計算機アーキテクチャ 133－2
（1999．5．21）

Cascade ALU を用いた命令実行手法の提案と評価

小沢基一† 中村 宏† 南谷 崇†

本稿では，非同期式スーパースカラプロセッサ TITAC-3 における命令実行機構の構成とその性能評価の結果について述べる．TITAC-3 の命令実行機構は，命令を in order 発行する場合でも限定された out of order 実行が実現できる Cascade ALU 方式を利用している．さらに，この Cascade ALU をパイプライン化することにより，性能の改善をおこなっている．この命令実行機構の性能を SPECint95 の命令トレースを用いたシミュレーションで評価した結果，4 命令まで同時発行できる場合に単純な Cascade ALU 方式と比較して 17 %，in order 発行を行うスーパースカラプロセッサと比較して，約 5 % の性能改善が得られることが分かった．

Instruction Execution Mechanism based on Cascade ALU

Motokazu Ozawa,† Hiroshi Nakamura and Takashi Nanya†

In this paper, we describe instruction execution mechanism of asynchronous super scalar processor TITAC-3 and present the performance evaluation results. The instruction execution is based on Cascade ALU. Cascade ALU realize restricted out of order execution though instructions are issued in order. The performance and resource efficiency of Cascade ALU is improved by Cascade ALU pipelining. Performance is evaluated based on instruction trace simulation of SPECint95. The evaluation reveals that our cascade ALU achieves 17 % better performance compared with normal Cascade ALU and 5 % better performance compared with in order issue super scalar processor.

1. はじめに

プロセッサの性能を高くする手法として，命令レベルの並列性を利用したアーキテクチャが広く用いられている．このようなアーキテクチャの代表的なものがスーパースカラである．スーパースカラでは，複数の演算器を用意することで複数命令を同時に実行する．

しかし，命令間にはさまざまな依存関係が存在するため，無条件に複数命令を同時実行できるわけではない．そこで，命令の出現順序を守らずに依存関係がない命令を捜すことで，同時実行可能な命令数を増加させる out of order 方式が広く用いられている．しかし，この方式を実現するには，レジスタリネーミング，リオーダバッファ，リザベーションステーションといった複雑な機構が必要となってしまう．

TITAC-3 では，設計を容易にするため命令の発行を出現順序通りに行うが，命令実行部として Cascade ALU を用いることで限定された out of order 実行を実現する．本稿では，このような Cascade ALU を用いた命令実行部の構成と，その性能をパイプライン化により向上させる手法を示す．また，SPECint95 の実行トレースを用いたシミュレーションでそれらの性能評価を行う．

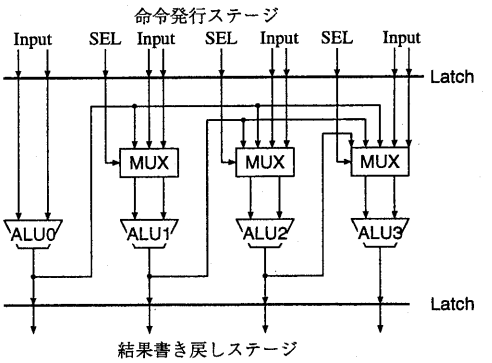

図1 Cascade ALU の構成

2. Cascade ALU の構成と動作

Cascade ALU[1] は，図1 のように各 ALU の出力をそれ以降の ALU の入力に接続したものである．各 ALU の入力には，入力オペランド (レジスタファイ

† 東京大学先端科学技術研究センター
Research Center for Advanced Science and Technology, The University of Tokyo

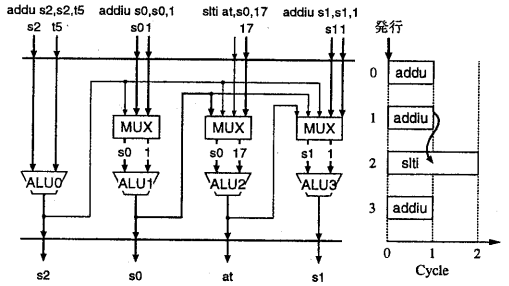

図2　RAW 関係がある命令の実行

ルから読み出す) とより左側にある ALU 出力のいずれかを選択するマルチプレクサを置く.

Cascade ALU での命令発行は，実行する命令を出現順序通りに左側の ALU から割り付ける．そのため，命令を実行する ALU は静的に決まる．命令を実行する ALU をこのように決定するため，マルチプレクサに入力する ALU 出力は自分より左側にあるものに限定される．また，命令発行は，同時発行する命令が必要とするレジスタ読み出しすべてが完了した時刻に行われる．

この Cascade ALU に必要な回路資源量は同時発行する命令数 n に対して次のように表せる．図1 の構成では $n = 4$ である．

ALU： n 個
MUX： 3 to 2, 4 to 2, $\cdots$, $n-1$ to 2 を 1 個ずつ
ALU 間配線： $n-1$ 本

2.1 RAW の処理

各 ALU 入力にあるマルチプレクサの動作は，その ALU に割り当てられた命令の入力オペランドが同時に発行された命令の実行結果に依存するかに応じて次のように決定される．

依存なし： 入力 (図1 の Input) を選択
依存あり： 依存する先行命令の出力を選択

マルチプレクサをこのように動作させることで，同時に発行された命令間に RAW 関係が存在しても，その RAW 関係を命令実行時に解決することができる．この場合，RAW 関係がある命令の実行が依存する命令すべての実行が終了する時刻から始まる．このような動作の例を図2 に示す．この場合，`slti at,s0,17` が `addiu s0,s0,1` の結果に依存しているため，`slti at,s0,17` を実行する ALU2 の入力 MUX が ALU1 の出力を選択している．

2.2 WAW の処理

同時に発行される命令の間には，結果の書き込み先が同じ命令が含まれる可能性がある．このような WAW 関係を持つ命令の結果を後続命令で利用する場合や，レジスタに書き戻す場合には，対応するレジスタへ結果を書き込む ALU のうち，最も右側の ALU から出力される結果を使う必要がある．なお，RAW

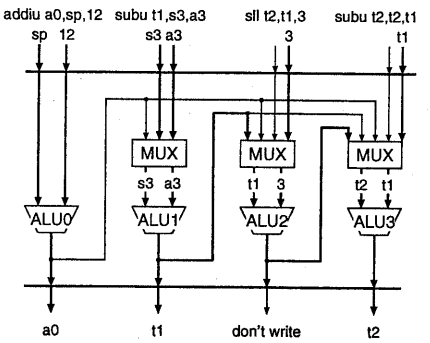

図3　WAW 関係がある命令の実行

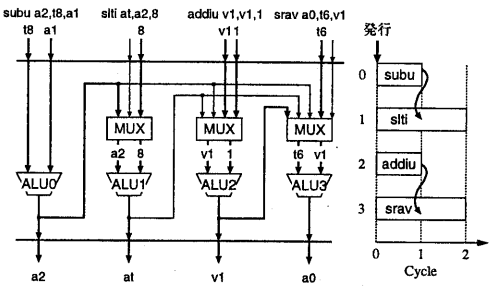

図4　命令の out of order 実行

関係の場合と異なり，WAW 関係を持つ命令の実行開始は遅延されない．

このようにすることで，通常の ALU を持ったスーパースカラではレジスタリネーミングを行わない限り解決できない WAW 関係を Cascade ALU 内部で実行時に解決することができる．このような動作の例を図3 に示す．この場合，`sll t2,t1,3` と `subu t2,t2,t1` の間に `t2` に関する WAW 関係がある．そのため，`sll t2,t1,3` の結果はレジスタに書き戻さない．しかし，この命令の結果を `subu t2,t2,t1` が使うため，`sll t2,t1,3` の実行自体は必要である．

なお，分岐予測を用いて分岐命令を越えた投機実行を行う場合や例外処理を行う場合には，書き戻しが不要な結果も保存する必要がある．

2.3 命令の out of order 実行

Cascade ALU で命令を実行することで，同時に発行しようとする命令内に閉じた RAW, WAW 関係を実行時に解決することができる．その結果，すべての命令を同一時刻に発行でき，同時に発行された命令内に閉じた out of order 実行を実現できる．なお，同時発行する命令が必要とするレジスタ読み出しはすべて命令発行時に終了しているため，WAR は発生しない．

このような動作が行われる例を図4 に示す．この場合，`subu a2,t3,a1` と `slti at,a2,8` の間，`addiu v1,v1,1` と `srav a0,t5,v1` の間に RAW 関係があ

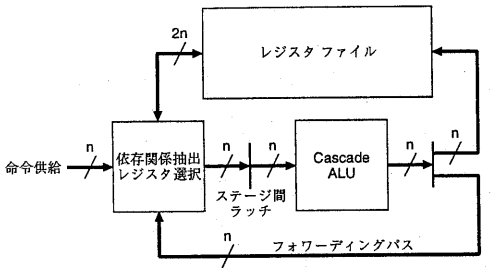

図5　Cascade ALU による命令実行機構の基本構成

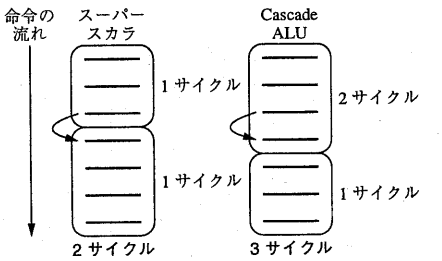

図6　Cascade ALU による性能低下

る．ALU の演算に 1 Cycle が必要と考えると，Cascade ALU では 2 Cycle で命令実行が終了する．しかし，通常の in order 発行スーパースカラでは，

```
1:  subu a2,t3,a1
2:  slti at,a2,8    addiu v1,v1,1
3:  srav a0,t5,v1
```

と実行されるため，3 Cycle が必要となる．この 4 命令内で out of order 実行を行えば，

```
1:  subu a2,t3,a1   addiu v1,v1,1
2:  slti at,a2,8    srav a0,t5,v1
```

のように実行されるため，Cascade ALU と同じ 2 Cycle で実行が終了する．

2.4　命令実行機構の構成

Cascade ALU を用いた場合の命令実行機構の基本的な構成を図5 に示す．WAW や RAW により命令発行を停止する必要がないため，全体の構成が単純になっている．Cascade ALU では，命令の終了が out of order となるが，実行中の全命令が終了しないと次の命令を投入できないため，発行された命令すべての実行が終了した時にフォワーディングやレジスタへの書き戻しを行う．なお，書き戻しの際には投機実行の状態を考慮する必要がある．

また，この構成では Cascade ALU ステージの遅延が一定にならない．その結果，通常のクロックに同期してデータを転送する構成では動作しない．そのため，各ステージで処理の終了を検出し，非同期にデータ転送を行うことが必要となる．

2.5　利点と欠点の考察

Cascade ALU による命令実行を行うことで，レジスタリネーミングを行わず，静的に演算器を割り当てる in order 命令発行を行っていても，次のような動作が実現できる．

- 同時に発行された命令内に閉じた WAW の考慮が不要
- 同時に発行された命令内に閉じた out of order 実行が可能

しかし，これらの特徴は同時に発行された命令内に限られたものであるため，同時に発行する命令幅をある程度大きくする必要がある．この結果，回路量が増加してしまうが，回路構造が通常の out of order 実行を行う構成より単純であるため，実装の最適化をしやすいと考えられる．さらに，非同期式回路による実装を行うことで，演算器のカスケード接続による性能向上[1] も利点となる．

その一方，通常のスーパースカラ向けに最適化されたコードを Cascade ALU により実行すると，通常のスーパースカラより性能が低下することがある．Cascade ALU では，同時に発行する命令内の依存関係は無視する．しかし，通常のスーパースカラでは依存関係があると命令の発行をその手前で中止する．その結果，依存関係の存在により命令発行が複数サイクルに分割されることを利用したコードを実行すると Cascade ALU の性能が通常のスーパースカラより低くなる．この様子を図6 に示す．なお，この図中の矢印は命令間の依存関係を示している．

3.　Cascade ALU の高性能化

Cascade ALU では，すでに発行されている命令と依存関係がない命令を発行する場合であっても，すでに発行された命令の実行がすべて終了するまで待つ必要がある．このような動作の例を図7 に示す．なお，この図の命令 1,2,3,4 と命令5,6,7,8 には依存関係がないとする．

このため，すでに発行されている命令と依存関係がない命令をすでに発行されている命令の実行終了を待たずに発行することで Cascade ALU の性能を高くで

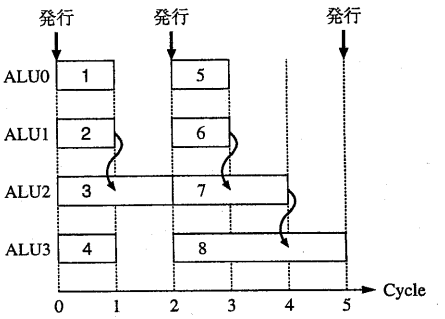

図7　Cascade ALU の命令発行動作

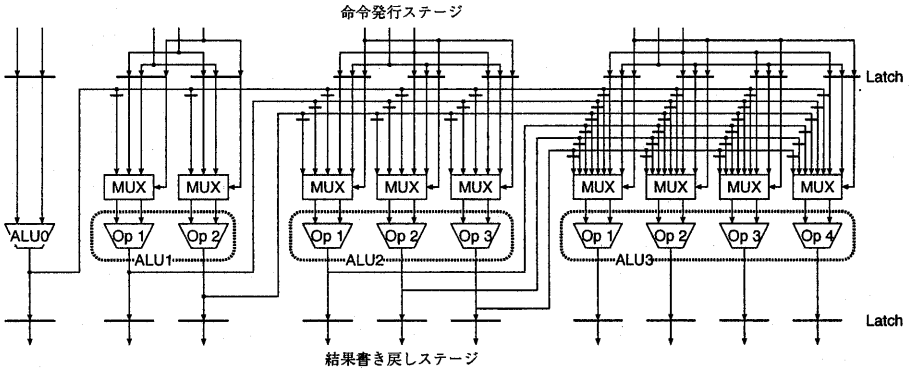

図9 パイプライン化した Cascade ALU の構成

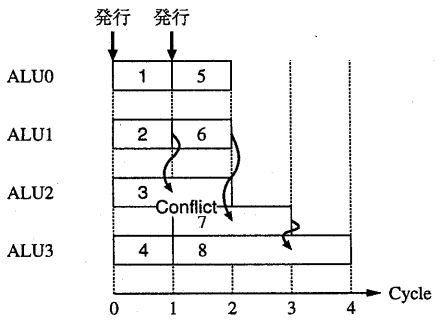

図8 Cascade ALU の高性能化

きる．

しかし，Cascade ALU では，同時に発行された命令の間に RAW 関係により，各 ALU の演算終了時刻が変化する．その結果，図8 の ALU2 のように，複数の命令が同一時刻に同じ ALU を使用することがある．この状態を避けるには，ALU を同時に実行されうる最大命令数だけ用意すれば良いが，同時発行する命令数が 4 である場合にも 10 ALU (1+2+3+4) が必要となり，現実的ではない．

そこで，高速な ALU の実装では，各演算機能 (加算，減算，シフトなど) がそれぞれ独立した演算器で構成されていることに着目し，各 ALU の演算機能に対して独立に命令発行を行う．このようにすることで，すでに発行されている命令が利用している演算器を発行しようとする命令が利用しない場合に先行命令の終了を待つこと無く命令を発行できる．このような命令発行を実現するための構成を図9 に示す．この図のような構成をパイプライン化 Cascade ALU と呼ぶ．この図で右側の ALU ほど演算器の種類が多くなっているのは，右側の ALU ほど発行から実行終了までの時間が長くなることがあり，演算器の衝突を起こしやすいためである．逆に最も左側の ALU については，演算器の衝突がないため，機能別の分割は不要である．

一般に n 命令を同時発行できるパイプライン化 Cascade ALU を実現するには，次のような回路資源が必要となる．

演算器： ALU n 個分

MUX： 3 to 2 を 2 個，5 to 2 を 3 個，$\cdots$，$(n(n-1)/2+2)$ to 2 を n 個

ALU 間配線： $n(n-1)/2$ 本．

3.1 命令発行と命令実行の動作

パイプライン化した Cascade ALU における命令発行の動作は次のようになる．この動作から，命令発行は同時に発行するすべての命令で (1), (2) が満たされた時刻に行われる．

(1) 発行しようとする命令すべてが読み書きするレジスタに，実行中の命令の書き込み先レジスタが含まれている場合，そのレジスタの書き込みが終了するまで待つ．

(2) 発行しようとする命令すべてについて，命令が使う演算器に対応するラッチが書き込み可能になるまで待つ．

(3) 各命令が行う演算に対応した入力ラッチにレジスタから読み出したオペランドと MUX の選択入力を書き込み，入力ラッチに対する書き込みを禁止する．

命令実行の動作は，各演算器で独立に行われる．その動作は次のようになる．RAW や WAW の処理は Cascade ALU の場合と同じである．

(1) 必要な入力が揃ったら演算を開始し，結果をラッチに書き込む．

(2) 対応する入力ラッチの書き込み禁止を解除する．

命令発行動作の (1) を行うには，レジスタの書き込み予約を行う機構が必要となる．また，命令実行動作が各演算器で独立に行われるので，フォワーディングやレジスタ書き込みを演算器ごとに制御する必要がある．この結果，パイプライン化した Cascade ALU による命令実行機構の基本構成は図10 のようになる．

この動作に基づいて，次のような命令列を実行する

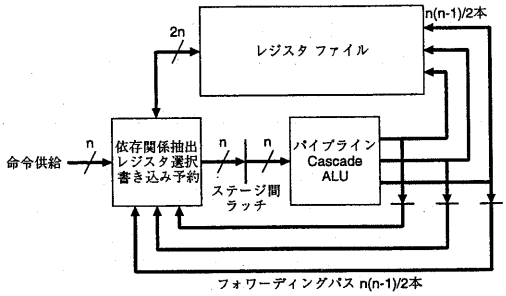

図10　命令実行機構の基本構成

場合の動作を図11 に示す.

```
1:  or      a1,a2,1
2:  sll     t6,a1,2
3:  addu    t7,a0,t6
4:  sw      t7,96(s4)
5:  sll     a1,a1,2
6:  addiu   s0,s0,1
7:  subu    s1,s0,17
8:  addiu   a1,a1,1
```

命令 5,6,7,8 を発行する際の動作を述べる．まず，命令 1,2,3,4 の書き込み先レジスタ a1,t6,t7 が命令 5,6,7,8 で利用されているかを調べる．この場合，命令 1 で生成される a1 を命令 5 が利用している．そのため，命令 1 の実行が終了する時刻 1 以降でないと命令 5,6,7,8 を発行できない．

次に，命令 5,6,7,8 が利用する演算器について調べる．レジスタに関する発行条件が満たされる時刻 1 では，命令 5,6,7,8 の利用する演算器がすべて空いている．この結果，時刻1 で命令 5,6,7,8 の発行を行うことができる．

この例で ALU2 の sub と add が 1 つの演算器にまとめられている場合を考える．この場合，命令 3,7 の演算器が衝突する．その結果，図12 のように，命令 5,6,7,8 の発行を命令 3 の実行終了までストールさせる．

図11 から，パイプライン化 Cascade ALU を用いて命令実行を行うと，同時に発行された命令のみでなく，すでに発行されている命令に対しても out of order 実行が実現されることがある．その結果，通常の in order 発行スーパースカラでは5 Cycle，通常の Cascade ALU では 6 Cycle が必要な命令実行を 4 Cycle で実現できている．

図11 の実行終了順序に着目すると $1 \rightarrow 2,5,7,8 \rightarrow 3 \rightarrow 4$ のようになっており，実行終了の順序が命令発行の順序と異なっている．このため，投機実行を行う場合の回復処理が Cascade ALU より複雑になる．また，load と store が同時に発行されている場合，その順序を保持するため，アドレスが確定するまで実際のメモリアクセスを遅延する必要がある．

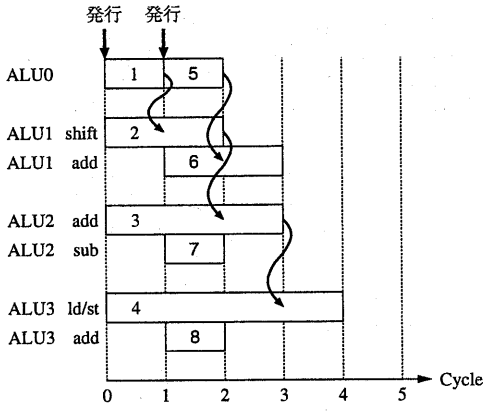

図11　命令の実行例

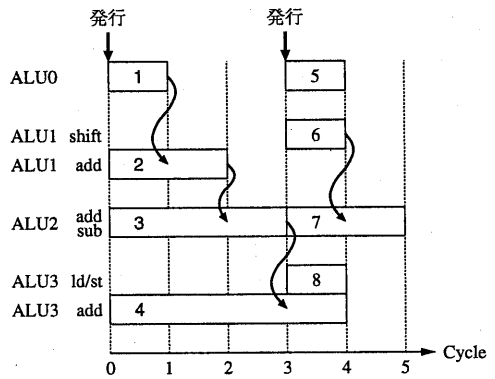

図12　命令の実行例 (演算器が衝突する場合)

3.2　利点と欠点の考察

パイプライン化した Cascade ALU により，Cascade ALU とほとんど同じ命令発行方式のままでより高い性能を得ることができる．パイプライン化した Cascade ALU で性能が向上するのは同時に発行する命令グループ間に依存関係や資源の衝突がない場合である．

また，Cascade ALU では性能が低下してしまう，図6 のような命令列を実行する場合でも，演算資源が衝突しなければスーパースカラと同じ性能を達成できることも利点である．

しかし，パイプライン化した Cascade ALU では，ALU の入力を選択する MUX，ラッチ，演算器間の配線が増加したり，命令実行機構自体の構成が複雑になってしまう．

なお，Cascade ALU では非同期式で実現することで演算遅延が削減できることも利点である．パイプライン化 Cascade ALU の各 ALU 間配線にはラッチが挿入されているが，演算遅延が削減されるための動作条件は満たしている．

表1　命令発行の条件

同時発行可能な命令数	4	分岐予測のミス率	5 %	乗除算命令のペナルティ	5 Cycle
同時発行可能な load/store 命令数	2	分岐予測のミスペナルティ	3 Cycle	演算器出力からフォワーディング	
同時発行可能な分岐命令数	1	データキャッシュのミス率	5 %	ALU の内部は add, cond, logic, mem	
同時発行可能な乗除算数	1	データキャッシュのミスペナルティ	5 Cycle	muldiv, shift, sub の 7 演算に分割	

表2　IPC の評価結果

命令の実行方式	IPC
in order 発行スーパースカラ	1.93
Cascade ALU	1.76
パイプライン化 Cascade ALU	2.06

4. 性能評価

Cascade ALU のパイプライン化による効果を確認するために，通常の in order スーパースカラ，Cascade ALU，パイプライン化 Cascade ALU のそれぞれについて命令実行のシミュレーションを行い，その結果から性能評価を行う．

4.1 評価条件

ベンチマークとして dhrystone, libmpeg, SPECint95 を用いた．評価には MIPS I 用にコンパイル (MIPS cc，オプション -O2) したベンチマークを SimOS 上で動作させ，その結果生成された実行トレースを用いている．なお，シミュレーションで用いた命令数は dhrystone が約 6 万，その他のベンチマークが約 100 万命令である．

シミュレーションを行う際の命令発行条件は表 1 のように設定した．なお，演算器の遅延は 1 Cycle (カスケードしても変化しないと仮定)，命令供給時のストールはないとし，同時発行できなかった命令については，発行を 1 Cycle 遅延し，その後続命令と合わせることで，可能な限り命令発行幅を大きくしてから発行している．

4.2 IPC の評価

それぞれの命令実行方式に対して，IPC (Instructions Per Cycle) を求めた結果を表 2 に示す．

この評価結果から，パイプライン化 Cascade ALU の性能は，in order 発行のスーパースカラより 5 %，Cascade ALU より 17 % 向上している．

Cascade ALU より in order 発行のスーパースカラの性能が良いため，図 6 のような状態が多いと考えられる．一方，パイプライン化 Cascade ALU では，資源が衝突しない限り，このような性能低下が起こらないため，限定された out of order 実行による性能向上が得られ，in order 発行スーパースカラより性能が高くなったと考えられる．

4.3 ストール原因の評価

今回の評価では，パイプライン化 Cascade ALU の各 ALU が 7 種類に分割されている．しかし，実際に 7 分割を行うのは配線量，回路量の観点から望ましくない．しかし，分割数を小さくすると演算器の衝突が増加し，性能が低下する．そこで，命令発行時におきるストールの原因を調べた．その結果を表 3 に示す．

表3　ストール原因

ストール原因	頻度	(%)	サイクル	(%)
なし	1681630	(64.8)	0	(0.00)
レジスタ依存	535062	(20.6)	1034492	(61.4)
mem	194993	(7.51)	390999	(23.2)
add	73497	(2.83)	102459	(6.08)
cond	74648	(2.88)	95541	(5.67)
shift	22009	(0.85)	32041	(1.90)
logic	10312	(0.40)	25098	(1.49)
sub	3006	(0.12)	4354	(0.26)
muldiv	121	(0.01)	621	(0.04)
合計	2595278	(100)	1685605	(100)

この結果，ストールサイクルの 60% がレジスタ依存の解決待ち，20 % がデータキャッシュアクセス待ちによることがわかる．このうち，データキャッシュを 100 % ヒットと仮定するとデータキャッシュによるストールサイクル数が約 1/2 になるため，データキャッシュによるストールはキャッシュミスによるものが支配的である．

このように，演算器衝突が原因でストールするサイクル数は比較的小さい．その結果，ALU の分割数を少なくしてもそれほど大きな性能低下はおきないと考えられる．

5. まとめ

命令を in order に発行し，命令を実行する演算器を静的に割り付ける場合であっても限定された out of order 実行を実現できる Cascade ALU 方式と，その性能を向上させるパイプライン化 Cascade ALU 方式について述べた．また，命令トレースを用いたシミュレーションにより，パイプライン化 Cascade ALU の性能が，in order 発行のスーパースカラに対して 5 %，Cascade ALU に対して 17 % 高くなることが分かった．

今後の課題として，性能や回路量の定量的な考察，フォワーディングやレジスタファイルの構成，投機実行や例外からの回復処理の検討がある．

なお，本研究の一部は科学研究費補助金（基盤研究（B）09480049）および（株）半導体理工学研究センターとの共同研究に基づくものである。

参考文献

1) 上野洋一郎, 深作泉, 南谷崇: 非同期式カスケード ALU アーキテクチャ, 信学技報 ICD98–9 (1998).