M-042

CHAIN 手法と RENAME 手法を混用するスーパースカラプロセッサの設計と評価

A design and evaluation of the Superscalar Processor which uses RENAME technique together with CHAIN technique

孟　　林†　　小柳　滋†

Lin　Meng　　Shigeru Oyanagi

1. はじめに

本研究では、スーパースカラプロセッサの構築を行っている。スーパースカラプロセッサでは、複数のパイプラインで並列に実行する。そのため、データの競合確率が高くなり、この競合による命令発行を制約し性能も低下する。本論文では、ハザードのことを考え、よりスケジューリングをし易い命令列を作るため、RENAME と CHAIN を混用する手法を提案し、検証する。

2. プロセッサの概要

2.1 スーパースカラプロセッサ

本プロセッサは、MIPSのパイプラインプロセッサのベースで構築されている。本プロセッサは二つのパイプラインを持ち、主に命令フェッチ（F）、命令のRENAME(R)、命令のCHAIN（CH）、命令のScheduling(SH)、命令ディコードとレジスタファイル読み出し（D）、演算とアドレス生成（E）、メモリアクセスとレジスタ書き込み（M）、レジスタ書き込み（W）の8部分で構成されている。演算とアドレス生成部（E）は、ロードストアユニット、分岐ユニット、シフトユニットと二つの算術演算ユニットで構成されている。図２はスーパースカラプロセッサのブロック図を示している。

2.3 ハザード

ハザードの対応はスーパースカラプロセッサの効率に対して、非常に重要である。本プロセッサは、スケジューリング前に、ハザードのことを考慮し、RENAME手法と CHAIN 手法を混用して対応する。最初の二つの同じタイミングに発行しようとする命令両方が代入命令で、かつその間にデータハザードが発生する場合を CHAIN 手法で対応し、異なるタイミングで発行しようとする命令間によるデータハザードを RENAME 手法で対応する。その二つの手法を混用することにより、命令のスケジューリングを効率的に行い、ハザードを減らす。

3. RENAME と CHAIN によるハザードの対応

3.1 RENAME 手法及びそのハザード対応

RENAME 手法とは、レジスタの名前を書き換えることによって命令を効率に発行し、ハザードを解消する技術である。図３は RENAME の例を示している。

図 3 の RENAME 前の命令列から見ると、In5 と In6 の間に$s2 に関するデータの競合が発生する。そのデータの競合を解消するために、In5 を In4 より先に実行する必要がある。しかし、In5 が In4 の先に実行すると、In5 の命令により$s2 の値を更新してしまう。そのため、In4 には正しい値を得ることができない。従ってスケジューリングを使えず、ストールしか解決できない

ここで、In4 と In5 が同じ$s2 を使っているが、値が異なるという特徴がある。この特徴を利用し、In2 と

† 立命館大学, Ritsumeikan University

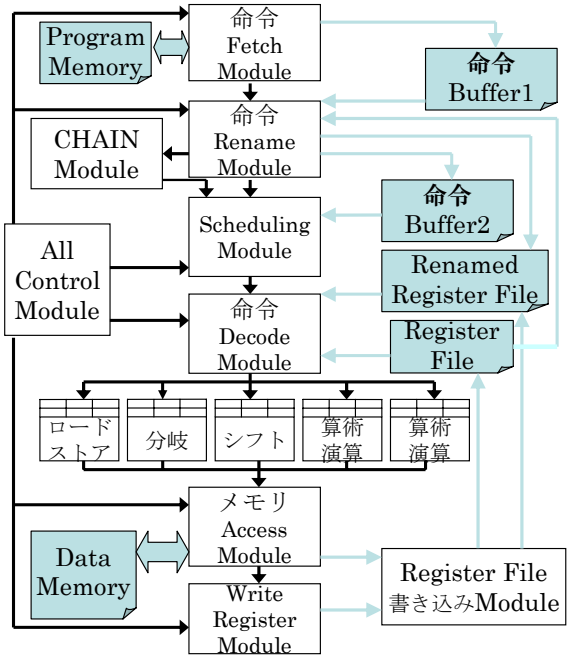

図２　プロセッサブロック図

In4 の命令の$s2 の NAME を換えると、In5 の実行結果と In4 のデータの競合関係がなくなり、スケジューリングができる。図 3 の RENAME 後 Scheduling された命令列に示すように、本来ストールしか対応できない In5 と In6 の間のデータ競合が RENAME により解消した。

```
In1: Mov $s1 100          In1: Mov $s1 100
In2: Sub $s2 $s3 $s5      In5: Add $s2 $s5 $s3
In3: Sub $s7 $s6 $s3      In2: Sub $s12 $s3 $s5
In4: Sub $s6 $s4 $s2      In3: Sub $s7 $s6 $s3
In5: Add $s2 $s5 $s3      In4: Sub $s6 $s4 $s12
In6: Beq $s4 $s2 $s3      In6: Beq $s4 $s2 $s
*RENAME 前の命令列        *RENAME 後 Scheduling された命令列
*$s2 RENAME 可能レジスタ  *$s12 RENAME されたレジスタ
Sub: '-', Move: 値の代入, Beq: if(=) goto,
Add: '+',    In: 命令
```

図３　RENAME 例

3.2 CHAIN 手法によるハザード対応

CHAIN 手法とは、最初の二つの命令が同時に発行され、生成されたデータをすぐに計算することにより、データハザードを解消する方法である。

```
In1 Add  $s1 $s4 $s5
In2 Sub  $s2 $s1 $s6      (CHAIN 手法の命令例)
```

具体的には、上の例からみると、In1, In2 を同じタイミングに実行するとき、二つ命令の間に$s1 に関するデータ競合が発生する。このとき、CHAIN 手法による制御信号を作り、算術演算モジュールに渡す。そして制御信号によって、まず一つ目の演算ユニットを用いて In1 を演算させ、次にその値を瞬時に二つ目の演算ユニットに渡し、In2 を実行する。その後、二つの演算が終わって

から、結果をレジスタファイルに書き込む。この手法で、データ競合により、In1 と In2 のように、本来並列に実行できない命令列を並列に実行することができる。

3.2 RENAME と CHAIN の流れ

RENAME と CHAIN は命令列に関する垂直スキャンと水平スキャンの結果によって決められる。図 3 が RENAME と CHAIN の流れを示している。

表 1.命令の構成表

	OP1	OP2	OP3
命令 1	a	b	c
命令 2	d	e	f
命令 3	g	h	i
命令 4	j	k	l
命令 5	m	n	o
命令 6	p	q	r

表 1.は各命令の構成表で、水平方向が各命令のオペランドを示し、垂直方向が命令列を示す。RENAME と CHAIN の動作について、命令 1 の第一オペランドをベースオペランドとして、例を挙げる。

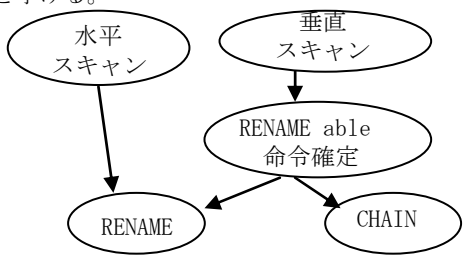

図 3. RENAME と CHAIN の流れ

垂直スキャンではベースオペランドと一致する第一オペランドを持つ代入命令を探す。表 2 は命令 1 の OP1 による垂直スキャン表を表す。これにより RENAME 可能な命令を確定する。垂直スキャン欄の a:x は、指定されているオペランドとベースオペランド(a) を比較することを示す。もし a と x が一致し、x に対応する命令が代入命令であるなら、この命令の垂直スキャン欄を 1 にする。その結果に基ついてベースの命令から最初の 1 が出現する間の命令を全部 RENAME 可能な命令として確定し、RENAME 可能欄の値を 1 にする。

表 2. 命令 1 の OP1 による垂直スキャン表

	垂直スキャン	RENAME 可能
命令 1	--	RENAME 可能条件 (RIns1)
命令 2	a:d & 命令 2 が代入命令	同上(Rins2)
命令 3	a:g & 命令 3 が代入命令	同上(Rins3)
命令 4	a:j & 命令 4 が代入命令	同上(Rins4)
命令 5	a:m & 命令 5 が代入命令	同上(Rins5)
命令 6	a:p & 命令 6 が代入命令	同上(Rins6)

*RENAME 可能条件(値を 1 にする条件) :垂直スキャン結果の中に 1 が存在する＆上位の命令の垂直スキャンの結果に 1 がない

水平スキャンはオペランドをスキャンすることによって、RENAME 可能なオペランドを探す。表 3 は命令 1 の OP1 をベースオペランドとしたときの水平スキャン表を示している。ベースオペランドをスキャンするときに、値を 1 にする。第一オペランドがスキャンされたときに、ベースオペランド(a)とスキャンされた命令の第一オペランドを比較し、その二つのオペランドが同一、かつ命令が代入命令である場合は、水平スキャン結果の値を 1 にする。第二第三オペランドをスキャンされたときに、ベースオペランドと比較し、二つのオペランドが同じであるなら、値を 1 にする。

表 3. 命令 1 の OP1 による水平スキャン表

	水平スキャン		
	OP1	OP2	OP3
命令 1	1	--	--
命令 2	a:d & 命令 2 が非代入命令(Rop2)	a:e	a:f
命令 3	a:g & 命令 3 が非代入命令(Rop3)	a:h	a:i
命令 4	a:j & 命令 4 が非代入命令(Rop4)	a:k	a:l
命令 5	a:m & 命令 5 が非代入命令(Rop5)	a:n	a:o
命令 6	a:p & 命令 6 が非代入命令(Rop6)	a:q	a:r

水平スキャンと RENAME 可能命令確定後に CHAIN と RENAME を行う。先ず CHAIN の可能性を調べ、もし表 2 の命令 2 の垂直スキャン欄 (a:d & 命令 2 が代入命令) が 1 になると、最初の二つの命令を CHAIN し、RENAME かどうかの判断は行わない。

CHAIN を行わないとき、表 2 の中の Rins1～Rins6 が 0 になる場合は RENAME 可能な命令がないため、RENAME を行わない。そうでないときに、RENAME 条件表を生成して RENAME を行う。RENAME 条件表を表 4 に示す。表の各欄はオペランドの RENAME が必要な条件を示し、プロセッサがこの条件に基づいて RENAME を行う。

表 4. 命令 1 の OP1 による RENAME 条件表

	RENAME		
	OP1	OP2	OP3
命令 1	1 & Rins1	--	--
命令 2	Rop2 & Rins2	a:e & Rins2	a:f & Rins2
命令 3	Rop3 & Rins3	a:h & Rins3	a:I & Rins3
命令 4	Rop4 & Rins4	a:k & Rins4	a:l & Rins4
命令 5	Rop5 & Rins5	a:n & RIns5	a:o & RIns5
命令 6	Rop6 & RIns6	a:q & RIns6	a:r & RIns6

4. 評価と検証

本提案を Xilinx 社の Spartan3E ディバイスファミリを用いて評価した。RENAME と CHAIN モジュールには Slices を 248, 4 input LUTs を 436, FFs を 0 個使用している。

1 から 100 までの総和 (sum100) と 1 から 100 までの 2 倍の総和(sumd100)の計算を使って検証した。検証の結果を表 5 に示す。検証の結果から見ると、RENAME と CHAIN 手法によるハザードを減らすことができる。

表 5. 検証結果表

	RENAME によるハザード処理できた数	CHAIN によるハザード処理できた数
Sun100	100	0
Sund100	100	100

5. おわりに

本提案で、RENAME によるレジスタの名前を変えて、より効率よいスケジューリングができることによりハザードを減らすことができ、CHAIN 手法より本来並列に実行できない命令列の並列実行ができた。今後の課題としては、CHAIN 手法を条件分岐で応用すること、スケジューリング時の CHAIN 手法の追加、RENAME アルゴリズムの最適化である。

参考文献
[1] 内田、小柳：コンピュータアーキテクチャ, オーム社, 2004.
[2] 安藤秀樹 ： 命令レベル並列処理, コロナ社, 2005
[3] Mike Johnson, 村上和彰 ：スーパースカラ・プロセッサ, 日経 BP 出版センター, 1994.