計算機アーキテクチャ　107－15
（１９９４．　７．　２１）

V++プロセッサにおける適応型再構成機能を活用した分岐処理方式

金岡弘記 †　　高木浩光 †　　有田隆也 ‡　　川口喜三男 †

名古屋工業大学 †　　名古屋大学 ‡

細粒度並列性を効率的に抽出するVLIWプロセッサに対し、規定型再構成機能と適応型再構成機能を付加することにより、コード量の増大化と実行時間変動に対する弱さを解決するV++プロセッサをすでに提案している。

本稿では、V++プロセッサにおける分岐命令の仕様やパイプライン構成を定めるとともに、２つの再構成機能を活用することにより、分岐処理方式と投機的実行が効率的に実現されることを示す。

Branch Architecture of V++ Processor Utilizing the Facilities for Adaptive Restructuring

Hiroki KANEOKA †　　Hiromitsu TAKAGI †

Takaya ARITA ‡　　Kimio KAWAGUCHI †

Nagoya Institute of Technology †　　Nagoya University ‡

An extended model of VLIW (Very Long Instruction Word) architecture called V++ has been proposed. V++ processor has facilities for predetermined restructuring and adaptive restructuring. Therefore, V++ processer can reduce code size remarkably and become robust against dynamic variation in instruction latency.

This paper describes the branch architecture utilizing both predetermined restructuring function and adaptive restructuring function, and shows the specifications of branch instructions.

1 はじめに

VLIW プロセッサでは、コンパイル時に並列性が抽出されるため、実行時には並列性抽出にともなうオーバヘッドがなく、また、ハードウェアを単純なものとすることができる。しかし、VLIW プロセッサの問題点として、コード量が増加することとオペレーションの実行時間変動に対して弱いということが挙げられる。これらを解決する手段として、規定型再構成と適応型再構成を用いた V++プロセッサアーキテクチャ[1][2][3] を提案している。本稿では、適応型再構成のための同期方式として重複可能バリア同期を用いた場合について、規定型再構成機能を活用した分岐予測方式と適応型再構成機能を活用した投機的実行方式を提案する。また、提案方式を効率よく実現するための分岐命令の仕様について述べる。

2 V++プロセッサの概要

2.1 規定型再構成

規定型再構成とは、コンパイラ等によりあらかじめオペレーションに付加された実行タイミングの情報(遅延タグ) に基づいて、実行時に命令の再構成を行なうものであり、各オペレーションは、フェッチ後指定されたクロックだけ遅らされた後、実行される。これにより、フェッチされた 1 つの Instruction Word(命令語) を構成するオペレーションの組と、実行される 1 つの命令語を構成するオペレーションの組とが異なることが許され、基本ブロック間でのオーバラップが基本ブロック間の静的なコード移動なしに可能となる。その結果、コード量を減らしコードの命令密度を高くすることができる [4]。

2.2 適応型再構成

適応型再構成とは、実行時に種々の要因により発生するオペレーション実行時間の変動のために生じる不要な待ちを、ハードウェアにより削減し、全体の処理時間の増大を抑えることを可能としたものである。具体的には、高速な同期機構 (重複可能バリア同期機構 [5]) を用いることにより、ユニット間の実行タイミングのずれを吸収し、無駄な待ちを削減する。また、動的にオペレーション間の先行制約を保証するので、従来の VLIW で必要とされていた、タイミングを保証するためだけの NOP は不要となる。

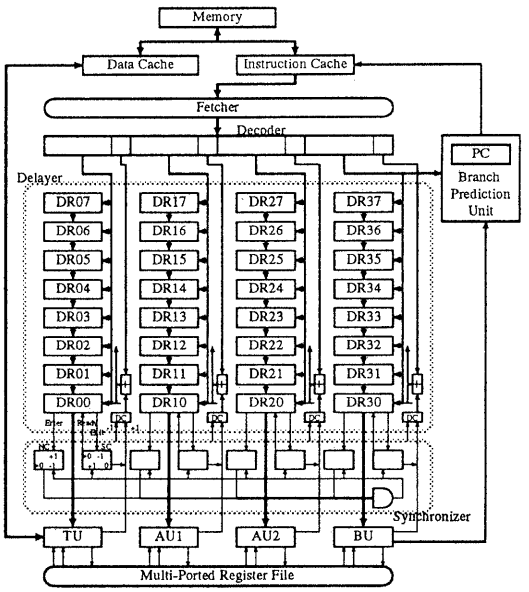

図 1: V++プロセッサの基本構成

2.3 プロセッサの基本構成

V++プロセッサの基本構成を図 1に示す。V++プロセッサには、規定型再構成を行なうために遅延レジスタ (Delay Register) と、適応型再構成を実現するために同期ユニット (Synchronization Unit) および同期によって生ずるユニット間の相対的な実行の遅れを示す遅延カウンタ (DelayCounter,DC) を、従来の VLIW プロセッサに付加している。本稿では、同期ユニットとして重複可能バリア同期機構を用いることを仮定しており、遅延レジスタの段数は 8 段とし、実行ユニットは、転送ユニット (TU)×1・演算ユニット (AU)×2・分岐ユニット (BU)×1 の構成としている。

3 命令パイプライン構成

命令パイプラインは、

1. F : 命令フェッチ
2. D : 命令デコード・レジスタ読み出し
3. E : 実行

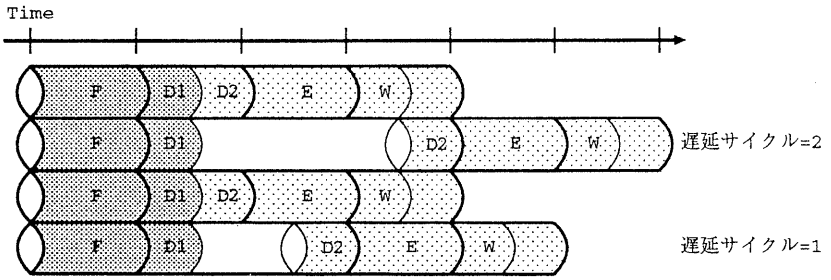

図 2: 実行遅延の例

4. W : レジスタ書き込み

の 4 ステージ構成とし、D ステージは次の 2 つのフェーズからなる。

1. D1 : 命令デコード
   命令コード (レジスタ番号を含む) のデコードを行なう。それと同時に、規定型再構成の遅延タグを遅延カウンタの値 (3 ビット) と加算し、これをデコードすることによって、オペレーションの遅延レジスタ (8 段) への格納位置を決め、フェーズの最終段階で、命令デコードの結果を指定の遅延レジスタにラッチする。ただし、オペレーションが NOP である場合は、遅延レジスタに上書きするのではなく、どの遅延レジスタにもラッチしないものとする。

2. D2 : レジスタ読み出し
   遅延レジスタの最終段 (DRx0) に格納されているレジスタ番号のデコード結果に基づいて、その時点での整数レジスタ/浮動小数レジスタの出力をラッチする。

D1 フェーズから D2 フェーズにかけて、規定型再構成により指定されたステップ数、また適応型再構成の同期操作のために実行が遅延される。このオペレーションの実行遅延は遅延レジスタにより行なわれる。遅延レジスタはクロックに同期して、保持しているデコード結果をシフトする。例えば、D1 フェーズで図 1の DR12 にデコード結果がラッチされた場合には、それは次のクロックで DR11 へ, また次のクロックで DR10 へとシフトする。これは 2 サイクルの実行遅延を意味する。ただし、遅延レジスタの最上段 (DRx7) には NOP をシフトインするものとする。

命令パイプライン構成を踏まえた実行遅延の例を図 2に示す。上から 2 番目のオペレーションは、規定型再構成と適応型再構成で遅延サイクルが計 2 サイクルであり、下端のオペレーションには遅延サイクルが計 1 サイクルであることを表している。

	F	D1	D2	E	W	
Load/Store オペレーション	命令デコード	命令フェッチ	レジスタ読み出し	メモリアクセス	レジスタ書き込み	
演算オペレーション				演算実行		

図 3: 命令パイプライン処理過程

図 3に、オペレーションの種類別に命令パイプラインの処理過程を示す。Load/Store オペレーションにおけるメモリアクセスのアドレスはレジスタに格納されており、アドレス生成は演算オペレーションで行なう。

4 分岐アーキテクチャ

4.1 分岐ユニットのオペレーションセット

4.1.1 アドレッシングモード

分岐オペレーションのアドレッシングモードは、以下の 2 つとする。

1. PC 相対 (PC + 符号付き 11 ビット)

2. レジスタ間接の PC 相対 (PC + Register)

表 1: 分岐オペレーション一覧

使用例	オペレーション名	意味
J label	jump	PC ← label
JR R1	jump register	PC ← R1 + (PC + 16)
BNE R2 label	branch not zero	if (R2 $\neq$ 0) PC ←label
BEQ R2 label	branch zero	if (R2 = 0) PC ← label
BGT R2 label	branch greater than	if (R2 > 0) PC ← label
BLT R2 label	branch less than	if (R2 < 0) PC ← label
BGE R2 label	branch greater equal	if (R2 $\geq$ 0) PC ← label
BLE R2 label	branch less equal	if (R2 $\leq$ 0) PC ← label

label = (PC + 16) + offset

命令語長は 32 ビット× 4 としているため PC のインクリメントは 16 バイト単位

4.1.2 分岐条件決定方式

分岐条件決定方式として branch-on-condition 方式を採用する。branch-on-condition 方式とは、条件分岐における条件判断のコンディション生成を算術・論理演算オペレーションによって行ない、その生成したコンディションをもとに、条件分岐オペレーションが条件に合った分岐先を決定する方式である。ただし、V++では、条件計算の並列処理も可能とするために、汎用レジスタにコンディションを格納するものとし、コンディションは通常の算術・論理演算オペレーションによって計算する。そして、分岐オペレーションがこのコンディションを 0 と比較することによって分岐条件が決定される。

この方式は、DLX[7] などで採用されている advanced-conditioning 方式に比較して、コンディションコードの共通化による最適化を可能とする利点がある。

4.1.3 オペレーションの種類

表 1に分岐オペレーションの一覧を示す。

call 操作と return 操作の実現について、DLX では jump と同時に戻り先アドレス (PC + 4) を R31 に格納する JAL(jump and link) 命令を用意しており、call 操作を JAL 命令で、return 操作を jump R31 とすることで実現している。しかし、実行ユニットが非均質で分岐専用ユニットを持っている V++の場合、JAL 命令を用意するとレジスタファイルの分岐ユニットからのポートが、読み出し用 1 ポートに加えて書き込み用 1 ポートも必要となる。そこで V++では、return 操作を JR オペレーションで実現するものとし、戻り番地を return 操作用の JR オペレーションからの相対番地として、call 操作用の J オペレーションの実行前にレジスタにセットしておくものとする。これによりマルチポートレジスタの分岐ユニット用ポートは、読み出し用 1 つに抑えられている。

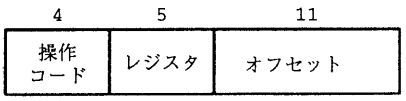

図 4: オペレーション・フォーマット

4.1.4 オペレーション・フォーマット

分岐オペレーションのフォーマットは、予測分岐先アドレスをデコードステージにおいて生成できるように、オフセットの位置と幅を固定としている。

分岐ユニットに発行されるオペレーションの種類は、分岐オペレーション 8 種 (表 1参照) と NOP で総数 9 種となる。よって操作コードは 4 ビットとなる。レジスタ数は 32 個としているので、レジスタを指定するために 5 ビットを用意する。オフセットは、分岐オペレーションと行き先との距離のデータ [7] から、10 ～ 12 ビット程度で十分であると判断し、符号付きで 11 ビットとする (図 4)。規定型再構成用のタグと適応型再構成用のタグの見積りは今後の課題である。

表 2: 分岐ペナルティ

分岐オペレーションの移動	分岐予測	分岐結果	分岐ペナルティ (サイクル)	BTB を用いた場合の分岐ペナルティ
可能	not-taken	not-taken	0	0
可能	not-taken	taken	2	2
可能	taken	not-taken	2	2
可能	taken	taken	0	0
不可能	not-taken	not-taken	0	0
不可能	not-taken	taken	2	2
不可能	taken	not-taken	3	2
不可能	taken	taken	1	0

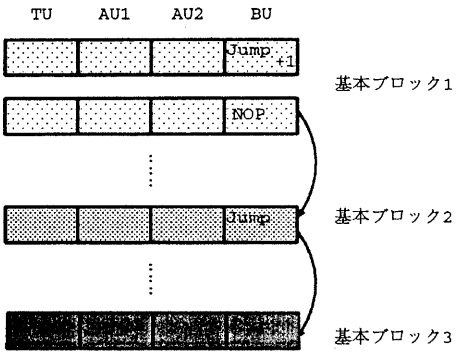

図 5: 分岐予測の可能な場合と不可能な場合

4.2 分岐予測

VLIW の特長を保持するという意味で、最小限度のハードウェア増に抑えるために、分岐先バッファ (Branch Target Buffer, BTB)[8, 9] などを用いない静的分岐予測方式をとる。コンパイラによって分岐命令に付加された分岐予測情報 (taken または not-taken の予測を示す 1 ビット) によって、デコードステージで予測分岐先アドレスを生成し、そのアドレスにしたがって次の命令語フェッチを行なう。通常ではこのとき、デコードステージが終了するまで分岐先アドレスを得られないために、次にフェッチされる命令語はコード上の次の命令語となる。分岐予測が taken のときはその命令は無効化されることとなる。しかし V++プロセッサでは、規定型再構成機能を活用することで、分岐オペレーションを本来分岐オペレーションが実行されるべき位置より早くフェッチすることができ、これにより予測分岐先アドレスを早く生成することができる。

例として、図 5に示すコードを考える。基本ブロック 1 から基本ブロック 2 に、また基本ブロック 2 から基本ブロック 3 に分岐予測をしていることを表している。図 5のコードを実行した場合の命令パイプラインのタイミングを図 6に示す。基本ブロック 1 は分岐オペレーションのフェッチを早められる場合であり、デコードステージで生成した予測分岐先アドレスを使って即座に分岐先 (基本ブロック 2) の命令語をフェッチすることができる。これに対し、基本ブロック 2 のようにサイズが 1 命令語しかない場合には、分岐オペレーションのフェッチを早めることができず、デコードステージで予測分岐先アドレスが生成される前に基本ブロック 2 に後続する命令語をフェッチしてしまう。このとき、この分岐オペレーションの分岐予測が taken の場合には、そのフェッチした命令語を無効化することになる。

これを、BTB を用いた分岐予測方式と比較する (表 2参照) 。分岐オペレーションの移動が可能な場合、分岐予測が正しかったときは、分岐予測が taken, not-taken のどちらの場合にも分岐ペナルティは 0 である。分岐予測が正しくなかったときは、実際の分岐先アドレスは実行ステージで生成されるため分岐ペナルティは 2 となる。次に、分岐オペレーションが移動不可能な場合、次にフェッチする命令語は後続する命令語となり、分岐予測が not-taken の場合には

ペナルティはなく、分岐予測が taken の場合には逆に分岐予測のためにペナルティが 1 増加する。

BTB を用いた場合については、予測分岐先アドレスがバッファに登録されいている場合、分岐予測が当たっていれば分岐ペナルティは 0 で、外れた場合は 2 となる。予測分岐先アドレスがバッファに登録されていない場合には、分岐予測は not-taken としてふるまい、実行時に生成するアドレスをバッファに登録する。よって、分岐結果が taken のときには分岐ペナルティは 2 となり、not-taken のときには分岐ペナルティは 0 となる。

V++ の分岐方式は、分岐オペレーションのフェッチを早められない場合で taken 予測となる場合に、分岐ペナルティの期待値が大きくなる。しかし、コンパイラによりできるだけ not-taken 予測となるような基本ブロックのスケジューリングを行なっておくことで、キャッシュ同様のハードウェアを必要とする BTB を使わなくても、規定方再構成の機能を活用することによって、BTB と同様の分岐予測が可能となる。

以下に、分岐ユニットの命令パイプライン構成をオペレーションの種類別に示す。条件分岐オペレーションは、デコードステージで予測分岐先アドレスを得るためのアドレス計算を行ない、実行ステージでは、条件を決定し分岐予測との比較を行なう。分岐予測が正しくなかった場合は、正しい分岐先アドレスを PC にセットする。Jump オペレーションはデコード時に taken 先のアドレスを計算するので、オペレーションの実行はデコードステージで終了する。レジスタ間接によるジャンプ (JR オペレーション) では、ジャンプ先アドレスが確定するまで、PC へのアドレスセットは行なわないものとする。

4.3 適応型再構成による部分投機的実行

分岐オペレーションがフェッチされると、次にフェッチされる基本ブロックが分岐予測によって決定されるが、分岐先が確定したときに分岐予測が失敗していたならば、通常、すでにフェッチしていた予測分岐先の命令を無効化せねばならない。しかし逆に、無効化しないで処理を続行させるという手法もある。これは投機的実行の一種といえる。

投機的実行を許しかつプログラムの正しい実行を保証するためには、しばしばレジスタファイルの多重化が導入されるが、V++ では、レジスタの多重化をすることなく、適応方再構成の機能を活用することで、部分的に投機的実行を実現する。具体的には、

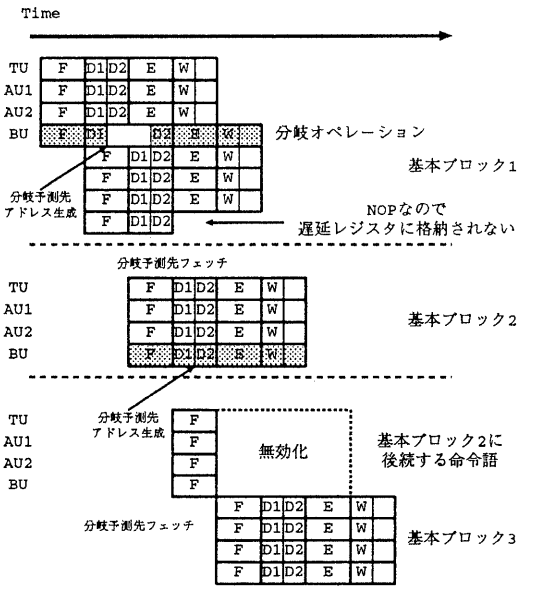

図 6: 命令フェッチタイミング

分岐予測が失敗していたときに実行してもプログラムの実行結果に影響しない部分 (副作用のない領域) を、コンパイラの最適化により最大限に確保したうえで、適応型再構成のための同期機構を用いて、副作用のない領域を越えて投機的実行が進まないことを保証する。

図 7の例は、基本ブロック B1 と基本ブロック B2 との境界における実行タイミングを示している。図 7 (a) は、動的変動のないときの実行タイミングであり、これはコンパイラが予測したオペレーション実行時間通りに実行が進んだ場合である。この場合は投機的実行は行なわれない [1]。図 7 (b) は、図 7 (a) に比べて AU2 と BU に含まれるオペレーションの実行時間がコンパイラが予測したオペレーション実行時間より長くなった場合で、投機的実行を行なわないとした場合である。基本ブロック B1 の分岐オペレーションに依存するオペレーションの終了時刻が遅れたなどの理由により、基本ブロック B2 の TU と AU1 の実行開始までもが、制御依存が解消されるまで遅らされている。図 7 (c) は、(b) のタイミングの

[1] コンパイラによる基本ブロック間の静的コード移動をも含めて投機的実行と呼ぶならば、ここでも投機的実行がなされる場合があるといえるが、本稿では静的コード移動は「投機的実行」に含めていない。

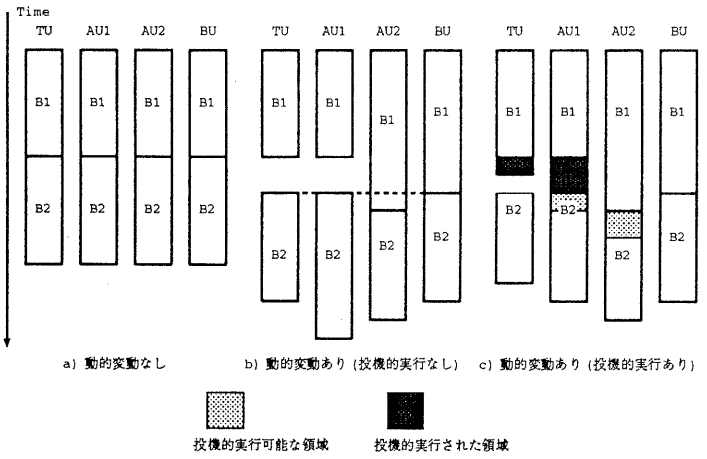

図 7: 適応型再構成を用いた投機的実行の概念

ときに投機的実行を行なった場合であり、基本ブロック B1 の BU の制御依存が解消される前に基本ブロック B2 の TU と AU1 は、副作用のない範囲まで実行を進める。基本ブロック B2 の TU は、副作用のない範囲のオペレーションをすべて実行してもまだ制御依存が解消されていない場合であり、この場合は同期処理によって制御依存が解消されるまでそれ以降のオペレーションの実行が待たされる。

分岐予測が失敗したときに、投機的実行がなされたユニットがある場合は、同期機構の状態を正しく保つための特別な対処が必要である。その方法として次の 2 つを検討している。どちらを選択するかについては今後の詳しい性能評価の結果に委ねることにする。

- 同期機構の状態を初期化する方法

  投機的実行がなされているときに、分岐予測が失敗した場合、同期機構の状態を初期化する。このとき、BU 以外のユニットが基本ブロックの境界に達しているものと達していないものとがある場合は、すぐには同期機構の状態を初期化できないので、境界に達していないユニット全てが境界に達した後、同期機構の状態を初期化して次の基本ブロックを実行する。一方、BU 以外の全てのユニットが基本ブロックの境界に達していなかった場合は、同期機構の状態に矛盾が生じないので初期化する必要がない。このような動作により同期機構の状態は常に正しい状態に保たれる。

  分岐予測が失敗した場合、基本ブロックの境界の検知が必要である。それは、DC の値を調べることによって可能である。BU が持つ DC に対するそれ以外のユニットの DC の相対値が正ならば、そのユニットは基本ブロックの境界に達していないと判断でき、0 以下ならば、そのユニットは基本ブロックの境界に達していると判断できる。

  この方法では、基本ブロック毎に実行が再開可能になるように同期タグの付加に関する制約が付く。

- 同期機構の状態を保存する方法

  分岐予測された基本ブロックのユニットの中で一番最初のユニット (図 8では AU1) のオペレーションが投機的実行する前に、そのときの全体の同期機構の状態を保存しておく。その後、まだ投機的実行されていないユニットのオペレーションに付加されている同期タグは、そのユニットのオペレーションが投機的実行されるまで差分として保存する。この 2 つの値により分岐予測が失敗したときにもとの同期機構の状態を復元できる。

  この方法では、初期化する方法のように同期タグの付加に対する制約が付くことはないが、同期状態を保存するためのハードウェアが必要と

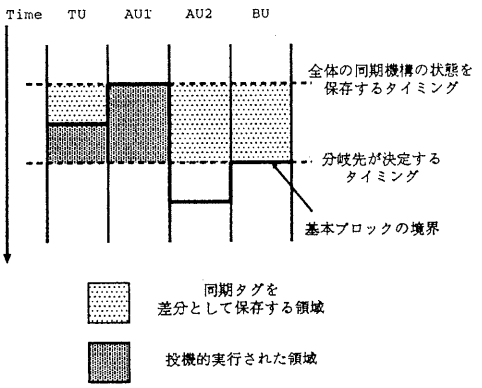

図 8: 同期機構状態の保存タイミング

なる。

このような投機的実行方式には、

1. 投機的に実行したオペレーションを無効化する、またはレジスタファイルの多重化するなどのためのハードウェアコストがかからない。

2. 機能ユニットに余裕が生じた場合にのみ投機的実行を行なうので、予測が外れた場合にかえって性能が低下してしまうといった現象が起きない。

などの特長がある。ただし、コンパイラの最適化によっても、副作用のない範囲を十分に大きくできない場合には、投機的実行の効果は限定されたものとなる。

5 おわりに

V++プロセッサの分岐命令の仕様とパイプライン構成を明らかにするとともに、規定型再構成機能を活用した分岐予測方式と、適応型再構成機能を活用した投機的実行方式とを提案した。今回本稿で新たに提案した両手法は、新たに大きなハードウェアを付加することなく、これまでのV++プロセッサに備わっていたハードウェアを活かすことにより、効率的に実現されている。

今後の課題としては、その他の機能ユニットの詳細の決定、規定型再構成用のタグと適応型再構成用のタグの詳細の決定、提案方式のシミュレーションによる性能評価、提案方式を効果的に利用するコンパイラの開発、などが挙げられる。

参考文献

[1] 有田隆也, 曽和将容,"動的再構成型 VLIW プロセッサアーキテクチャ V++" , 並列処理シンポジウム JSPP '92 論文集, pp.265–272 (1992).

[2] 有田隆也, 曽和将容,"命令語再構成型プロセッサアーキテクチャ" , 電子情報通信学会論文誌, Vol. J76-D-I, No. 4, pp. 184–186 (1993).

[3] T. Arita, H. Takagi and M. Sowa, "V++: An Instruction-Restructurable Processor Architecture", Proceedings of the 27th Hawaii International Conference on System Sciences, Vol. I , pp.398–407 (1994).

[4] 加藤工明, 有田隆也, 曽和将容,"長命令語 (LIW) コンピュータにおける命令実行遅延方式", 電子情報通信学会論文誌, Vol.J74-D-I,No.9, pp.613–622 (1991).

[5] 高木浩光, 有田隆也, 曽和将容,"細粒度並列実行を支援する種々の静的順序制御方式の定量的評価" , 並列処理シンポジウム JSPP '91 論文集, pp.269–276 (1991).

[6] 原哲也, 久我守弘, 村上和彰, 富田眞治, "DSN 型スーパースカラ・プロセッサ・プロトタイプの分岐パイプライン", 情処研報, ARC-86-3 (1991).

[7] J. L. Hennessy and D. A. Patterson, " Computer Architecture - A Quantitative Approach" , Morgan Kaufmann Publishers (1990).

[8] J. E. Smith, "A Study of Branch Prediction Strategies", Proc. 8th Symp. on Computer Architecture, pp. 135-148 (1981).

[9] J, K. F. Lee and J. Hennessy, "Branch Prediction Strategies and Branch Target Buffer Design", IEEE Computer, Vol.17, No.1, pp.6–22 (1984).