Vol.2014-ARC-208 No.3
2014/1/23

MIPS命令パイプラインベースの簡易VLIWプロセッサ

平石 康祐[1,a)]　田中 耕司[1,b)]　大津 金光[1,c)]　大川 猛[1,d)]　横田 隆史[1,e)]

概要：MIPS パイプラインプロセッサをベースに，2 命令同時実行可能な VLIW プロセッサを開発した．本 VLIW プロセッサは 2 つの MIPS 命令をつないで一つの VLIW 命令として扱い，下位アドレス側 32 ビットをメインパイプ，上位アドレス側 32 ビットをサブパイプにより実行する．これにより，ハードウェアの設計コストを抑えながら命令レベル並列処理が可能なハードウェアを実現した．また，メモリアクセス性能の改善するために，高速にアクセス可能なメモリを FPGA 上に実装した．FPGA ボードを用いた評価を行った結果，オフチップの SDRAM へのアクセスと比較して，ストア命令の必要サイクル数を平均して 94%削減，ロード命令の必要サイクル数を平均して 89%削減した．

1. はじめに

今日のプロセッサの性能向上には，プログラムの並列性が重要となっている．命令レベルの並列実行が可能なアーキテクチャとして主にスーパースケーラアーキテクチャと VLIW アーキテクチャがある．スーパースケーラアーキテクチャは命令間の依存解析により同時処理可能な命令を見つけ出し，複数の命令を同時に処理することで高性能化を達成するアーキテクチャである．しかし，命令間の依存解析や命令の同時発行のためのハードウェアが必要となり，プロセッサの構成が複雑になる問題がある．それに対して VLIW アーキテクチャは，命令間の依存解析と並列実行可能な命令の抽出をコンパイル時に行うため実装に必要なハードウェアコストおよびその開発にかけるコストの低減が期待できる [1][2].

以上のことから，短期間で命令レベル並列性を活用したプロセッサハードウェアを開発する上で VLIW 方式が適していると考えられる．そこで本稿では VLIW 方式を採用したプロセッサを開発する．

2. プロセッサの構成

今回開発したプロセッサは，配布されたリファレンスデザインからの変更を最小限として，VLIW プロセッサを実現したものである．2 つの MIPS 命令パイプライン [3] を使用し，同時に 2 つの命令を処理できる VLIW プロセッサを構成する．

2 つの命令パイプ間は，論理的に一つのレジスタファイルを共有する．また，異なるパイプで実行される命令間でのデータハザードに対応するため，命令パイプ間にフォワーディングパスを通し，互いの実行結果を最小限のサイクル数で利用可能とした．

ハードウェアの簡略化のため，2 つの命令パイプのうちプログラムカウンタを変更できるパイプはメインパイプのみとした．そのため，分岐命令はメインパイプでのみ処理可能である．また，2 つの命令パイプは互いにパイプラインのストール条件を共有しており，クロック単位で同期して動作する．

図 1 に今回開発したプロセッサの構成図を示す．

プロセッサにはレジスタファイルを各パイプラインごとに 1 セットずつ用意した．これらは論理的には一つのレジスタファイルであり，読み出しポート数の増加を目的として，各パイプごとにレジスタファイルを複製して備えている．2 つのレジスタファイルの内容は，同時に同じ値が両方のレジスタファイルに書き込まれることで一貫性が保証される．

開発したプロセッサとリファレンスデザインの相違点は以下のようになっている．

(1) デュアルパイプライン構成の VLIW プロセッサとし，命令レベル並列実行を可能とした．
(2) 32 ビットの MIPS 命令 2 つをひとまとめにして 64 ビットの VLIW 命令として扱う．下位アドレス側の MIPS 命令をメインパイプ，上位アドレス側の MIPS 命令をサブパイプで実行する．

[1] 宇都宮大学
Utsunomiya University
[a)] kosuke@virgo.is.utsunomiya-u.ac.jp
[b)] koji@virgo.is.utsunomiya-u.ac.jp
[c)] kim@is.utsunomiya-u.ac.jp
[d)] ohkawa@is.utsunomiya-u.ac.jp
[e)] yokota@is.utsunomiya-u.ac.jp

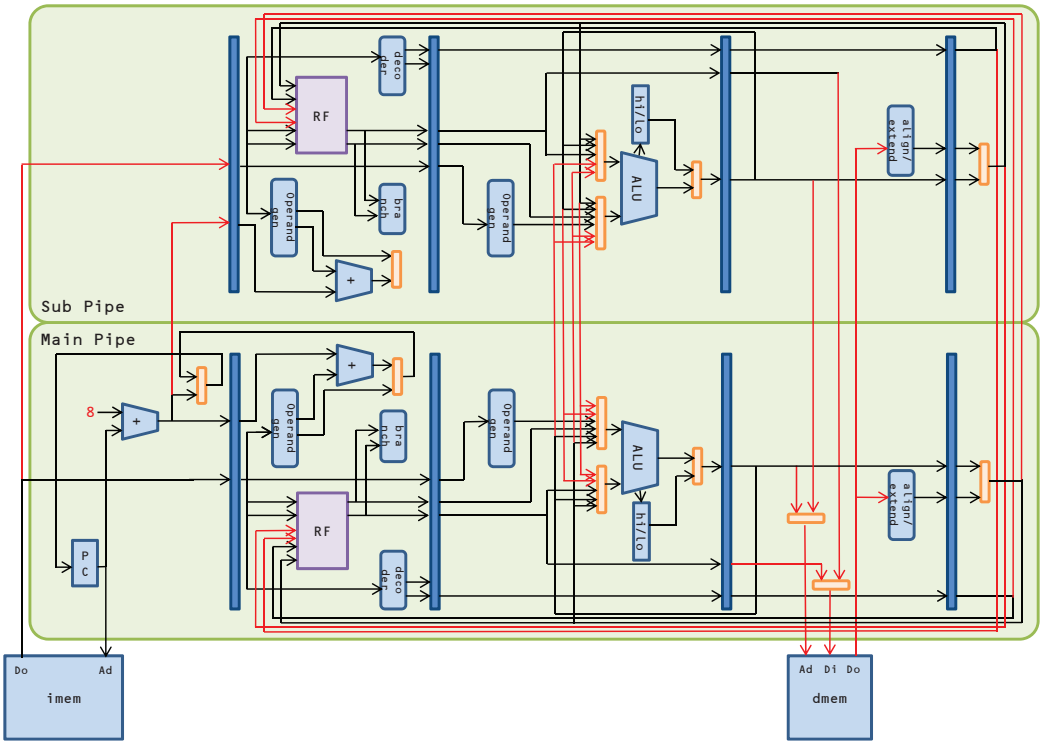

図 1 開発したプロセッサの構成図

(3) 命令長の変更に伴いプログラムカウンタの扱いを 4 バイト単位から 8 バイト単位に変更した．
(4) メインパイプとサブパイプでプログラムカウンタの値を共有するように変更した．
(5) 2 つの命令パイプがクロック単位で同期して動作するために，各パイプのストール条件を共有して使用するように変更した．
(6) ロードストア命令を両命令パイプで行うことができるようにするため，両パイプとデータメモリをセレクタを経由して接続した．ただし，ハードウェアの簡略化のため，同一 VLIW 命令内では片方の MIPS 命令のみがメモリアクセスを行うことができる．
(7) 2 つの命令パイプの実行結果を同時に書き込むことができるようにするため，レジスタファイルの書込みポート数を 2 つに変更した．
(8) 両パイプが備えるレジスタファイルを互いに論理的に共有させるために，互いの実行結果を自分側と相手側のレジスタファイルに同時に書き込むようにした．
(9) 異なる命令パイプで実行されている命令間でのデータハザードを解消するために，パイプ間にフォワーディングパスを追加した．
(10) 頻繁に行われるデータメモリアクセスを高速化するために，スタック領域などのデータアクセスの用として FPGA のブロック RAM を使用したオンチップのデータメモリを追加した．

2.1 VLIW 命令

今回開発したプロセッサは，命令長 32 ビットの MIPS 命令 2 つをまとめて 1 つの 64 ビット VLIW 命令として入力し，下位アドレス側の MIPS 命令をメインパイプ，上位アドレス側の MIPS 命令をサブパイプで並列実行する．そのため，プログラムカウンタを 8 バイト単位で扱うように変更を行った．これにより，8 の倍数アドレスの MIPS 命令がメインパイプ，8 の倍数＋ 4 のアドレスの MIPS 命令がサブパイプで実行される．

これにともない，ホスト側とのインタフェースを変更する必要がある．命令メモリ (imem) への書き込みは 64 ビット単位で行うが，データメモリ (dmem) は 32 ビット単位で行うため，両者でデータの書込み幅が異なる．そのため，修正が必要となる．シリアル通信でホスト側からアプリを転送する際に，FPGA 上のプログラムローダ回路は，4 バイト単位で FPGA ボード側のメモリに書き込むことを想定した設計になっているため，命令コード領域へは 8 バイト単位で書き込むように設計を変更した．なお，それ以外のデータ領域へは従来通り 4 バイト単位でデータが書き込まれる．

2.2 命令パイプ間の同期

VLIW プロセッサとして動作するためには，2 つの命令パイプがクロック単位で同期して動作する必要がある．実行に複数サイクルかかる命令の場合は，両方の命令パイプを止めるように制御する必要がある．そのため，各パイプ

ラインのストール信号を互いに相手側にも送り，相手側パイプがストールする場合に自身側のパイプもストールするように制御した．これにより，クロック単位での同期動作を実現した．

2.3 データメモリへのロードおよびストア

今回開発した VLIW プロセッサでは，2 つの命令スロットのうちどちらでもロード・ストア命令を実行可能とした．そのため，メインパイプとサブパイプ共にロード命令やストア命令を実行する可能性があるが，dmem は書込みポート，読み出しポートともに 1 つしかないため，同時にアクセスすることはできない．この問題については，メインパイプ，サブパイプのどちらか一方のアクセス要求をセレクタ回路により選択し，dmem に送るようにした．ストア命令，ロード命令を同時に実行しないよう，ソフトウェアレベルでの保証を行う．また，ハードウェア回路の簡略化のため，誤って一つの VLIW 命令内に 2 つのロードストア命令が含まれていた場合は，メイン側のみを実行し，サブ側は無視される．

2.4 レジスタファイル

今回開発した VLIW プロセッサは最大 2 命令同時実行を可能とするため，レジスタファイルは全体で 4 つの読み出しポートと 2 つの書込みポートを備える必要がある，これを実現するため，読み出しポート 2 ＋書込みポート 2 のレジスタファイルを各命令パイプごとに 1 セットずつ用意する．これにより各パイプ毎に 2 つの読み出しポートが利用可能になる．また，各レジスタファイルはの 2 つの書込みポートにより両方の命令パイプの実行結果を毎サイクル書き込むことが可能となる．これによって，2 つのレジスタファイル間で値の一貫性を維持することができ，論理的に一つのレジスタファイルが実現できる．

2.5 フォワーディングパス

リファレンスデザインの命令パイプラインではフォワーディングパスが備わっているため，同一パイプ内での命令間でのデータフォワーディングはできるが，異なるパイプ内にある命令間でのデータフォワーディングはできない．そこで，各パイプ間にフォワーディングパスを追加し，パイプラインストールの機会を減らす．これにより，異なるパイプ内の命令間であっても直前の命令の結果を受け取ることができ，命令スケジューリングに関する制約が大幅に緩和される．

2.6 FPGA 上のデータ用メモリ

リファレンスデザインでは，0x07FFFFFF アドレス周辺のメモリ領域はスタック領域として使用される．スタック領域のメモリは関数呼び出しの戻り先番地やスタックポインタおよびフレームポインタの退避および復帰，レジスタに乗り切らない局所変数のために使用される．そのため，そのアクセス性能を改善することは全体の処理性能の向上に大きく貢献する．そこで，アドレス 0x07FFFFFF から 0x07FE0000 間のメモリアクセスを FPGA 上にブロック RAM を使用して開発したデータメモリにアクセスするように変更を行った．これにより，リファレンスデザインでは平均して約 18 サイクルかかっていたロードストアの処理がストア命令で 1 サイクル，ロード命令で 2 サイクルに短縮した．

3. ソフトウェアへの制約条件

今回開発したプロセッサには，ソフトウェアに対して，以下の 2 つの制約が存在する．

(1) 1 つの VLIW 命令には最大 1 つのロード・ストア命令を含めることができる．
(2) 分岐命令はメインパイプでのみ実行できる．

3.1 ロードストア命令

データメモリ (dmem) は書込みポートと読み出しポートをそれぞれ 1 つずつしか持たないため，ロード・ストア命令が同一 VLIW 命令中に 2 つ含まれる場合，アクセス要求が衝突する．そのため，メインパイプとサブパイプでロード・ストア命令が実行された場合，どちらか一方のアクセス要求をセレクタ回路により選択し，データメモリにアクセスするようにした．また，ソフトウェアレベルでどちらか一方のパイプのみがロード・ストア命令を行うように保証する．同一 VLIW 命令内に 2 つロード命令またはストア命令が含まれていた場合，メイン側の命令のみ実行され，サブ側は無視される．

3.2 分岐命令

同時に 2 つの分岐命令を処理する場合，その分岐先を決定するロジックが非常に複雑となり，プログラムカウンタの更新がクリティカルパスとなる可能性がある．そこで同時に 2 つの分岐命令を実行しないように制限した．

今回は，分岐命令をメインパイプでのみ実行するように制限し，ハードウェアの簡略化を図った．そのため，ソフトウェアでメインパイプ側にのみ分岐命令が含まれるように保証する必要がある．もしサブパイプ側に分岐命令が現れた場合，命令は無視される．

4. 評価

今回開発したプロセッサは，ロード・ストア命令は複数サイクルかかる可能性があり，実行性能向上の妨げとなる．しかし，FPGA 上にブロック RAM を使用して作成したデータメモリにアクセスするように変更したことで，設定したアドレス領域へのアクセスにかかるサイクルを削減す

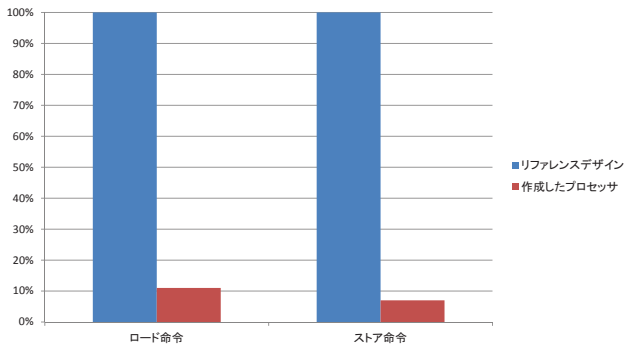

図 2 FPGA オンチップメモリによるアクセス性能の改善

ることができる．

オフチップの SDRAM に対するストア命令はその実行に 4 サイクルから 23 サイクルかかるが，FPGA 上のメモリに対するストア命令は，1 サイクルで実行することができる．また，ロード命令はオフチップ DRAM に対しては 12 サイクルから 20 サイクルかかるが，FPGA 上に対するアクセスであれば，2 サイクルで実行することができる．ロード・ストア命令にかかるサイクルを 18 サイクルとすると，FPGA 上のメモリに対するストア命令は 1 サイクルとなるので，1 命令あたり 94%のサイクル削減，ロード命令が 1 命令あたり 89%のサイクル削減できることを確認した．図 2 は，ChipScope[4] を使用して波形を観測し，オフチップ SDRAM に対するデータアクセスのレイテンシと FPGA 上のオンチップメモリに対するデータアクセスのレイテンシを比較し，そのサイクル数を削減した割合である．FPGA 上のオンチップメモリへのロード・ストア命令の実行サイクル数が大幅に削減されていることが分かる．

5. まとめ

MIPS パイプラインプロセッサをベースにした VLIW プロセッサの開発を行った．VLIW プロセッサの性能を引き出すためには命令レベル並列性の向上が重要となる．今回開発したプロセッサでは，2 つの命令パイプライン間にフォワーディングパスを追加した．これにより，異なる命令パイプで実行されている命令間でのデータハザードを解消することが可能となり，命令レベル並列性を向上することができた．

また，FPGA 上にデータ用のオンチップメモリを確保することで，そのメモリ上の値にアクセスする場合に，大幅な実行サイクルの削減が可能となった．

命令パイプをさらに追加し，依存関係やソフトウェアへの制約による並列性の低下を可能な限り抑えるようにソフトウェアを作成するなどの工夫によりさらにの並列度の高いプログラムの実行が可能となる．これにより，プロセッサの大きな性能向上を達成することが見込まれる．

参考文献

[1] 安藤 秀樹: 命令レベル並列処理ープロセッサアーキテクチャとコンパイラー，コロナ社，(2012).
[2] ジョン・L・ヘネシー，デイビッド・A・パターソン 著 中條 拓伯 監訳・訳, 天野 英晴，吉瀬 謙二，佐藤 寿倫 訳: ヘネシー&パターソン コンピュータアーキテクチャ 定量的アプローチ 第 4 版, 翔泳社, (2008).
[3] MIPS Technologies, Inc.：MIPS software user's manual, Document Number MD00016, Revision 01.17, (2002).
[4] JAPAN XILINX：ChipScope Pro 14.1 ソフトウェアおよびコア ユーザーガイド，(2012).