命令並列処理機構を意識したスケジューリングを支援するレジスタ構成とその効果

藤井啓明† 稲上泰弘†

RISC プロセッサにおける性能低下の要因である主記憶アクセスレイテンシの問題を解決するレジスタ構成として Queue Register 方式を提案した．Queue Register は，主記憶アクセスレイテンシを隠蔽するという方針のもと命令並列処理機構を活用して高性能を実現可能とするコードスケジューリングであるモジュロスケジューリングを支援し，キャッシュあふれを起こす大規模数値処理においても，RISC プロセッサの高い処理性能を引き出す．Livermore 14 ループで本方式の効果を評価した結果，従来アーキテクチャに対して平均で1.50～2.63倍の性能向上見込みを得た．また，Queue Register 方式は，従来から同様の目的で提案されている他のレジスタ構成方式よりも高性能を実現できる．

A Register File Architecture for Instruction Level Parallel Processing and Its Evaluation

Hiroaki Fujii† and Yasuhiro Inagami†

We propose a new register configuration for RISC processor, named "Queue Register." Queue Register assists compiler realizing "Modulo Scheduling," that is a kind of software pipeline technique. Performance of conventional RISC processor is kept high by utilizing cache memory. On the other hand, programs for large scale numerical applications handle many data. Most of such data cannot stay in cache memory, so a processor must read many data from main memory. Reading many data from main memory results in performance degradation. Modulo scheduling can improve performance of conventional RISC processor by operating memory access and other operations simultaneously. Modulo scheduling needs many registers. Queue Register can support that. In Queue Register, every logical register has plural registers and consists data queue. In the case of processing Livermore 14 Kernel, Queue Register can increase performance of RISC processors by 1.5-2.63 times. Furthermore, Queue Register has more ability on drawing high performance of RISC processors than former proposed register configurations, which have same purposes as Queue Register.

1. はじめに

RISC プロセッサの高い命令処理性能を維持するには，演算器への高いデータ供給性能が要求される．現在の RISC プロセッサは，キャッシュによってこの要求を満たしている．しかし，大規模数値処理ではプログラムが扱うデータ量がキャッシュ容量より大きく，キャッシュミスが頻発するため，主記憶アクセス時間の全体処理時間に占める割合が増大し，性能が低下する．この問題は，

（1）主記憶のデータ供給能力（メモリスループット）の向上

（2）主記憶アクセスに伴うプロセッサの待ち時間（主記憶アクセスレイテンシ）の削減

によって解決できる．（1）の課題は，メモリチップのアクセスピッチ短縮および主記憶の多バンク化により解決される．一方，主記憶アクセスレイテンシを完全になくす方法は存在しないが，その影響を隠蔽する手段としてデータ先読み処理がある．データ先読み処理は，ある演算命令が必要とするデータを，あらかじめ主記憶アクセスレイテンシを考慮した時間分だけその演算命令実行タイミングよりも先行させて実行するロード命令によって得る処理手法である．データ先読み処理において，先行するロード命令の得たデータの格納先はレジスタである．同種の処理手法に先行するロード命令の得たデータの格納先をキャッシュとするプリフェッチ技法が存在する．しかし，プリフェッチ技法では，

（1）キャッシュ上でプリフェッチ・アクセスと通常アクセスの競合が起こり，スループットが不足

† (株)日立製作所中央研究所
Central Research Laboratory, Hitachi, Ltd.

する；
（2）プリフェッチ命令が余分に必要であり，性能的に問題となる可能性がある；
（3）先読みデータのキャッシュへの登録によって必要なデータが追い出される；

などの問題が懸念されるため，本研究はデータ先読み処理を対象とした．

データ先読み処理実現の中心的な課題は，コンパイラの主記憶アクセスレイテンシを考慮したコードスケジューリングアルゴリズムである．

データ先読み処理を実現する1つのコンパイラ技法に，モジュロスケジューリング（Modulo Scheduling）[1),2)]がある．モジュロスケジューリングは，プログラムのループ部に対してマシン語コードレベルでソフトウェアパイプライン技法[3)]を適用した命令コード生成方法であり，本来演算器レイテンシを隠蔽する手法であるが，その概念を全く変えずにデータ先読み処理，すなわち主記憶アクセスレイテンシ隠蔽も実現する．

モジュロスケジューリングなどが実現するデータ先読み処理では，長大な主記憶アクセスレイテンシを隠蔽するために，数多くのデータ先読みのためのロード命令を連続発行する．このため，連続するそれぞれのロード命令が使用するレジスタを個別に用意する必要があり，多くのレジスタが必要になる．この課題に対して，単純に多数のレジスタをプロセッサ内に用意し，そのすべてを命令から直接指定可能とした場合，

（1）命令中のレジスタ指定フィールドの大幅な拡張が必要になる；
（2）多数のレジスタをそれぞれレジスタ指定番号を違えて指定する必要があり，プログラムコード量が増大する；
（3）コンパイラのレジスタ割当てアルゴリズムが複雑になる；

という問題を抱える．

われわれは，モジュロスケジューリングによる命令コードの，繰り返し処理に関わるレジスタ使用の順序性に着目して，個々のレジスタをハードウェアキュー構成（複数のレジスタでキューを構成）とする Queue Register 方式を提案し，性能評価によって本方式の有効性を確かめた．

2. Queue Register 方式

2.1 Queue Register の構成

Queue Register の構成を**図1**に示す．Queue Register 方式では，命令が指定するレジスタ番号は論理的なレジスタ番号である．各論理レジスタ R0～Rn−1 は，複数本の物理的なレジスタをサイクリックに利用して FIFO キューを構成する．また，各論理レジスタは，キュー構成を実現するために，それぞれキュー制御部とキューの先頭と最後尾を示すポインタ（Top, Bottom）を持つ．

Queue Register 方式における命令のレジスタ指定フィールドは，キュー制御ビットと論理レジスタ番号指定フィールドからなる．論理レジスタ番号指定フィールドは，従来アーキテクチャのレジスタ指定フィールドに等しい．キュー制御ビットは，レジスタキューを活用するための情報を示すフィールドである．キュー制御ビットは，その 0/1 情報により，指定した論理レジスタをキューモードで使うか通常モードで使うかの選択を行うために用いる．キューモード，通常モードを，以下に定義する．

【キューモード】

レジスタからのデータ読み出し時は，論理レジスタを構成するキューの先頭ポインタ（Top）が指す物理レジスタから値を読み出し，データを読み出した後，先頭ポインタの内容を，キューを構成する次の物理レジスタを指すように変更する．すなわち，読み出した値をキューから取り除く．

レジスタへのデータ書込み時は，キューの最後尾ポインタ（Bottom）を，キューを構成する次の物理

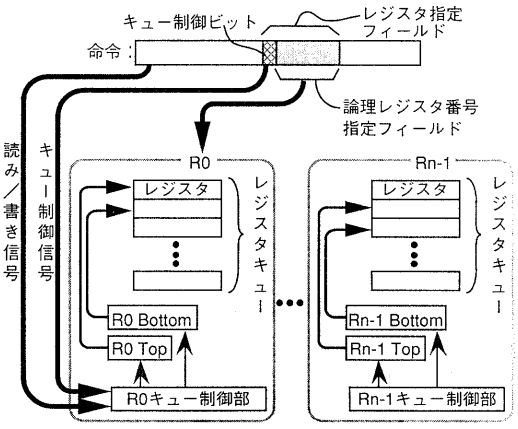


図1 Queue Register の構成
Fig. 1 Organization of queue register.

レジスタを指すように変更した後に，新たな最後尾ポインタが指す物理レジスタへ値を書き込む．

【通常モード】

レジスタアクセスはすべて，キューの先頭ポインタが指す物理レジスタに対して行い，先頭ポインタや最後尾ポインタの内容は変更しない．

Queue Register 方式では，Queue Register を常に通常モードで利用して，従来アーキテクチャとの互換性を保てる．しかし，キュー制御ビットを各レジスタ指定フィールドに用意するために，新しい命令フォーマットの導入が必要になる．

2.2 Queue Register の利用方法

2.2.1 モジュロスケジューリング

1章で述べたモジュロスケジューリングは，VLIW 方式などの命令並列処理機構を意識したソフトウェアパイプライン技法の1つである[1)~3)]．モジュロスケジューリングは，対象とするプロセッサで同時動作可能な処理ユニットの種類と数，それぞれの処理ユニットの処理レイテンシなどを意識して，命令コードスケジューリングを行い，プログラムのループ部の処理性能向上を達成する．

モジュロスケジューリングでは，まずループの iteration 1回分のコードを作成し，このコードを iteration の1回目分から順に一定間隔（この間隔を Iteration Interval と呼ぶ）ずつずらせて重ね合わせる．

モジュロスケジューリングが次に行う処理はレジスタ再割当てである．レジスタ再割当て前の命令コードは，ループの iteration 1 回分のコードを順に重ねただけであるので，レジスタの使用法に問題がある．すなわち，i 回目の iteration と $(i+1)$ 回目の iteration で，同じ番号のレジスタを使用しながら，それぞれで異なったデータを保持しようとするため，プログラムが正しい結果を得られない．したがって，通常のレジスタ構成をとるプロセッサに対しては，コンパイラが，処理の重なる iteration ごとにレジスタ番号を違えて割り当てるレジスタ再割当てを行う必要がある．この手続きはコンパイラにとって負担となるばかりでなく，命令コード量増加を招く．また，番号つけ変えに足るレジスタがないときには，iteration の重なり度を減らす必要があり，性能が低下する．

2.2.2 Queue Register 方式によるモジュロスケジューリングのサポート

Queue Register 方式は，iteration の各回で使用するレジスタを，各回ごとにレジスタキューの深さ方向で使い分けるといった使用法を提供する．したがって，命令コード上では，iteration の各回で常に同じレジスタ番号を指定できるため，モジュロスケジューリングに基づく命令コードの量を低減する．

```
      DO 3 K = 1, 1000
3     Q = Q + Z(K) * X(K)
```

図 2 Livermore #3 ソースコード
Fig. 2 Source code for livermore 3rd loop.

図2のプログラムは，次章で述べる性能評価において使用した Livermore 14 ループのうちの1つ (LOOP #3) である．

図2のループコードの1回の iteration 処理をマシン語レベルで記述すると，**図3**のようになる．なお，図3では，メモリレイテンシを7マシンサイクル，浮動小数点演算レイテンシを7マシンサイクル，浮動小数点演算レイテンシを2マシンサイクルと仮定した．

モジュロスケジューリングでは，図3の iteration 1回分のコードを1回目分から順に一定間隔 (Iteration Interval) ずつずらせて**図4**のように重ね合わせる．図4のマシン語コードでは，VLIW 方式のプロセッサを想定して，ロード/ストア系命令，浮動小数点乗算命令，浮動小数点加算命令，整数系演算命令および分岐命令の5命令が同時実行可能であると仮定した．したがって，図4の左端の番号は VLIW の命令番号である．さらに図4では，ロード命令のアクセス先アドレスをロード命令実行時に自動更新可能であるとした．

図4のそれぞれの VLIW 命令中の各機能ユニットを操作する命令の左に付した白抜き数字は，その命令が図2のループコードの何回目の iteration に関わるかを示すが，ちょうど5回目の iteration に関わるロード命令が実行されるサイクルから数えて2マシンサイクル分の中に，図3に示した iteration 1 回分に関わる全命令が入っている．これらの命令は本来の

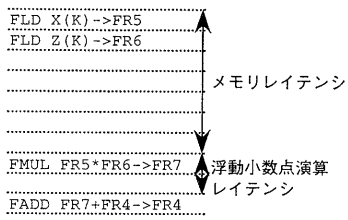


図 3 LOOP #3 の1回の繰り返しコード例
Fig. 3 Sample iteration code for livermore 3rd loop.

#	ロード命令	浮動小数点乗算命令	浮動小数点加算命令
1	1FLD X(1)->FR5		
2	1FLD Z(1)->FR6		
3	2FLD X(2)->FR5		
4	2FLD Z(2)->FR6		
5	3FLD X(3)->FR5		
6	3FLD Z(3)->FR6		
7	4FLD X(4)->FR5		
8	4FLD Z(4)->FR6	1FMUL FR5*FR6->FR7	
9	5FLD X(5)->FR5		
10	5FLD Z(5)->FR6	2FMUL FR5*FR6->FR7	1FADD FR7+FR4->FR4
11			
12		3FMUL FR5*FR6->FR7	2FADD FR7+FR4->FR4
13			
14		4FMUL FR5*FR6->FR7	3FADD FR7+FR4->FR4
15			
16		5FMUL FR5*FR6->FR7	4FADD FR7+FR4->FR4
17			
18			5FADD FR7+FR4->FR4

図 4 LOOP #3 の iteration コードの重ね合わせ
Fig. 4 Overlapped iteration codes for livermore 3rd loop.

ループコードの1回の iteration に関わる命令ではないが，命令コードとしては以降このような2マシンサイクルは，ループ回数分（1,000 回）繰り返される．つまり，プログラム上の本来の iteration 処理（図3）とは異なる iteration 処理（図4の命令9，10）が形成される．

モジュロスケジューリングでは，次にレジスタアクセスの競合を解決するために，レジスタの再割当てを実施する．Queue Register 方式導入下では，レジスタ再割当ては，アクセス競合のあるレジスタのキューモードでの使用指定という単純な処理で実現する．図4のコードの場合，レジスタ FR 5, FR 6, FR 7 をキューモードで使用する．

3. Queue Register 方式の性能評価

3.1 性能評価の仮定と評価用プログラム

評価対象として，以下の特徴を持つ RISC プロセッサを仮定した．

（1）命令処理方式：以下の5種命令を同時実行可能（5命令並列処理）
- ロード/ストア系命令
- 浮動小数点加算命令
- 浮動小数点乗算命令
- 整数系演算命令
- 分岐命令

（2）主記憶アクセスレイテンシ：20 MC*
（3）浮動小数点演算レイテンシ：4 MC
（4）キャッシュラインサイズ：64 Byte
（5）キャッシュ容量：1 MByte
（6）ロード/ストア命令：ベースレジスタ値更新（ポストインクリメント等）可能
（7）ループ計算向け分岐予測機構：あり
（8）キュー構成：浮動小数点レジスタのみ
（9）論理浮動小数点レジスタ数：32
(10) レジスタキューの深さ：8

主記憶構成，主記憶容量については特に限定しない．

評価用プログラムとしては，Livermore 14ループ*を選択した．

3.2 性能評価手法

性能評価は，評価用プログラムに対して，
（a）Queue Register を利用しない従来のコンパイル技法に基づくアセンブラ語コード
（b）Queue Register を利用したモジュロスケジューリングに基づくアセンブラ語コード
をハンドコーディングによって作成し，そのコードからループの実行に要するマシンサイクル数をカウントするという方法で行った．

なお，(a)のコードに基づく評価は，
(a.1) Cold Start での評価
(a.2) All in Cache での評価
の2種類について行った．

キャッシュミスの影響の考慮は，静的にアクセス先が決まっているデータについてのみ行った．LOOP #13, LOOP #14 には，1回の iteration 中でアクセス先アドレスを計算した後にアクセスを行う箇所がある．このようなアクセスは，キャッシュミスの状況が予測できないので，(a.1)ではこのアクセスに関わるキャッシュミスのペナルティは考慮しなかった．一方，(b)では，スケジューリングの性質上このようなアクセスについても常に 20 マシンサイクルのレイテンシを考慮する．

3.3 性能評価結果および結果分析

性能評価結果を**図 5** に示す．図5の性能指標 FLOPC は，FLoating Operations Per Cycle の略で，マシンサイクル当たりの浮動小数点演算数を表す．評価の仮定で，浮動小数点乗算命令と浮動小数点加算命令が同時に実行可能であるとしたので，FLOPC の最高値は 2.0 となる．(b)は，(a.1)に対して，最高で 4.87 倍，最低で 0.997 倍，平均（算術平均）で

* MC：マシンサイクル．

* Livermore ループと呼ばれるベンチマークプログラムには，14のループからなるプログラムと24のループからなるプログラムの2種が存在する．今回の評価で使用したのは，14のループからなるプログラムの方である．

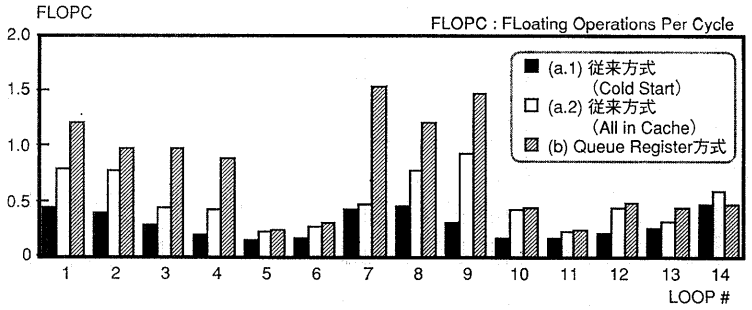


図 5 各ループの処理性能
Fig. 5 Performance for each livermore loops.

2.63 倍の性能向上を達成し，(a.2)に対して，最高で 3.19 倍，最低で 0.77 倍，平均（算術平均）で 1.50 倍の性能向上を達成する．

(b)が高性能を達成するのは，Queue Register 方式導入によって実現可能となったモジュロスケジューリングの効果である．

(b)でそれほど性能向上を達成できないループには，主に以下のような原因がある．特に，LOOP #14 については各繰り返し間でのデータ依存に関わるデータがメモリ渡しによるデータであるので，その影響は大きい．

- ループ間のデータ依存（LOOP #5, LOOP #6, LOOP #11, LOOP #14）
- もともと浮動小数点演算数が少ない（LOOP #10, LOOP #12）

4. レジスタ構成方法の比較

4.1 他のレジスタ構成方法

主記憶アクセスの効率を高める目的でレジスタをキュー構成とする考え方は，ベクトルレジスタと同様の発想に基づいている．これまでにも，RISC プロセッサに対してベクトルレジスタを導入するという提案はなされている[4)~6)]．

ただし，ベクトルレジスタは，通常はベクトル命令によってのみアクセス可能である．そのベクトル命令にはベクトル長を意識する必要性や，さまざまなアドレスパターンでのメモリアクセスに柔軟に対処できないという欠点がある．この問題を解決するために，ベクトルレジスタをスカラ命令によってアクセスできるようにしても，そのスカラ命令は，ベクトルレジスタ内の要素番号をある程度意識する必要がある．これに対し，レジスタキューはスカラ命令によってのみアクセス可能であり，命令からレジスタキューを構成する個々のレジスタを意識する必要がない．

さらに，Queue Register 方式は，命令のレジスタ指定ごとにキューを制御する機構を持つため，レジスタキューを通常のレジスタとしても使用可能である．

以上のように，レジスタキューとベクトルレジスタの本質的な違いに起因する差異が，Queue Register 方式とベクトルレジスタを RISC プロセッサに導入する方式（ベクトルレジスタ導入方式）の間に存在する．この差異がハードウェア/ソフトウェアの実装に影響を与えると考えられるが，基本的には，この 2 方式は同種の方式であると考えてよい．

一方，Queue Register 方式と同様にモジュロスケジューリングへの適用を考えたレジスタ構成として，Rotating Register File[1),2)] および Rotating Register File の考え方の元になった Context Register Matrix[7),8)] がある．これらのレジスタ構成では，レジスタ指定を，Iteration Control Pointer (ICP) と呼ぶハードウェアポインタが指示する物理番号からの相対値によって行う．ICP の値は，ループの iteration ごとに更新して，iteration ごとのレジスタの使い分けを保証する．ICP 値の更新は，モジュロスケジューリングで生成した命令コードの繰り返し処理のための分岐命令に同期して行う．

Rotating Register File や Context Register Matrix は，レジスタ番号のリネームを行う方式である．その意味では，文献 9)に提案のあるレジスタウィンドウを利用する方式も同類である．以降では，これらの方式をレジスタ番号リネーム方式と呼ぶ．

Queue Register 方式およびベクトルレジスタ導入方式とレジスタ番号リネーム方式の大きな違いの一つは，レジスタ番号リネーム処理の必要性の有無である．レジスタ番号リネーム方式では，繰り返し処理のための分岐命令などのプログラムコード中の同期点で

全レジスタに対するレジスタ番号のリネーム（レジスタ番号リネーム処理）を一斉に行う必要がある．レジスタ番号リネーム処理は，モジュロスケジューリングが生成する命令コード中にはもともと存在せず，また，演算命令やロード/ストア命令との同時実行が不可能であるため，性能低下要因となりうる．一方，Queue Register 方式では，レジスタ番号のリネームに相当するキュー管理処理がレジスタキューごとのハードウェア制御によって個々に行われるため，レジスタ番号リネーム処理のようなオーバヘッドが存在しない．この特長はベクトルレジスタ導入方式によっても実現可能である．

レジスタ番号リネーム処理を実行する前後では，命令から同じ番号でレジスタを指定しても，結果として使用する物理レジスタが異なる．したがって，レジスタ番号リネーム処理は，本来独立に動作可能なロード命令処理ストリームと演算処理ストリームの間に同期を必要とさせる．この性質については，後に詳述するが，RISC プロセッサのさらなる性能向上を目指す際の性能阻害要因となる．

以降では，Queue Register 方式とレジスタ番号リネーム方式について，

（1）レジスタ番号リネーム処理オーバヘッド

（2）レジスタ番号リネーム処理時の同期必要性

という2つの観点から性能評価を行う．

4.2 レジスタ番号リネーム処理オーバヘッドを考慮した性能比較

図6は，レジスタ番号リネーム処理の影響を表す性能評価結果である．図6の結果は，**図7**に示した LINPACK における中心的なループコード（DAXPY に対応する部分）の最内側ループに対して，3.1 節で述べた評価の仮定に基づき，Queue Register 方式とレジスタ番号リネーム方式それぞれの場合でモジュロスケジューリングを適用したコードを生成して得た．

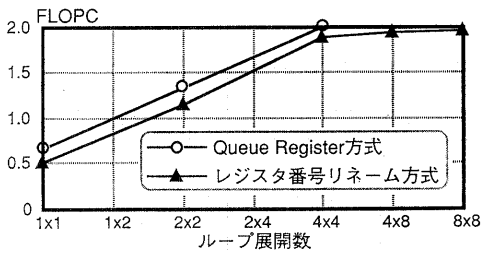


図 6 レジスタ番号リネーム処理の性能への影響
Fig. 6 Impact of register renaming operation on performance.

```
      DO 10 K = 1, N-1
      DO 10 J = K+1, N
      DO 10 I = K+1, N
         A(I,J)=A(I,J)+W(1,J)*A(I,K)
10    CONTINUE
```

図 7 LINPACK における中心的なループコード
Fig. 7 Main loop code for LINPACK.

ループ展開数については，図7のソースコードのJをループカウンタとするループを m 重に展開し，Kをループカウンタとするループを n 重に展開した場合を $m \times n$ と表現する．なお，レジスタ番号リネーム方式に対しては，指定可能なレジスタ数を制限していない．また，レジスタ番号リネーム処理オーバヘッドは1マシンサイクルと仮定した．

Queue Register 方式は，32 本という指定可能なレジスタ数の制限によって，図7のプログラムで対処可能なループ展開数が 4×4 までである．しかし，Queue Register 方式は，この展開数ですでにモジュロスケジューリングが達成可能なピーク性能を実現する．

一方，レジスタ番号リネーム方式は，常に Queue Register 方式の性能を下回る．この性能差の原因が，iteration の各回に必要となるレジスタ番号リネーム処理である．

モジュロスケジューリングに基づく命令コードの iteration のサイズが小さいほど，レジスタ番号リネーム処理のオーバヘッドは性能に影響を与える．したがって，レジスタ番号リネーム方式では，レジスタ番号リネーム処理オーバヘッドの影響を隠すためにループ展開数を上げる必要がある．図6の 4×4 展開以降の性能値は，レジスタ番号リネーム方式におけるループ展開の効果を示す．図6からも明らかなように，8×8 展開を試みても，レジスタ番号リネーム方式は，Queue Register 方式が 4×4 展開で達成する性能に追いつけない．

図7のループがモジュロスケジューリングによって最高性能を達成するためには，少なくとも 4×4 展開を要する．逆にループ展開数を上げるほど，命令コード量は増加し，必要なレジスタ数も増加するため，必要以上のループ展開は望ましくない．この観点からも，Queue Register 方式は，レジスタ番号リネーム方式よりも優れている．

4.3 レジスタ番号リネーム処理時の同期必要性を考慮した性能比較

モジュロスケジューリングでは，命令コードのスケジューリング時に主記憶アクセスレイテンシの大きさ

を仮定し，その大きさにしたがって最適コードを生成して高性能を実現する．3章および4.2節の性能評価結果は，3.1節で仮定した20マシンサイクルという主記憶アクセスレイテンシの仮定が正しかった場合の結果である．

主記憶アクセスレイテンシは，メモリ構成やシステム構成によって変化する．すなわち，主記憶アクセスレイテンシの仮定には確実性がない．例えば，多バンク構成の主記憶を採用したときのバンクへのアクセス競合によって主記憶アクセスレイテンシはコンパイラの仮定値よりも増加する．

モジュロスケジューリングは，主記憶アクセスレイテンシの正確な仮定の下では高性能を期待できるが，上記のような形で仮定が外れると性能が低下する可能性がある．

主記憶アクセスレイテンシの増加に対する性能低下の問題は，ソフトウェア手法では解決不能であり，ハードウェアで解決する必要がある．すなわち，主記憶アクセスレイテンシ隠蔽という課題の解決には，これまでに述べてきたモジュロスケジューリングなどによるデータ先読み処理だけでなく，命令処理機構などのハードウェア側からの支援が必要である．そのようなハードウェア支援機構として，演算命令とロード命令を独立に（非同期で）動作させる方式すなわちDecoupled Architecture[10]に代表されるような命令処理方式（以降，この方式を非同期命令並列処理方式と称する）が考えられる．非同期命令並列処理方式では，データ先読み処理において，主記憶アクセスレイテンシの増加によって演算命令実行が待ち状態にある間にもデータ先読みロード命令を継続して発行できるため，プログラム実行時に，演算命令実行タイミングとその演算命令実行に必要なデータの先読みロード命令域実行タイミングとの間の時間を実際の主記憶アクセスレイテンシを反映した値とし，性能低下を最小限に抑える．

非同期命令並列処理方式を仮定し，すべての主記憶アクセスで同等の主記憶アクセスレイテンシ増加があった場合の，図7に示したループの処理性能を各レジスタアーキテクチャについて評価した結果を**図8**に示す．

レジスタ番号リネーム方式では，レジスタ番号リネームの前後でレジスタ番号が参照する物理レジスタが異なるため，データ先読みロード命令は，レジスタ番号リネームのタイミングよりも先に実行できない．**図9**（a）に示すように，レジスタ番号リネームのタイミングは，命令コード上の iteration ごとに存在するので，1回の演算命令の実行タイミングのずれが後の iteration 処理に影響を与えて波及効果を生み，結果として主記憶アクセスレイテンシ増加にともなう演算実行待ちの影響を何度も受けて，性能が低下する．一方，Queue Register 方式では，図9（b）に示すように初めの1回だけ主記憶アクセスレイテンシ増加分の影響を受けるが，以降はレジスタ番号リネーム方式におけるレジスタ番号リネームタイミングのような制限が存在しないため，性能がほとんど低下しない．つま

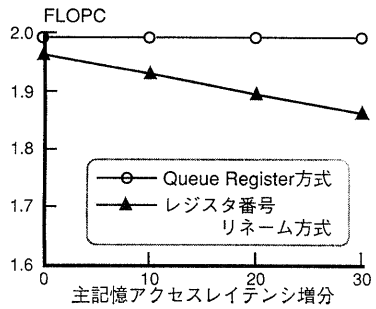
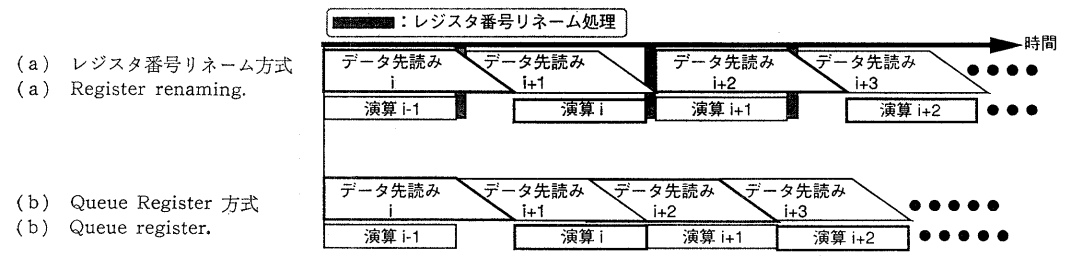


図8 主記憶アクセスレイテンシ増加の影響
Fig. 8 Impact of increasing memory access latency on performance.



（a） レジスタ番号リネーム方式
（a） Register renaming.

（b） Queue Register 方式
（b） Queue register.

図9 主記憶アクセスレイテンシ増加の影響の様子
Fig. 9 Impact of increasing memory access latency.

り，Queue Register 方式は，データ先読み処理に対して多数のレジスタを提供する支援機構となるだけでなく，主記憶アクセスレイテンシの不確定性にともなう影響をプログラム実行過程で解消する能力も持ち合わせている．

なお，本章で示した Queue Register 方式による性能は，ベクトルレジスタ導入方式によっても実現可能である．

5. まとめ

Queue Register 方式は，主記憶アクセスレイテンシ隠蔽を目的とするモジュロスケジューリングなどのデータ先読み処理実現のためのハードウェアサポートとして有効に機能する．また，Queue Register 方式は，従来から同様の目的で提案されているレジスタ番号リネーム方式の各レジスタ構成よりも高性能を実現し，特に主記憶アクセスレイテンシの不確定性といったプログラム実行時の動的な要因に柔軟に対処する能力を持つ．

今後は，Queue Register 方式の実現に要するハードウェア量およびインプリメント上の問題点についても考察を深めていく．

参考文献

1) Rau, B.R., Lee, M., Tirumalai, P.P. and Schlansker, M.S.: Register Allocation for Software Pipelined Loops, *Proceedings of the SIGPLAN '92*, pp. 283-299 (1992).
2) Tirumalai, P., Lee, M. and Schlansker, M.: Parallelization of Loops with Exits on Pipelined Architectures, *Proceedings of Supercomputing '90*, pp. 200-212 (1990).
3) Lam, M.: Software Pipelining: An Effective Scheduling Technique for VLIW Machines, *Proceedings of the SIGPLAN '88*, pp. 318-328 (1988).
4) 村上和彰：ハイパースカラ・プロセッサ・アーキテクチャ—命令レベル並列処理への第5のアプローチ—，JSPP '91 論文集，pp. 133-140 (1991).
5) 村上和彰，橋本 隆，弘中哲夫，安浦寛人：マイクロベクトルプロセッサ・アーキテクチャの検討，情報処理学会研究会報告，ARC-94-3, pp. 17-24 (1992).
6) Wulf, Wm. A.: Evaluation of the WM Architecture, *Proceedings of the 19th Annual ISCA*, pp. 382-390 (1992).
7) Rau, B.R., Yen, D.W.L., Yen, W. and Towle, R.A.: The Cydra 5 Departmental Supercomputer, *IEEE Computer*, Vol. 22, No. 1, pp. 12-35 (1989).
8) Dehnert, J.C., Hsu, P.Y.-T. and Bratt, J.P.: Overlapped Loop Support in the Cydra 5, *ASPLOS-III Proceedings*, pp. 26-38 (1989).
9) 中村 宏，位守弘充，伊藤元久，中澤喜三郎：レジスタウィンドウとスーパースカラ方式による擬似ベクトルプロセッサの提案，JSPP '92 論文集，pp. 367-374 (1992).
10) Smith, J.E.: Decoupled Access/Execute Computer Architectures, *ACM Transactions of Computer Systems*, Vol. 2, No. 4, pp. 289-308 (1984).

(平成5年9月14日受付)
(平成6年2月17日採録)

藤井 啓明（正会員）

1966年生．1989年京都大学工学部情報工学科卒業．1991年同大学大学院工学研究科情報工学専攻修士課程修了．同年(株)日立製作所中央研究所入所．以来並列計算機の研究開発に従事．

稲上 泰弘（正会員）

1953年生．1977年京都大学工学部情報工学科卒業．1979年同大学大学院工学研究科情報工学専攻修士課程修了．同年(株)日立製作所中央研究所入所．以来スーパーコンピュータおよび並列計算機の研究開発に従事．現在，(株)日立製作所中央研究所超高速プロセッサ部主任研究員．