Bcc(taken)	234	189	50	39	9	1	9	5
Bcc(not taken)	66	31	14	14	18	14	18	14
Bcc小計	298	220	64	47	27	15	27	19
ヒット率(Bcc)	73.8%		73.4%		55.6%		70.4%	
BRA	33	33	12	12	10	10	10	10
BSR	0	0	0	0	15	15	15	15
全体計	323	253	86	65	52	40	52	44
ヒット率(全体)	78.3%		75.6%		76.9%		84.6%	

表4.1 ATOMにおける動的ブランチ予測のヒット率

動的ブランチ予測のヒット率は約80％である。

ブランチ予測方法とそのヒット率についてはすでにいくつかの研究が行われており、本報告で取り上げた動的ブランチ予測方法でも実際のアプリケーションでかなりのヒット率が期待できる[1][7][8]。本報告で用いた「エラトステネスのふるい」のベンチマークプログラムによるシミュレーションでは動的ブランチ予測の対象となる命令はBRA,Bccの２つであり、表4.1に示すように約75％のヒット率を得ている。ベンチマークの性格上初期値が大きいほどループの実行回数が増え、動的ブランチ予測のヒット率も上がっている。参考のため「Dhrystone」のベンチマークで動的ブランチ予測を行った場合のヒット率もあわせて示す。「Dhrystone」のベンチマークでもほぼ同じヒット率となる。

デコード段階でブランチしないと判断された場合にはプログラムカウンタ値を計算する回路で現在デコード処理を行っている命令のアドレスにその命令の命令長を加えてその命令に引き続くアドレスに存在する命令をつぎにデコード処理する。ブランチすると判断された場合はプログラムカウンタ値を計算する回路で現在デコード処理を行っている命令のアドレスにブランチ幅を加えてブランチ先アドレスに存在する命令を次にフェッチしてデコード処理する。この動作を本報告ではプリブランチと呼ぶことにする。

タイプ３とタイプ４の命令に対してはブランチ先がダイナミックであるためブランチ先アドレスをデコード段階で求めることが難しく、今回のハードウエアモデルではこのタイプのブランチ命令に対してはプリブランチを行わないことにした。

プリブランチ処理が間違っていた場合やタイプ３、タイプ４のブランチ命令に対しては命令が実行段階に移った時点でブランチを起こす。

プリブランチを行う場合、デコードから２クロック後にブランチ先アドレスを出力できる。実行段階でブランチする場合にはパイプラインモデルによりブランチ命令実行によるパイプラインのオーバーヘッドが異なる。実行段階に至るまでの前処理が多ければ多いほど実行段階でのブランチ命令実行によるオーバーヘッドも大きくなる。

	性能比		ﾌﾞﾗﾝﾁによる改善度	
	ﾌﾞﾗﾝﾁ予測無し (1)	ﾌﾞﾗﾝﾁ予測有り (2)	(2)-(1)	(2)/(1)
ﾓﾃﾞﾙ2	1.00	1.02	0.02	1.02
ﾓﾃﾞﾙ3-1	1.18	1.29	0.11	1.09
ﾓﾃﾞﾙ3-2	1.61	1.68	0.07	1.05
ﾓﾃﾞﾙ4-1	1.76	1.97	0.21	1.12
ﾓﾃﾞﾙ4-2	1.71	1.80	0.09	1.05
ﾓﾃﾞﾙ5-1	2.22	2.45	0.23	1.10
ﾓﾃﾞﾙ5-2	1.81	2.01	0.20	1.11
ﾓﾃﾞﾙ6	2.37	2.56	0.19	1.08

表5.1 各パイプラインモデルのシミュレーション結果(1)

各パイプラインモデルに対しデータバスが１本の場合について行ったシミュレーション結果をモデル２の動的ブランチ予測処理なしの場合を１として比較した。

5. シミュレーション結果

図3.1に示した各種パイプラインモデルについて「エラトステネスのふるい」のベンチマークを実行して性能を比較した。各々のモデ

ルについてメモリバスが１本の場合に動的ブリブランチ処理を行った場合と行わなかった場合それぞれに要したクロック数をもとにモデル２のブランチ予測なしの場合を１としたときの性能比、動的ブランチ予測処理による性能改善度を表5.1に示した。

	性能比		ﾌﾞﾗﾝﾁによる改善度	
	ﾌﾞﾗﾝﾁ予測無し (1)	ﾌﾞﾗﾝﾁ予測有り (2)	(2)-(1)	(2)/(1)
ﾓﾃﾞﾙ2	1.00	1.02	0.02	1.02
ﾓﾃﾞﾙ3-2	1.65	1.75	0.10	1.06
ﾓﾃﾞﾙ4-1	1.86	2.09	0.23	1.12
ﾓﾃﾞﾙ5-1	2.34	2.65	0.31	1.13
ﾓﾃﾞﾙ6	2.46	2.79	0.33	1.14

表5.2　各パイプラインモデルのシミュレーション結果(2)

各パイプラインモデルに対しデータバスが２本の場合について行ったシミュレーション結果を表5.1とおなじ相対値で比較した。

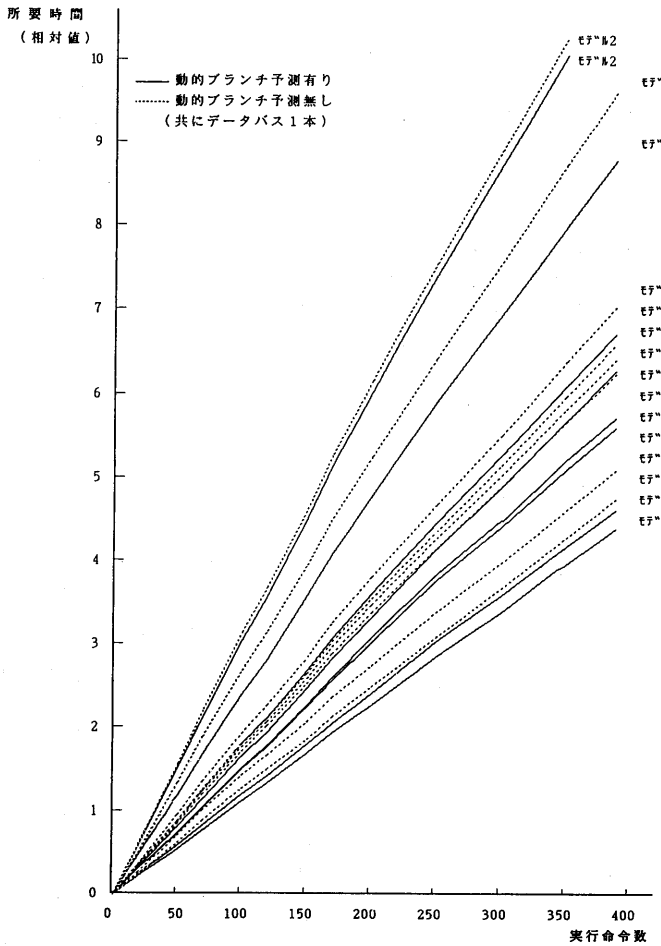

図5.1 実行命令数と処理時間の途中経過

どのモデルも処理時間は実行命令数に比例して増大する。

表5.1に示した数値は初期値を10H(16進数)とした場合のものであるが「エラトステネスのふるい」のベンチマークでは所要時間は処理命令数にほぼ比例して増加し、初期値を大きくしても相対的関係はまったく同じである。これは図5.1に示した各種のモデルの場合の命令実行数と処理時間の途中経過より明らかである。

またモデル２、モデル６とパイプライン段数３、４、５の場合の性能がよかった方のモデルであるモデル3-2、モデル4-1、モデル5-1についてはデータバスが２本のハードウエアモデルについても同じようにシミュレーションを行った。その結果は表5.2に示す。

シミュレーションは大型計算機上でPL/1とGPSSにより各モデルを記述することにより行った。

６．パイプライン構成と性能

パイプライン段数３、４、５段の場合には図3.1に示したようにそれぞれ２つのモデルを考えたが、同じパイプライン段数でもパイプライン構成の違いによりこれらのモデル間で大きな性能差が生じた。この性能差は動的ブランチ処理を行った場合も行わなかった場合も同程度生じる。

データバスが１本の場合についてパイプライン段数２、３、４段各２つ計６個のモデルについて図5.2に性能比較を示す。モデル3-1とモデル3-2、モデル5-1とモデル5-2では特に性能差が著しい。これはモデル3-2とモデル5-1では各パイプラインステージに負荷がバランスよく分配されているが、モデル3-1とモデル5-2では各パイプラインステージへの負荷分配の

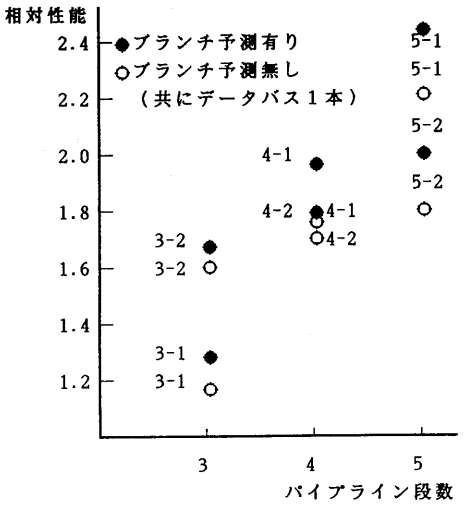

図5.2 パイプライン構成の違いによる性能比較

パイプライン処理段数が同じでも、その構成方法が異なると性能は大きく変わる。

バランスが悪いためと考えられる。

パイプライン処理を行う場合には各パイプラインステージに負荷をバランスよく分配することが大切である。たとえばパイプラインステージ間の負荷バランスがよいモデル3-2のデコードとアドレス計算のステージをデコードステージとアドレス計算ステージに分けてモデル4-1にしたり、オペランドフェッチと実行とオペランドストアのステージからオペランドフェッチを分けて別ステージとしてモデル4-2にしてもあまり性能向上はない。しかし、この際分割されなかった側のステージも分割してモデル5-1にすれば再び負荷バランスがよくなり、かなりの性能向上となる。またパイプラインステージ間の負荷バランスが悪いモデル3-1のアドレス計算とオペランドフェッチと実行とオペランドストアのステージからアドレス計算を分けて別ステージにする場合もかなりの性能向上となる。

このようにパイプライン処理方式はその計算機のもつ命令セット自身の特徴やハードウエア構成の特徴を十分考慮して決定されるべきものである。

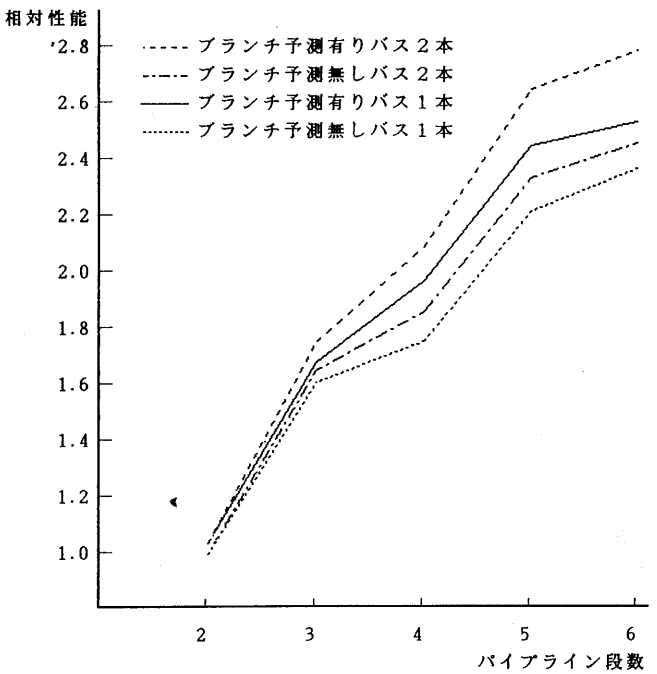

図5.3 各種パイプライン段数における動的ブランチ予測処理の効果

パイプライン段数が増加するにつれ動的ブランチ予測処理の効果も大きくなるが、データバスが１本の場合にはその効果に飽和傾向が見られる。

7．動的ブランチ予測処理の効果

モデル２、モデル６とパイプライン段数３、４、５の場合の性能がよかった方のモデルであるモデル3-2、モデル4-1、モデル5-1について動的ブランチ処理を行った場合と行わなかった場合それぞれについてパイプライン段数と性能の関係を表すグラフを図5.3に示す。

パイプライン段数が多いほど処理性能はよいが処理性能はパイプライン段数に比例してはよくならない。データバスが１本のとき動的ブランチ予測処理がない場合パイプライン段数を６段にしても２段の場合に比べ2.37倍の性能向上効果し

か得られていない。動的ブランチ予測処理を行った場合には2.50倍の性能向上効果が得られている。データバスが２本の場合にはこの差はさらに大きい。動的ブランチ予測処理をおこなった場合でも処理性能はパイプライン段数に比例してよくなるわけではない。しかし動的ブランチ予測処理はパイプライン処理のオーバーヘッドを吸収してパイプライン処理による処理性能の向上を比例関係に近づける効果がある。

パイプラインモデルにより違いはあるが図5.3に示すように動的ブランチ予測処理を行った場合にはすべてのモデルに対して性能改善が成されている。データバスが２本のハードウエアモデルでは動的ブランチ予測処理による性能改善効果はパイプライン段数が多いほど大きいがデータバスが１本のハードウエアモデルでは動的ブランチ予測処理による性能改善効果はパイプライン段数を５段以上にしてもほとんど同じである。これはパイプライン段数が多くCPU内部の処理性能が高い場合にはデータバスの性能が全体の処理性能に大きな影響を与え、動的ブランチ予測処理による性能改善効果を制限してしまっているためであると思われる。データバスが１本の場合デコードステージでプリブランチ処理を行ってブランチ先命令をフェッチしようとしてもプリブランチ処理を行った命令に先行する命令がまだパイプライン中で処理されているわけで、オペランドフェッチステージやオペランドストアステージがデータバスを使用するため命令フェッチが待たされることが多いためである。

本報告のパイプラインモデルでのシミュレーションではパイプライン段数が３段以上の場合に動的ブランチ処理により5～14％の性能向上が見られた。「エラトステネスのふるい」のベンチマーク以外のプログラムに対しても同じ効果があるかどうかは本報告でのシミュレーションの範囲では不明である。しかし動的ブランチ予測処理の効果を決定する主な要因は第２章で述べたブランチ命令の実行頻度と第４章で述べた動的ブランチ予測のヒット率であり、この２点に関してすでに述べたように「エラトステネスのふるい」のベンチマークは実際のアプリケーションを代表している。従って、多段のパイプライン処理を行うATOMは多くのアプリケーションで動的ブランチ処理により10％前後の性能改善効果を得ることが期待できる。

８．おわりに

本報告ではオリジナル32ビットマイクロプロセッサの開発にあたり新しいマイクロプロセッサのアーキテクチャに適する各種のパイプライン方式を検討した結果について報告した。

パイプライン処理におけるブランチ命令のオーバーヘッドをなるべく少なくするため、ブランチ命令の履歴に従ってブランチするかどうかを判断する動的ブランチ予測処理を採用した。

パイプライン処理方式の例として８種類のパイプラインモデルを考え、「エラトステネスのふるい」のベンチマークプログラムに対して各種モデルにおける動的ブランチ予測処理の効果をシミュレーションにより検討した結果、パイプライン段数が４段以上の場合に動的ブランチ処理により10％前後の性能向上が見られた。

動的ブランチ予測にしたがうプリブランチ処理はデコードステージで行う。このため本質的にはパイプラインが深く、デコードステージと実行ステージの間により多くのステージが存在するほど動的ブランチ予測処理の効果は大きいはずである。しかし、CPUとメモリを結ぶデータバスが１本のときには処理性能の改善に飽和傾向が現れた。データバスを２本にした場合は飽和傾向はなかった。

動的ブランチ予測処理などによるCPU内部の高性能化の効果を十分生かし、マイクロプロセッサの処理性能をさらに向上させるためには、今後、データバスのバンド幅を広げるあるいはそれにかわるものとしてキャッシュやブランチターゲットバッファ[1]を内蔵するなどの手法が必要とされるだろう。

謝辞

新しいマイクロプロセッサ開発の機会と本報告の機会を与えていただいた三菱電機（株）LSI研究所マイクロプロセッサ開発部榎本部長に感謝します。本報告の研究に際し多くの助言とはげましをいただいた同マイクロプロセッサ開発部の方々に感謝します。また実行命令頻度やパイプラインモデルに関し助言をいただいた三菱電機（株）情報電子研究所の方々に感謝します。

参考文献

1. Jhonny K.F.Lee and Alan Jay Smith,"Branch Prediction Strategies and Branch Target Buffer Design,"IEEE COMPUTER,Vol.17,no.1,January 1984, pp.6-22.

2. Joseph A.Lukes,"HP Precision Architecture Performance Analysis", Hewlett-Packard Journal, August.1986, pp.30-39.

3. Cheryl A.Wiecek,"A Case Study of VAX-11 Instruction Set Usage For Compiler Execution", Proceedings of Symposium on Architectural Support for Programming Languages and Operating Systems,ACM,March 1982, pp.177-184.

4. "80386 32ビットマイクロプロセッサの概要",Intel Japan k.k.,November 1985.

5. Jim Gilbreath and Gary Gilblieath,"Eratosthenes Revisited Once More through the Sieve",BYTE January 1983, pp.283-326.

6. Reinhold P. Weicker,"Dhrystone: A Synthetic Systems Programming Benchmark",Communications of the ACM, Vol.27, No.10, October 1984, pp.1013-1030.

7. Scott McFarling and John Hennessy,"Reducing the Cost of Branches",Proceedings of the 8th Annual International Symposium on Computer Architecture,June 1986,pp.396-403

8. Paul Chow,"MIPS-X Architectural Overview Instruction Set Design", Stanford University, May 1985.