計算機アーキテクチャ 101－19
（１９９３．８．２０）

- Restructured Instruction and Switched Context RISC processor - (RISC)^{2}の提案

石井吉彦　野上　忍　小野寺毅　三浦敏孝　　村岡洋一
ishii@muraoka.info.waseda.ac.jp
早稲田大学理工学部

科学技術計算からコンパイラ・テキスト処理などといった汎用計算全てを細・粗粒度並列処理により高速実行できる新しいプロセッサ・アーキテクチャ(RISC)^{2}を提案する。(RISC)^{2}では、遅延サイクルを付加したVLIW命令を用い、複数のコンテクストに対しVLIW命令を同時供給し、コンテクスト内をin-order実行し細粒度並列処理を行い、コンテクスト間をout-of-order実行し粗粒度並列処理を行う。これによりシングルプロセッサ構成では、複数のコンテクストに対し制御依存を越えてオーバラップ実行（投機的実行）させ、制御依存の隠蔽を図り、マルチプロセッサ構成では、複数のコンテクストに対し遅延時間なしで切り換え、通信レイテンシの隠蔽を図る。

- Restructured Instruction and Switched Context RISC processor - (RISC)^2

Yoshihiko ISHII Shinobu NOGAMI Takashi ONODERA
Toshitaka MIURA Yoichi MURAOKA

School of Science and Engineering, Waseda University
3-4-1 Okubo, Shinjuku-ku, Tokyo 169, Japan

This paper proposes an new processor architecture (RISC)^2. (RISC)^2 provides VLIW to plural contexts in parallel by using VLIW tagged delay cycle, executes each context in order, and executes out of order between plural contexts.

Therefore, in the case of single processor system, (RISC)^2 extracts fully parallelism of instructions, because (RISC)^2 executes plural contexts in parallel except the control dependency, in other words, (RISC)^2 executes plural contexts on the eager evaluation.

In the case of multi-processor system, (RISC)^2 extracts fully parallelism of contexts, because (RISC)^2 suspends the execution of only context delayed by communication latency, and continues the execution of other contexts.

1、はじめに

まず、シングルプロセッサ構成に対して考察する。プロセッサでの細粒度並列処理において、抽出可能な命令レベルの並列性を阻害する要因として、命令間のデータ依存と制御依存が挙げられる。命令間のデータ依存といった並列性の阻害は、コンパイラによる静的な最適化によって大部分は対処できる。しかし、命令間の制御依存といった並列性の阻害は、分岐命令といった実行時の動作に依存するため、分岐予測などによる静的な最適化では十分に対処できない。

そこで、本稿で提案する(RISC)^{2}では、この命令間の制御依存といった並列性の阻害を回避し、十分な並列性を抽出する方法として、分岐命令で区切られた命令群をコンテクストとしたとき、複数のコンテクストに対し命令を同時供給し、コンテクストをin-order実行させ、コンテクスト間をout-of-orderで制御依存を越えてオーバラップ実行（投機的実行）する機能を具現化する。

また、プロセッサでの細粒度並列処理を行うアーキテクチャとして、コンパイル時に静的に並列性を抽出するVLIWと実行時に動的に並列性を抽出するスーパスカラが挙げられる。VLIWでは、コンパイル時に静的に、1コンテクスト中の1サイクルに対して並列実行させる命令群を1VLIW命令として構成するため、複数のコンテクストに対しVLIW命令を同時供給することが困難である。また、スーパスカラでは、実行時に動的に、複数のコンテクストに対して並列実行させる命令群の依存関係を調べるため、クロック速度の向上が困難である。

そこで、(RISC)^{2}では、遅延サイクルを付加したVLIW命令を導入し、コンパイル時に静的に、1コンテクスト中の数サイクルに対して並列・逐次実行させる命令群を1VLIW命令として構成し、複数のコンテクストに対しVLIW命令を同時供給する機能を具現化する。

次に、マルチプロセッサ構成に対して考察する。プロセッサでの粗粒度並列処理において、抽出可能なコンテクストレベルの並列性を阻害する要因として、通信レイテンシが挙げられる。通信レイテンシといった並列性の阻害は、遠隔データのLoad命令といった実行時の動作に依存するため、実行時に動的にコンテクストを切り換えることによって対処する。しかし、コンテクスト切り換えに伴う遅延時間が問題である。

そこで、(RISC)^{2}では、この通信レイテンシといった並列性の阻害を回避する方法として、複数のコンテクストに対しVLIW命令を同時供給し命令フェッチステージで保持させ、実行可能なコンテクストを選択しオペランドフェッチステージへ進め、遅延時間なしでコンテクストを切り換える機能を具現化する。さらに、(RISC)^{2}では、実行可能なコンテクストが複数選択されたとき、コンテクスト間を機能ユニットが競合しない限りont-of-order実行する機能を具現化する。

本稿では、(RISC)^{2}のアーキテクチャの概要を述べ、簡単なプログラムによる定量的評価により、(RISC)^{2}のアーキテクチャの有効性を述べる。

2、(RISC)^{2}のアーキテクチャ

本章では、(RISC)^{2}特有の機能をまとめ、アーキテクチャを説明する。

2.1　機能

(RISC)^{2}特有の機能として次の3つが挙げられる。

(a)投機的実行機能

複数のコンテクストに対しVLIW命令を同時供給し、コンテクスト内をin-order実行させ、コンテクスト間をout-of-orderで制御依存を越えてオーバラップ実行（投機的実行）する機能

なお、投機的実行するコンテクスト内に他のプロセッサや他のコンテクストに対して副作用を発生させるStore命令を含んではならない。このStore命令による副作用とは、投機的実行したコンテクストで求めたリザルトデータがStore命令によってメモリ上に反映された後、もし投機的実行が失敗であったとき、メモリの状態の復元が困難であることを意味する。

(b)命令再構成（Restructured Instruction）機能

遅延サイクル付きのVLIW命令[1]を1サイクル毎にコンテクストを変えて分配し、命令フェッチステージで通常のVLIW命令に展開し、複数のコンテクストに対しVLIW命令を同時供給する機能

(c)コンテクスト切り換え（Switched Context）機能

複数のコンテクストに対しVLIW命令を同時供給し命令フェッチステージで保持させ、実行可能な複数のコンテクストを選択し、複数のコンテクストを機能ユニットが競合しない限りont-of-orderでオ

ペランドフェッチステージへ進め、遅延時間なしでコンテクストを切り換える機能

2.2 構成

図1に(RISC)^{2}のマルチプロセッサ構成を示す。

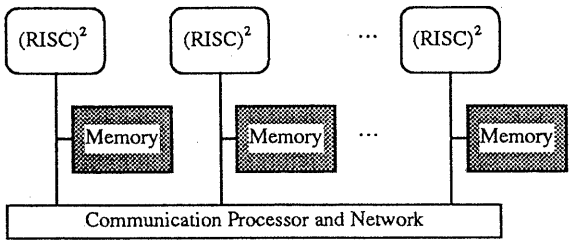

図1　(RISC)^{2}のマルチプロセッサ構成

図1に示すように、(RISC)^{2}のマルチプロセッサ構成は、(RISC)^{2}が通信プロセッサ（Communication Processor）を介して複数結合したものである。

(RISC)^{2}は、複数のコンテクストに対し細・粗粒度並列処理を行い、通信プロセッサは、遠隔データのLoad/Store命令の処理と共に、コンテクストが実行可能かどうか判断を行う。

例えば、コンテクストが遠隔データのLoad命令を含む場合、(RISC)^{2}では、このコンテクストは実行不可能なので命令フェッチステージで保持される。但し、遠隔データのLoad命令のみオペランドフェッチステージまで先行実行させ、通信プロセッサへコンテクスト番号と遠隔データのLoad命令の内容を伝える。通信プロセッサでは、遠隔データのLoad/Store命令を処理すると共に、(RISC)^{2}へ実行可能となったコンテクスト番号を伝える。その後、(RISC)^{2}では、実行可能になったコンテクストに対し命令フェッチステージでの保持を解除しオペランドフェッチステージへ進める。

なお、通信プロセッサの詳細は別途報告する。

図2に(RISC)^{2}の構成を示す。

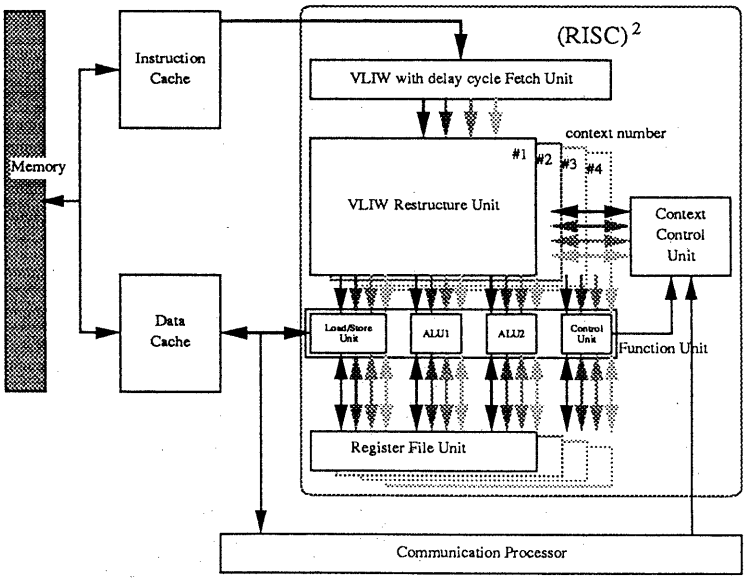

図2　(RISC)^{2}の構成

図2に示すように、$(RISC)^2$は、遅延サイクル付きVLIWフェッチユニット（VLIW with delay cycle Fetch Unit）、VLIW再構成ユニット（VLIW Restructure Unit）、機能ユニット（Function Unit）、レジスタファイルユニット（Register File Unit）、コンテクスト制御ユニット（Context Control Unit）、といった5つのユニットから構成される。

遅延サイクル付きVLIWフェッチユニットは、2.1で述べた(b)命令再構成機能を実現するために、命令キャッシュから供給された遅延サイクル付きのVLIW命令を1サイクル毎にコンテクスト番号が同じVLIW再構成ユニットに対し分配する。

VLIW再構成ユニットは、(b)命令再構成機能を実現するために、遅延サイクル付きVLIWフェッチユニットから供給された遅延サイクル付きのVLIW命令を、遅延サイクルによって通常のVLIW命令に展開する。また、VLIW再構成ユニットは命令キューのように実行不可能なコンテクストを保持する[2]。

機能ユニットは、Load/Storeユニット、ALU1、ALU2、Controlユニット、により構成される。また、機能ユニットは、2.1で述べた(a)投機的実行機能を実現するために、実行時エラーが投機的実行によるものかプログラミングミスによるものか判断する[3]。例えば、実行時エラーが発生した場合、演算ステージでエラー情報をリザルトデータに変換し、レジスタストアステージでエラー情報をレジスタファイルユニットに書き込み、エラー情報を伝達させ処理を続ける。その後、その実行時エラーが投機的実行によるものであればエラー情報は投機的実行の失敗によって無効化されるが、実行時エラーがプログラミングミスによるものであればエラー情報は無効化の前に副作用を発生させるStore命令まで伝達するため、この時点で処理を中止しエラー情報をユーザに伝える。

レジスタファイルユニットは、コンテクスト毎に用いるローカルレジスタファイルと、データ依存を有するコンテクスト間で用いるグローバルレジスタファイルにより構成される。また、ローカルレジスタファイルは、(a)投機的実行機能を実現するために、投機的実行するコンテクストのリザルトデータをスプールする。

コンテクスト制御ユニットは、データ依存を有するコンテクスト間の同期をとると共に、2.1で述べた(c)コンテクスト切り換え機能を実現するために、通信プロセッサからのコンテクスト番号を用い、複数のコンテクストに対し命令フェッチステージで保持／解除を制御する。また、コンテクスト制御ユニットは、(a)投機的実行機能を実現するために、投機的実行を失敗したコンテクストを無効化する。

2.3 命令セット

表1に$(RISC)^2$の命令セットを示す。

表1 $(RISC)^2$の命令セット

CLASS	OPECODE	OPERANDS	COMMENTS	
ALU	add	Rd, Rs, S2	Rd ← Rs + S2	add
	sub	Rd, Rs, S2	Rd ← Rs - S2	subtract
	subr	Rd, Rs, S2	Rd ← S2 - Rs	subtract reversed
	mul	Rd, Rs, S2	Rd ← Rs × S2	multiply
	and	Rd, Rs, S2	Rd ← Rs & S2	logical AND
	or	Rd, Rs, S2	Rd ← Rs ! S2	logical OR
	nor	Rd, Rs, S2	Rd ← Rs nor S2	logical NOR (if Rs = R0 then logical NOT)
	exor	Rd, Rs, S2	Rd ← Rs exor S2	logical EXOR
	sll	Rd, Rs, S2	Rd ← Rs << S2	shift left (Data type is two's compliment)
	srl	Rd, Rs, S2	Rd ← Rs >> S2	shift right logical
	sra	Rd, Rs, S2	Rd ← Rs >> S2	shift right arithmetic
	set0	Rd, S1	Rd ← 0<31:14> S1<13:0>	set data<13:0>
	set1	Rds, S1	Rds ← Rds<31:28> S1<27:14> Rds<13:0>	set data<27:14>
	set2	Rds, S1	Rds ← S1<31:28> Rds<27:0>	set data<31:28>
Control	eqz	Rs, S2	if Rs = 0 then PC = S2	equal to zero (if Rs = R0 then non-condition jump)
	eqzr	Rs, S1	if Rs = 0 then PC = PC + S1	equal to zero relative (if Rs = R0 then non-condition jump)
	nez	Rs, S2	if Rs ≠ 0 then PC = S2	not equal to zero
	nezr	Rs, S1	if Rs ≠ 0 then PC = PC + S1	not equal to zero relative
	gtz	Rs, S2	if Rs > 0 then PC = S2	greater than zero
	gtzr	Rs, S1	if Rs > 0 then PC = PC + S1	greater than zero relative
	gez	Rs, S2	if Rs ≧ 0 then PC = S2	greater than or equal to zero
	gezr	Rs, S1	if Rs ≧ 0 then PC = PC + S1	greater than or equal to zero relative
	ltz	Rs, S2	if Rs < 0 then PC = S2	less than zero
	ltzr	Rs, S1	if Rs < 0 then PC = PC + S1	less than zero relative
	lez	Rs, S2	if Rs ≦ 0 then PC = S2	less than or equal to zero
	lezr	Rs, S1	if Rs ≦ 0 then PC = PC + S1	less than or equal to zero relative
	pre	S2	PCpre = S2	branch prediction
	prer	S1	PCpre = PC + S1	branch prediction relative
Load/ Store	ld	Rd, S2	Rd ← M [S2]	load from memory
	st	Rs, S2	M [S2] ← Rs	store to memory
	ls	Rds, S2	Rds ← M [S2] with no delay cycle , S2' ← S2 + α executed by ALU, M [S2'] ← Rds with delay cycle	load and store

Rd : Distination Register
Rs : Source Register
Rds : Distination and Source Register
S1 : Immediate Date
S2 : Source Register or Immediate Date
PC : Program Counter
PCpre : Program Counter (Branch Prediction)
R0 = 0

コンテクスト毎に用いるローカルレジスタファイル内のレジスタと、データ依存を有するコンテクスト間で用いるグローバルレジスタファイル内のレジスタを指定できるように、ALUで用いる命令セットでは3オペランド形式を採用している。

Controlユニットで用いるpre/prer命令は投機的実行に先だって分岐予測先のプログラムカウンタ求める命令である。

Load/Storeユニットで用いるls命令は、他の命令と比較して出現頻度の高いLoad/Store命令を圧縮した命令[4]であり、これを用いることで、1コンテクスト中の多くのサイクルに渡って並列・逐次実行させる命令群を1VLIW命令として構成でき、円滑に複数のコンテクストに対しVLIW命令を同時供給できる。

さらに、図3に$(RISC)^2$の命令フォーマットを示す。

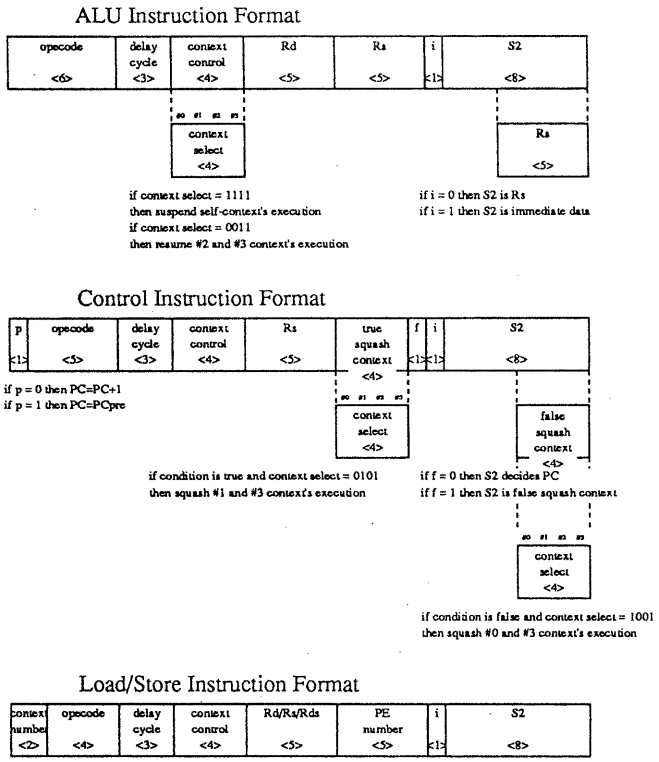

図3　$(RISC)^2$の命令フォーマット

$(RISC)^2$特有の命令フォーマットの項として、delay cycle<3>とcontext control<4>が挙げられる。

delay cycle<3>は、VLIW再構成ユニットにおいて、遅延サイクル付きのVLIW命令を通常のVLIW命令に展開するために用いる遅延サイクルのことである。

context control<4>は、コンテクスト制御ユニットにおいて、データ依存を有するコンテクスト間の同期をとるために用いる。

Controlユニットで用いる特有の命令フォーマットの項として、p<1>、true squash context<4>、false squash context<4>が挙げられる。

p<1>は、投機的実行に先だって求めた分岐予測先のプログラムカウンタを使用するかどうかを表すフラグである。

true squash context<4>／false squash context<4>は、コンテクスト制御ユニットにおいて、分岐命令の条件が真／偽のとき、投機的実行していたコンテクストを無効化するために用いる。

Load/Storeユニットで用いる特有の命令フォーマットの項として、context number<2>とPE number<5>が挙げられる。

context number<2>は、遅延サイクル付きVLIWフェッチユニットにおいて、遅延サイクル付きのVLIW命令をVLIW再構成ユニットに分配するために用いるコンテクスト番号のことである。

PE number<5>は、遠隔データのLoad/Store命令であるかどうかを表すと共に、通信プロセッサにおいて用いる。

3、(RISC)^{2}の動作

本章では、(RISC)^{2}の投機的実行とコンテクスト切り換えにおける動作を説明する。

3.1 投機的実行における動作

投機的実行するコンテクストとして、ニュートン法で3次方程式$ax^3+bx^2+cx+d=0$を解くプログラムを用い、(RISC)^{2}の投機的実行における動作を説明する。

比較のために通常のVLIW命令で表したプログラムを図4に示し、(RISC)^{2}で用いる遅延サイクル付きのVLIW命令で表したプログラムを図5に示す。なお、単純化のため全ての命令の実行時間を1クロックと仮定する。

```
     Load/Store      ALU1        ALU2          Control
loop:                t0=b+x
                     t1=t0*x     t5=3*x
                     t2=t1+c     t6=2b+t5
                     t3=t2*x     t7=t6*x
                     t4=t3+d     t8=t7+c
                     t9=t4/t8
                     x=x-t9      t10=t9*t9
                                 t11=t10-eps
                                               if t11>=0 then loop
exit:
```

a,b,c,d,eps…constant number

図4 VLIW命令で表したプログラム

```
     Load/Store      ALU1        ALU2          Control
loop:                t0=b+x                    context number #1
                     t1=t0*x     t5=3*x
                     t2=t1+c     t6=2b+t5
                     t3=t2*x     t7=t6*x
                     t4=t3+d     t8=t7+c
                     t9=t4/t8    t11=t10-eps[2]  if t11<0 then exit[3]
                     x=x-t9      t10=t9*t9
                     t0=b+x                    context number #2
                     t1=t0*x     t5=3*x
                     t2=t1+c     t6=2b+t5
                     t3=t2*x     t7=t6*x
                     t4=t3+d     t8=t7+c
                     t9=t4/t8    t11=t10-eps[2]  if t11<0 then exit[3]
                     x=x-t9      t10=t9*t9       PCpre=loop
exit:
```

[?]…delay cycle

図5 遅延サイクル付きのVLIW命令で表したプログラム

次に、遅延サイクルによって通常のVLIW命令に展開された命令フェッチステージでのプログラムの挙動を図6に示す。

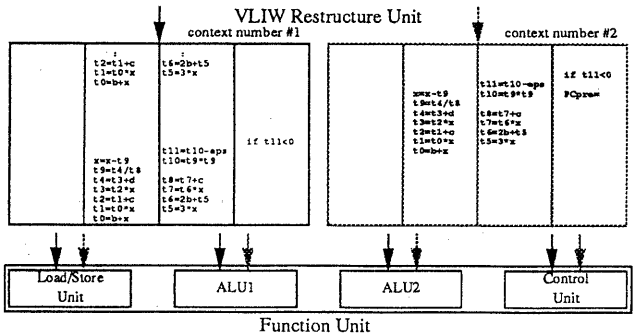

図6 命令フェッチステージでのプログラムの挙動

図6に示したように、コンテクスト内をVLIWによりin-order実行し、番号1のコンテクストと番号2のコンテクスト間はout-of-orderで制御依存を越えてオーバラップ実行（投機的実行）している。
また、図4と図6より、1イタレーション当りの命令実行時間を比較すると、投機的実行により約22%実行時間を短縮している。

3.2 コンテクスト切り換えにおける動作

コンテクスト切り換えが生じるコンテクストとして、s=s+A(i)（A(i)…遠隔データ）といった総和を求めるプログラムを用い、(RISC)^{2}のコンテクスト切り換えにおける動作を説明する。なお、コンテクスト切り換え先はs=s+B(i)（B(i)…ローカルメモリに対するデータ）といった同種のプログラムとする。
比較のために通常のVLIW命令で表したプログラムを図7に示し、(RISC)^{2}で用いる遅延サイクル付きのVLIW命令で表したプログラムを図8に示す。

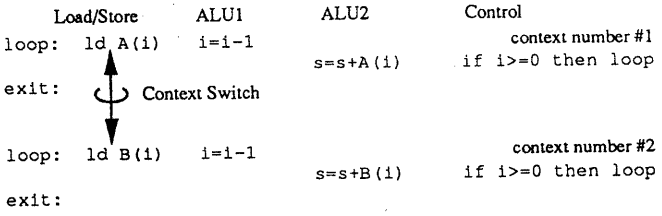

図7　VLIW命令で表したプログラム

```
     Load/Store     ALU1        ALU2                Control
loop: ld B(i')      i'=i'+1     s'=s'+B(i')[1]      if i'<0 then exit;PCpre=loop[2]  context number #2
      ld A(i)       i=i+1       s=s+A(i)[1]                                          context number #1
exit:
```

図8　遅延サイクル付きのVLIW命令で表したプログラム

次に、遅延サイクルによって通常のVLIW命令に展開された命令フェッチステージでのプログラムの挙動を図9に示す。なお、図9は、ループ繰り返し回数が3のときの遠隔データA(i)のLoad命令において通信レイテンシが4サイクルと動的に変化した様子である。

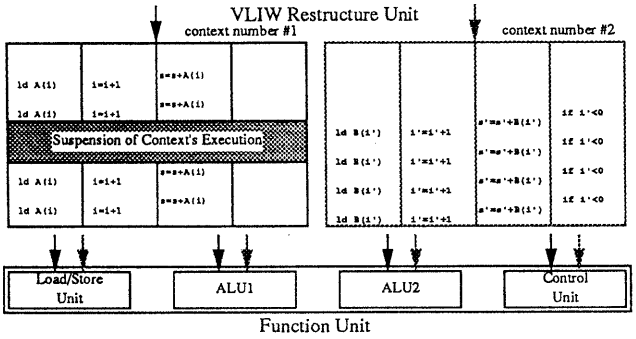

図9　命令フェッチステージでのプログラムの挙動

図9に示したように、番号1のコンテクストと番号2のコンテクスト間をout-of-order実行することにより、番号1のコンテクストで生じた通信レイテンシを、番号2のコンテクストの実行を続けることで、隠蔽を図っている。なお、図9で示したプログラムでは、番号1のコンテクストと番号2のコンテクスト間にデータ依存を有していないが、データ依存を有するコンテクスト間でも、2.3で述べたcontext control<4>を用いて部分的に同期をとりながら、限定したout-of-order実行により通信レイテンシの隠蔽を図れるものと考える。

4、おわりに

本稿では、VLIWを基に(a)投機的実行機能、(b)命令再構成機能、(c)コンテクスト切り換え機能、を具現化した新しいプロセッサ・アーキテクチャ$(RISC)^2$を提案した。

これらの機能により、シングルプロセッサ構成では、複数のコンテクストに対しVLIW命令を同時供給し、コンテクスト内をin-order実行させ、コンテクスト間をout-of-orderで制御依存を越えてオーバラップ実行（投機的実行）させる。その結果、制御依存の隠蔽を図る。

また、マルチプロセッサ構成では、複数のコンテクストに対しVLIW命令を同時供給し命令フェッチステージで保持させ、実行可能な複数のコンテクストを選択し、複数のコンテクストを機能ユニットが競合しない限りont-of-orderでオペランドフェッチステージへ進め、遅延時間なしでコンテクストを切り換える。その結果、通信レイテンシの隠蔽を図る。

現在、$(RISC)^2$のシステム・シミュレータとコンパイラを作成している。今後は、$(RISC)^2$の仕様を詰めると共に、多くのプログラムを用いて制御依存の隠蔽と通信レイテンシの隠蔽の効果のシミュレーション評価が課題である。

参考文献

[1]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", 信学論, Vol. J74-D-I, N.9, pp.613-622 (1991)

[2]丸島敏一,西　直樹,大沢謙二,中崎良成: "パイプライン計算機における基本ブロック間並列処理アーキテクチャ", 並列処理シンポジウムJSPP'91, pp.117-124 (1991)

[3]萩本　猛, 草野義博, 山名早人, 村岡洋一: "並列処理システム-晴-における実行時エラーの処理", 情処第39回全大, 6W-5, pp.1800-1801 (1989)

[4]児玉祐悦, 坂井修一, 山口喜教: "データ駆動型シングルチッププロセッサEMC-R　ー動作原理と実装ー", 並列処理シンポジウムJSPP'90, pp.161-168 (1990)