非同期式プロセッサ TITAC-3 の命令実行機構

2H－4

小沢基一　中村 宏　南谷 崇
東京大学 先端科学技術研究センター

1 はじめに

近年のデバイス技術の進歩により，スイッチング遅延が数 ps の素子が実現されつつある．その一方で，要求される搭載機能が増加した結果，デザインルールが縮小したにもかかわらずチップ面積が増大している．この結果，従来のプロセスと比べ，配線遅延が相対的にも絶対的にも増加し，チップ全体を単一クロックに同期させる同期式システムではクロックスキューにより性能が制約されてしまう．

この問題を解決するための手法として非同期式システムが注目されている．非同期式では全体を同期させるクロックを用いず，信号遷移の因果関係のみを用いた自律的な動作でシステムを実現する．そのため，システム全体に分配するクロック信号が不要となり，素子の高速性を活かすことが可能になる．

このような非同期式システムで汎用プロセッサを実現した例がいくつか存在するが，いずれも単純なスカラプロセッサであり，現行のプロセッサが行うような複数命令の並列実行を行っていない．そこで，本稿では，複数命令の同時実行を行う非同期式プロセッサ TITAC-3 の命令実行手法を示し，その基本性能をシミュレーションで評価する．

2 非同期式プロセッサ TITAC-3 の概要

非同期式プロセッサ TITAC-3 の特徴として，次のようなことがある．

- MIPS I 互換 (FP を除く) な命令セット
- パイプライン化 Cascade ALU による命令の並列実行
- 分岐予測機構・投機実行機構の搭載
- 命令・データキャッシュの搭載
- 同期式 SRAM による主記憶の実現

また，回路実装上の特徴として，次のようなことがある．

- DCVSL を利用したセルレベルパイプライン [1]
- パイプライン段数を多くして制御遅延を隠蔽 [2]

3 命令の実行方式

3.1 Cascade ALU 方式

TITAC-3 の命令実行方式として Cascade ALU 方式 [3] を用いる．Cascade ALU 方式では，各 ALU の出力をそれ以降の ALU の入力に接続する．その結果，同時に発行する命令内に RAW, WAW 関係が存在しても，全命令を同一時刻に発行することができる．

図3(i) は ALU 数 3 の場合に次のような命令列を実行するタイミングを示している．(図3 では，ALU の演算遅延を 1 cycle と考えている) また，図1 は，命令 4,5,6 について Cascade ALU 内のデータの流れを示す．

```
1:  andi   v1,v0,0xf
2:  addiu  a3,v1,2
3:  subu   t5,zero,a3
4:  addiu  s3,s3,1
5:  sll    t7,s3,2
6:  addu   s3,s2,s3
```

この場合，(4,5) (4,6) の間に RAW，(4,6) の間に WAW が存在する．そのため，4 の出力を 5 と 6 に接続し 4 の出力をレジスタに書き込まないようにする．

Instruction Execution Mechanism for Asynchronous Processor TITAC-3
Motokazu Ozawa, Hiroshi Nakamura, Takashi Nanya
University of Tokyo, Research Center for Advanced Science and Technology

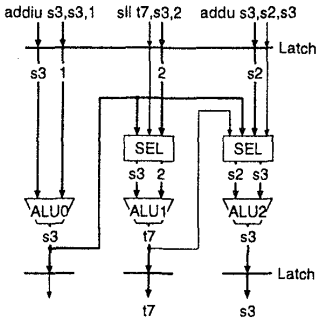


図 1: Cascade ALU 方式

3.2 Cascade ALU のパイプライン化

Cascade ALU 方式の性能を向上させる手法として，Cascade ALU 全体をパイプライン化し，実行中の命令が終了しない状態での命令投入を可能にすることが考えられる．しかし，Cascade ALU 方式では，同時に発行された命令の持つ依存関係により各 ALU で演算の行われる時刻が遅くなることがある．そのため，複数命令が同一時刻に同じ ALU を使用することが起こる．この状態を避けるには，ALU を同時に実行される最大命令数だけ用意すれば良いが，回路量やその結果得られる性能の点で現実的でない．

そこで，ALU の各機能 (加算，減算，論理演算など) がそれぞれ独立した演算器で構成されることに着目し，ALU の各機能を独立に動作可能にすることを考える．この場合，配線量が増加してしまうが，使いたい演算器が使用されていない場合，先行命令の終了を待つこと無く命令を発行できるようになり，Cascade ALU のパイプライン化が可能になる．このようにパイプライン化した Cascade ALU を図2 に示す．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 ほど演算器が衝突しやすいためである．

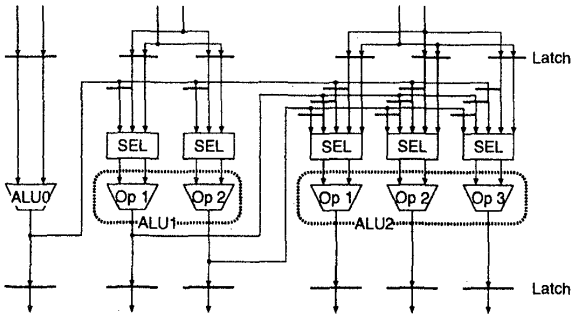
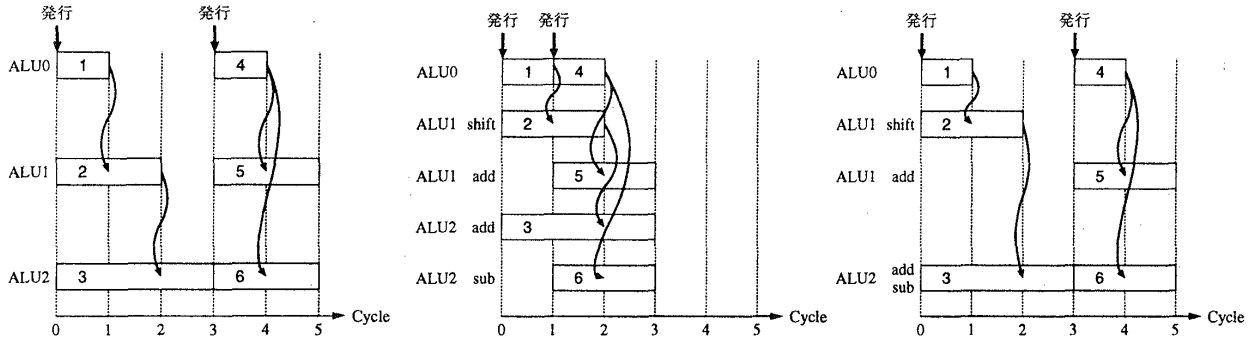


図 2: パイプライン化した Cascade ALU

パイプライン化した場合の動作は次のようになる．

- 命令を行う演算に対応した入力部のラッチに書き込む．このラッチへの書き込みは対応する演算の結果が出力されるまで禁止される．
- RAW 関係がある場合，必要な結果の出力まで待つ．
- 入力が揃ったら演算を行い，結果をラッチに書き込む．

正しく命令を実行するための命令発行条件として，次の条件がある．



(i) パイプライン化しない場合　(ii) パイプライン化した場合　(iii) 演算器が衝突する場合

図 3: 命令実行の様子

表 1: 命令発行の条件

同時発行可能な命令数	4 命令	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 命令	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 命令	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命令数	1 命令	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	

1. 発行する命令が読み書きするレジスタに，実行中の命令の書き込みレジスタが含まれない．
2. 発行する命令の利用する演算器が使われていない．

この条件で 3.1 節の命令列を実行する場合を考える．この場合，命令 1, 2, 3 と命令 4, 5, 6 の間で共通に使われるレジスタが存在しない．また，命令 1, 4，命令 2, 5，命令 3, 6 の間で演算器の衝突がない．これは前述した条件を満たすため，図 3(ii) のように命令が処理される．

次に，ALU2 の sub と add を 1 つの演算器にした場合を考える．この場合，命令 3, 6 の演算器が衝突してしまう．その結果，図 3(iii) のように，命令 4, 5, 6 の発行を命令 3 の実行終了までストールさせることになる．

この結果，パイプライン化した Cascade ALU の性能には，同時に発行する命令内の依存関係のみでなく，同時に発行する命令間の依存関係や演算器の衝突も影響することになる．

また，図 3(ii) の実行終了順序を見ると，(1) → (2, 4) → (3, 5, 6) のように発行順序と異なった順序になる．そのため，パイプライン化した Cascade ALU で投機実行を行う場合には注意が必要となる．また，load と store が同時に発行された場合，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．

4 命令実行部の評価

4.1 評価条件

ベンチマークとして，dhrystone, libmpeg, SPECint95 の compress, gcc, go, ksim, ijpeg, li, perl を用いた．評価には MIPS I 用にコンパイル (MIPS cc を使用) したベンチマークを SimOS 上で動作させて生成した実行トレースを用いた．なお，実行トレースに含まれる命令数は dhrystone が約 60k，その他のベンチマークが約 1M 命令である．

命令実行部のシミュレーションでは，同時発行の条件を表 1 のように設定した．なお，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て発行した．

4.2 評価結果

表 2 に実行に必要なサイクル数から求めた Instruction Per Cycle (IPC) を示す．この結果から，パイプライン化により性能が約 20 % 向上することがわかる．

表 3 に命令発行時のストールサイクル数を決定した要因とその頻度を示す．この結果から，ストールサイクルの 50%

表 2: IPC の評価結果

パイプライン化なし	パイプライン化あり
1.70	2.05

以上がレジスタ依存の解決待ちによるものであることがわかる．Cascade ALU をパイプライン化する場合，配線量の増加が問題となるため，ALU の分割数を少なくする必要があるが，この結果から，ALU の分割数を少なくしても性能低下はそれほど大きくならないと考えられる．

表 3: ストールサイクルの内訳

ストール原因	頻度 (%)		サイクル (%)	
なし	1478673	(65.2)	0	(0.00)
レジスタ依存	447060	(19.7)	792724	(56.7)
mem	189213	(8.34)	401701	(28.8)
add	83887	(3.70)	102994	(7.37)
sub	24557	(1.08)	31488	(2.25)
cond	18733	(0.84)	24054	(1.72)
shift	17105	(0.75)	25777	(1.85)
logic	8822	(0.39)	18148	(1.30)
muldiv	25	(0.00)	117	(0.01)
合計	2268075	(100)	1397003	(100)

5 まとめ

Cascade ALU をパイプライン化する手法を示し，実行トレースによる性能評価を行った．その結果，レジスタリネーミングなどを行わない単純な命令発行方式でもパイプライン化しないものと比べ，性能が約 20 % 向上することがわかった．

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よるものである。

参考文献

[1] 今井雅, 中村宏, 南谷崇. DCVSL を利用した非同期式細粒度パイプライン・データパスの論理合成. 情処研報, September 1998.

[2] 小沢基一, 高村明裕, 上野洋一郎, 中村宏, 南谷崇. 非同期式パイプラインプロセッサの高性能化手法について. 情報処理学会第 56 回全国大会, March 1998.

[3] 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, April 1998.