計算機アーキテクチャ　106－2
（１９９４．　６．　１３）

多様なプログラムに適応するVLIWマシンの開発

遠藤浩太郎　境隆二　鈴木慎一郎
竹内陽一郎　石川禎　野崎正治　森良哉

東芝　情報通信システム技術研究所

４つのＲＩＳＣ型命令を並列実行するＶＬＩＷマシンを開発した。ＶＬＩＷ方式は静的なスケジューリングによって並列化を行うため、並列性の高いプログラムほど性能を確保しやすく、逆に非数値計算などのプログラムでは並列性が低いために性能の確保は難しい。ＶＬＩＷ方式を採用するにあたって、この低並列度なプログラムの性能をも考慮した。すなわち、低並列度なプログラムでも局所的に存在する並列性を静的に抽出できるように、ハードウェアの単純化とアーキテクチャの対称化を行ない、新たに条件実行制御方式を提案し、コンパイラによる高度な最適化を可能とした。これは同時にＶＬＩＷ方式が本来得意とする高並列度なプログラムの並列化の向上にもつながっている。本稿ではこれらがＶＬＩＷ方式として有効に機能することを示し、コンパイラで生成したオブジェクトコードを用いて他のアーキテクチャとの定性的な比較を試みる。

A multi-purpose VLIW architecture

Kotaro Endo, Ryuji Sakai, Shinichiro Suzuki
Yoichiro Takeuchi, Tadashi Isikawa, Ryoya Mori

TOSHIBA Information & Communication System Laboratory

We have developped a VLIW machine which can execute 4 RISC type instructions simultaneously. VLIW architecture is suitable for programs with high parallelism because it schedules instructions statically. On the other hand, it is difficult to get high performace of programs with little parallelism, such as non-numerical applications. On applying VLIW architecture, We also considered the performance with little parallelism. We propose some ideas which support compilers to realize advanced optimization, and to extract local parallelism. This paper describes the architecture outline, scheduling tactics typical of this architecture, and performance evaluations.

1．はじめに

コンピュータ技術の進歩にともなってあらゆる分野でのコンピュータ利用が進み、そしてコンピュータへのさまざまな要求は大きくなってきている。制御用途などのリアルタイムシステム、各種のサーバシステムなどでは高信頼性・リアルタイム・高性能なコンピュータが求められている。

一方技術の進歩によって素子の高速化と高密度実装、記憶装置の大容量化、コンパイラによる高度な最適化などが可能となっている。これらの要素技術の革新は、コンピュータへの新しいニーズとあいまって、新しいコンピュータアーキテクチャの必要性を高めている。

以上の考えにたって我々はＶＬＩＷ方式を採用したＲＩＳＣマシン（ＶＬ２０００）を開発した。ＲＩＳＣアーキテクチャはいまやありふれた方式であるが、その背景にはコンパイラに代表されるソフトウェア技術の進歩とハードウェアの大容量化がある。これらの技術革新によって、複雑な処理はソフトウェアでかたがわりしハードウェアを単純化して高速化を狙うアプローチが可能となっている。そしてこのアプローチを並列実行方式に押し進めたものが、ＶＬＩＷ方式である。また高信頼性と高性能という相反する要求もハードウェアの単純化によって解決できると考え、ＶＬＩＷ・ＲＩＳＣをアーキテクチャとして選択した。

以下本稿では、アーキテクチャの概要としてＶＬＩＷ方式、コンパイラによる高度な並列化の保証、低並列度なプログラムに対する性能の保証、ＣＩＳＣ型の従来機との互換性、特徴的な命令と分けて説明した後、このアーキテクチャでの並列化の戦略をいくつか紹介する。最後に評価とまとめをのべる。

2．アーキテクチャの概要

ＶＬ２０００のアーキテクチャの概要を紹介する。このアーキテクチャは高信頼、リアルタイム、高性能なマシンを指向し、比較的新しい技術であるＶＬＩＷ方式によるＲＩＳＣ構成で、従来のＣＩＳＣ型マシンが提供していた高機能なプリミティブも実現している。

2．1　ＶＬＩＷ方式

命令の種類はいわゆるＲＩＳＣ型である。命令はひとつのＶＬＩＷ命令を４フィールドに分けた各フィールドに配置される。ＶＬＩＷ方式の特徴をいかした命令語長の長い命令もあり、ＶＬＩＷ命令のなかで２フィールドまたは４フィールドにまたがって配置される。ひとつのＶＬＩＷ命令の命令長を１６バイトとコンパクトに構成するため各フィールドは演算器にオーバラップして割り当てられている。

○　命令の種類

命令には以下の種類がある。

- ロード・ストア命令
- 固定小数点演算命令
- 論理演算命令
- 浮動小数点演算命令
- 事務用データ処理命令
- 制御命令
- スタック操作命令
- ファームウェア化命令

オペランドでの主記憶のアクセスはロード・ストア命令、スタック操作命令のみが行える。そのほかの命令は基本的に３オペランド形式のレジスタ・レジスタ方式である。特殊なフォーマットをもつ命令としてはキャリ付加減算命令、シフト命令、データ並べ換え命令、ファームウェア化命令がある。

○　並列実行

1クラスタ内の命令は並列に実行される。フィールド指定にしたがって配置された命令はすべて並列実行可能であって、例えば整数演算命令は4並列実行が可能であり、浮動小数点演算は除算が1、加減算が2、乗算が1の並列度をもつ。またロード命令・ストア命令・分岐命令の3つの命令が並列に実行可能である。

1クラスタ内の複数の命令で同一レジスタ参照が可能である。同一レジスタへの同時書き込みはプログラムエラー割り込み発生となる。

○　フィールド指定

1サイクルに最大4つの命令の実行を指定できる。同時実行される命令は静的に決定され、全体で16バイトの命令群を構成する。これを命令語クラスタと呼ぶ。命令語クラスタはフィールド0～3に分けられ、各命令ごとに配置可能なフィールドが決まっている。32ビット即値命令や5オペランド命令は2つのフィールドにわたって配置される。ファームウェア化命令は1クラスタで1命令である。

（図1）

フィールド0	フィールド1	フィールド2	フィールド3
整数ロード 浮動小数点ロード		整数ストア 浮動小数点ストア	
整数ロード	浮動小数点ロード	整数ストア 疑似ストア	浮動小数点ストア
固定小数点加減算 固定小数点テスト バイトデータ比較	固定小数点加減算 固定小数点乗算 固定小数点テスト バイトデータ比較	固定小数点加減算 固定小数点テスト バイトデータ比較	固定小数点加減算 固定小数点テスト バイトデータ比較
固定小数点加減算 キャリー付き固定小数点加減算 固定小数点テスト		固定小数点加減算 キャリー付き固定小数点加減算 固定小数点テスト	
浮動小数点除算 浮動小数点データ転送 浮動小数点テスト	浮動小数点加減算 浮動小数点データ変換	浮動小数点加減算 浮動小数点データ変換	浮動小数点乗算 浮動小数点データ転送 浮動小数点テスト
論理演算 シフト ビットテスト	論理演算 シフト ビットテスト	論理演算 シフト ビットテスト	論理演算 シフト ビットテスト ビット探索
論理演算 ビットテスト	シフト	論理演算 ビットテスト	シフト
10進加減算 事務用データ比較	事務用データ比較	10進加減算 10進乗算 事務用データ比較	事務用データ比較
キャリー付き10進加減算 データ並べ換え		キャリー付き10進加減算	
フレーム解放 ジェネラルレジスタ回復 浮動小数点レジスタ回復		ジェネラルレジスタ退避 浮動小数点レジスタ退避	
制御レジスタ読出し			制御レジスタ書換え
ファームウェア化命令			

2．2　コンパイラによる高度な並列化の保証

ＶＬＩＷ方式ではコンパイラによる並列スケジューリングが性能に大きく影響する。コンパイル時の解析を容易にするために、命令セット・アドレッシングモード・レジスタは直交性をもっている。パイプラインは均一でありほとんどの場合ハザードは起きない。そしてスケジュールの制約にならないように並列実行に十分なレジスタ数が確保されている。

○　レジスタ構成

整数レジスタが６４個（３２ビット長）、浮動小数点レジスタが３２個（６４ビット長）そのほかに特殊レジスタがいくつかある。整数レジスタ、浮動小数点レジスタはそれぞれ命令セットに直交している。またｒ６１に実行制御フラグ、r/６２にプログラムカウンタ、ｒ６３にゼロがマッピングされており整数レジスタとして読みだし可能である。

○　アドレッシングモード

アドレッシングモードはレジスタ間接と直接の２種類で命令セットに直交している。もっとも単純なアドレッシングモードしかもたない理由は、ロード・ストア命令のパイプライン段数を減らしパイプラインの均一化を図るためである。もうひとつの理由はアドレス計算はコンパイラによる最適化をおこなうため、メモリアクセスと分離して並列化が行えるようにするためである。

○　パイプライン

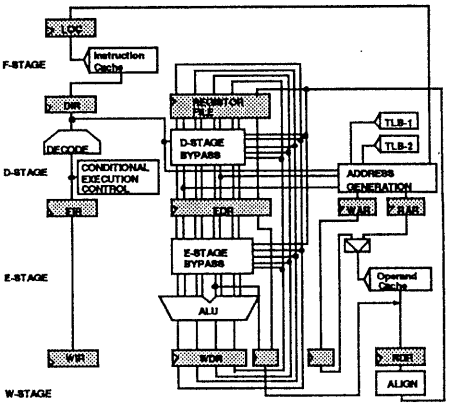
各命令は命令語クラスタ単位でパイプライン実行される。パイプラインはフェッチ、デコード、実行、書き込みの４ステージ構成である。（図２）

固定小数点演算ユニットではパイプラインは均一である。また遅延分岐方式を採用しており、遅延スロットとして１クラスタを実行しハザードは発生しない。

マルチサイクル演算はすべて浮動小数点演算ユニットで実行される。マルチサイクル演算は命令語クラスタとは非同期に実行でき、固定小数点演算のパイプラインを乱さない。（背後実行と呼ぶ。）さらに除算をのぞくマルチサイクル演算はパイプライン実行が可能であり、毎サイクルの投入が可能である。

2．3　低並列度なプログラムに対する性能の保証

非科学技術計算分野のアプリケーション、ＯＳ、関数呼出手続きなどは比較的並列度が低く、平均命令並列度は１～２程度である。このような場合に局所的な命令レベルの並列性を高めるため、命令は並列化されるように単純化されパイプライン段数は小さくなっている。また分岐命令まわりでの並列性を高めるため条件実行制御方式を用いている。



AR : address register
IR : instrnction register
DR : data register

図2　パイプライン構成

○　条件実行制御方式

条件実行制御方式は以下の構成によって実現する。

（１）条件比較命令

指定条件によって比較を行い、その真偽値を実行制御フラグに格納する命令。格納する実行制御フラグ番号は明示的に指定される。

（２）実行制御フラグレジスタ

全１６ビットのレジスタで、各ビットは条件比較命令によって設定され、後続の命令の実行選択を決定する。整数レジスタｒ６１にマッピングされており、整数レジスタとしても参照可能である。実行制御フラグ１５番は常に０である。

（３）各命令につけられた実行制御フラグ番号と真偽値

指定実行制御フラグの値と指定された真偽値とが一致する場合にのみその命令は実行される。（図３）

とくに条件分岐は分岐命令の条件実行制御によって実現される。また比較命令の条件実行制御によって複合条件も生成できる。さらに複雑な条件はｒ６１参照によって生成することもできる。

○　浮動小数点データ転送命令

６４ビットの浮動小数点レジスタは３２ビットづつ上位と下位にわけて、整数レジスタとのあいだで直接転送可能である。固定小数点演算ユニットと浮動小数点演算ユニットのあいだの転送命令をもつことにより、ロード・ストア命令で転送を行う場合よりも並列度が向上する。

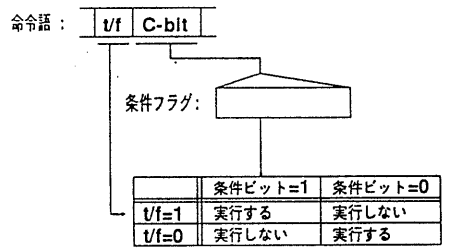
○　スタック操作命令

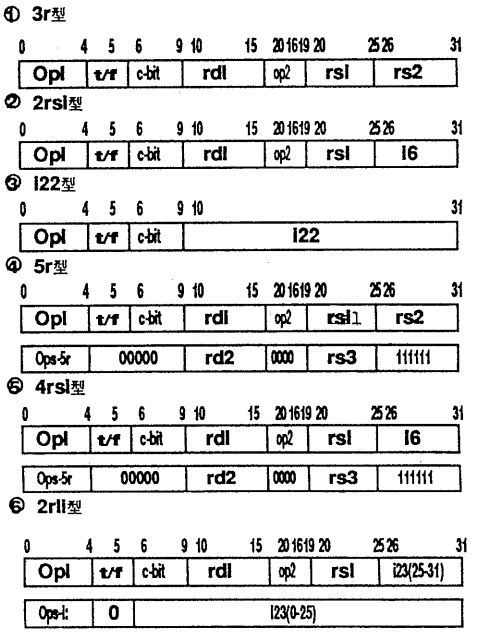
レジスタ退避・復元手続きは関数入口・出口でスタックフレームの確保・解放のみを行ない、実際の退避・復元はそのほかの命令と並列に実行できる。

２．４　ＣＩＳＣ型の従来機との互換性

制御用途などのリアルタイムシステムや即応性の要求されるサーバマシンとしての要求から、ＣＩＳＣ形の従来機ではアーキテクチャレベルで高機能なプリミティブを実現していた。ＶＬ２０００ではより一般的なプリミティブを用意して、これらと同等の機能を実現している。

(図3)条件実行制御方式と命令語の形式





op1 , op2, ops-5r, ops-ll ：命令コード
t/f, c-bit：条件実行制御フラグ指定
rs1, rs2 , rs3：ソースレジスタ番号
rd1, dr2：ディスティネーションレジスタ番号
I6, I22, I32：即値

○　ファームウェア化命令

ＣＩＳＣ形の１命令に相当する命令である。ファームウェア化命令が実行されるとプロセッサはファームウェアモードとよぶ”超特権”モードとなり、ファームウェアロム内にある対応するルーチンを実行する。このルーチンは通常のプログラム実行と同様に処理されるが、それが超特権モードで実行されるためファームウェア化命令があたかも１命令であるかのように動作する。約１３０種類のファームウェア化命令がある。

○　割り込み制御

命令語クラスタ単位での割り込み制御が可能である。すなわち、命令語クラスタ内の各命令の不分割処理が保証され、任意の命令語クラスタ間での割り込み発生、実行再開が可能である。また割り込み禁止命令につづく任意のクラスタ数を割り込み禁止としてその間発生した割り込みを遅延させることができる。

○　疑似ストア命令

アドレス変換機構のみを動作させて、実際にはストアは実行しない。メモリアクセスを不分割処理したいときにまえもってアドレス変換割り込みなどを発生させるために用いる。

○　マルチプロセッサ

共有メモリ型のマルチプロセッサ構成が可能である。（最大８台）マルチプロセッサ間の排他プリミティブはファームウェア化命令として用意される。

２．５　特徴的な命令

○　トラップ付ロード・ストア

ミスアラインなアドレスへのロード・ストアが可能。これはミスアラインなアクセスがあったときにトラップし、ファームウェアモードでアラインにあわせたアクセスを複数実行することで実現する。

4000520	fldt([%r7],%f9)	nop	nop	fmul(%f1,%f0,%f6)
4000530	add(%r4,#0x40081f58,%r7)		nop	fmul(%f3,%f9,%f9)
4000540	add(%r4,#0x40081f60,%r8)		nop	fmul(%f3,%f10,%f7)
4000550	add(%r4,#0x40081f50,%r6)		fadd(%f8,%f5,%f8)	fmul(%f1,%f10,%f5)
4000560	fldt([%r2],%f10)	nop	nop	fmul(%f3,%f0,%f0)
4000570	fldt([%r7],%f9)	nop	fadd(%f6,%f9,%f8)	fmul(%f8,%f10,%f6)
4000580	fldt([%r8],%f5)	fadd(%f5,%f0,%f0)	nop	nop
4000590	fldt([%r6],%f7)	nop	fadd(%f2,%f7 ,%f5)	fmul(%f8,%f5,%f2)
40005a0	add(%r4,#0x40080018,%r6)		nop	fmul(%f0,%f9,%f9)
40005b0	add(%r4,#0x40080008,%r8)		nop	fmul(%f5,%f7,%f7)
40005c0	add(%r4,#0x40080010,%r7)		fadd(%f4,%f9 ,%f9)	nop
40005d0	nop	fadd(%f4,%f2,%f6)	fadd(%f4,%f6 ,%f2)	nop
40005e0	nop	fadd(%f4,%f7,%f7)	fstt(%f9,[%r7])	add(%r3,#%04,%r3)
40005f0	tstlt(%r3,#0x000003e5,@1)		fstt(%f6,[%r6])	nop
4000600	[1f]tstlt(%r3,#0x000003e9,@0)		fstt(%f7,[%r8])	[1t]branch(#0x400004c0)
4000610	add(%r0,#%20,%r0)	add(%r2,#%20,%r2)	fstt(%f2,[%r0])	add(%r4,#%20,%r4)

(図4)　オブジェクトコードの例

○　キャリ付加減算、５ｒ型シフト命令

これらの命令は固定のキャリを使って行うのではなく、キャリ値を格納するレジスタもオペランドで指定する５オペランド形式となっている。固定のレジスタをもたないことによって並列性の向上を図っている。

○　バイトデータ比較命令

４バイトデータの各バイト単位に比較を行う命令。この命令を４フィールドすべてに配置すれば１サイクルで１６バイトの比較ができることになり、文字列操作でのデリミタ処理などに有効である。

○　事務用データ処理命令

パック１０進数の演算をサポートした。さらにパック１０進演算を高速化するためのデータ並べ変え命令がある。

３．並列性向上へのアプローチ

○　アドレス計算

アドレス計算はスカラ倍と加減算からなっているので、コンパイラがこれをシフトと加減算の組合せとして結合法則・分配法則を適用し高度に並列化することができる。そしてこれらの固定小数点命令は１サイクルで実行でき、全てのアドレス計算を固定小数点命令として処理すれば並列性がより向上する。この様な理由からロード・ストアのアドレッシングモードは絶対とレジスタ間接のみとした。

○　遅延スロット

分岐命令は遅延分岐となっていて遅延スロットは１クラスタであるが、これはフィールドで考えると、分岐命令のあるクラスタも含めて７フィールドの遅延スロットがあることに相当する。一般にはこれらのフィールドすべてに分岐前の命令を配置することは困難であるので、空いた

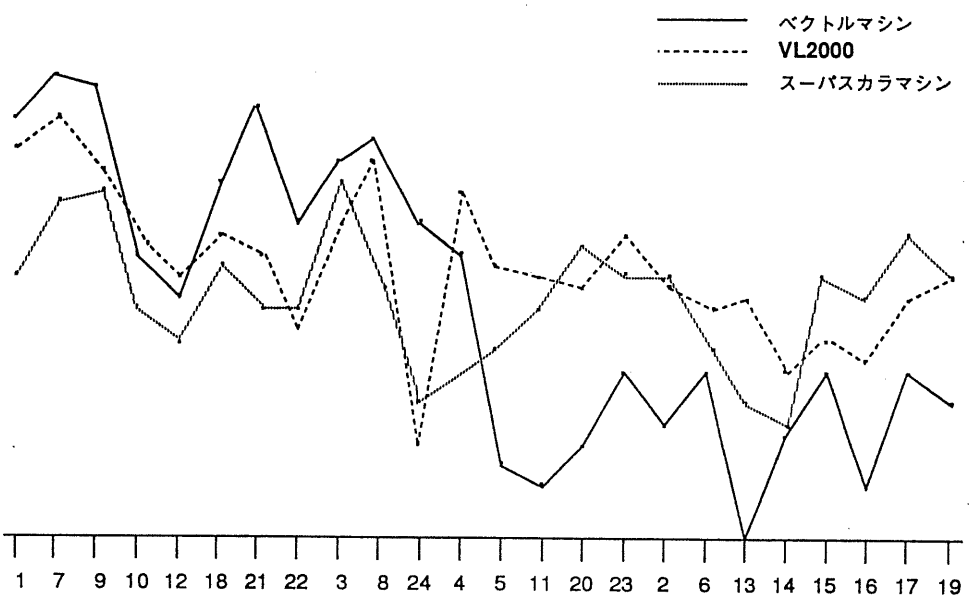


図5

フィールドには分岐先の命令または後続の命令を条件実行指定して配置する。この方法は分岐確率が半々であるような場合でも十分な効果が期待できる。

○　条件実行展開

ｉｆ－ｔｈｅｎ－ｅｌｓｅ形の制御構造は条件実行制御を用いれば分岐命令を必要としない。この方法を条件実行展開と呼んでいる。展開される命令数が少ないときに条件実行展開すれば前後の命令とのスケジュールが可能になり性能が大幅に改善する。また条件分岐の連続は、条件実行制御によって生成された複合条件での条件分岐とすることができる。

○　評価

ＶＬＩW方式ではコンパイラによる並列スケジュールが性能を大きく左右する。従ってここで評価に用いるものは実際にコンパイラによって生成した。プログラムは Livermore Fortran Kernel である。

図５に評価結果を示す。ＬＦＫを比較的並列性が得られ易いものから順に並べてある。縦軸は実行時間をマシンサイクルによって正規化し対数スケールとした。対象としてベクトルマシンとスーパスカラマシンによる結果も示した。参考としてオブジェクトコードの一部を図４に示す。

○　まとめ

並列アーキテクチャであるＶＬＩW方式はコンパイラによって並列性の抽出を行い、命令レベルの並列化を静的に指定する方法であり、ベクトル方式が行うループの並列化、スーパスカラ方式が行う命令レベルの並列化の双方の並列化が適用できる。

従ってＶＬＩWマシンでは局所的な並列性の抽出によって平均性能の下限が決まり、大域的な並列性の抽出によってピーク性能が決まると考えられる。そしてこれらの並列性がどの程度存在するかはアプリケーション、言語、コンパイラ、アーキテクチャによって大きく異なっており、とくに大域的な並列性はアプリケーションと言語により深く依存し、逆に局所的な並列性はコンパイラとアーキテクチャにより深く依存する傾向にある。

以上の観点からＶＬＩWマシンにおいては、科学技術計算のアプリケーションのように大域的な並列性が顕著であり、ループアンローリングやソフトウェアパイプライニングなどの方法で十分な並列性が得られる場合はもちろん、そのような並列性がない場合でもコンパイラやアーキテクチャの改良によって局所的な並列性を確保し性能を保証することが可能であると考えられる。

参考文献

(1) D.Patterson, J.Hennessy, COMPUTER ARCHITECTURE A QUANTITATIVE APPROACH, Morgan Kaumann Publishers, 1990

(2) J.Ellis, BULLDOG A COMPILER FOR VLIW ARCHITECTURES, The MIT Press, 1986

(3) F.Chow, A PORTABLE MACHINE-INDEPENDENT GLOBAL OPTIMIZER-DESIGN, Ph.D. disser. Stanford University, 1984

(4) 石川禎　ＶＬＩWアーキテクチャの実現　情報処理学会第46回全国大会6L-1, 1993

(5) 竹内陽一郎　微視的並列度向上のための中間コード最適化戦略　情報処理学会第43回全国大会5-209, 1991

(6) 森良哉　ミニコンピュータ・ワークステーションの命令セットアーキテクチャ　情報処理29, 12, pp.1412-1419, 1988