Vol.2020-EMB-54 No.1
2020/6/26

組み込みシステムに適した RISC-V ソフトプロセッサの設計と実装

金森 拓斗[1,a)]　宮崎 広夢[1,b)]　吉瀬 謙二[1,c)]

概要：本稿では，組み込みシステムに適した RISC-V ソフトプロセッサを提案する．RISC-V における圧縮命令拡張は，命令のサイズを約 25%程度削減することができるため，組み込みシステムに適している．我々は高い性能を発揮した上で圧縮命令に対応する命令フェッチアーキテクチャを提案し，さらにこのアーキテクチャを用いた RISC-V ソフトプロセッサを提案する．この提案プロセッサを Verilog HDL で実装し，Verilog シミュレーションにより得られる IPC(Instruction Per Cycle) と FPGA への実装により得られる動作周波数及びハードウェア量，プロセッサ性能を関連研究と比較する．

Design and implementation of a RISC-V soft processor targeting on FPGA-based embedded systems

Kanamori Takuto[1,a)]　Miyazaki Hiromu[1,b)]　Kise Kenji[1,c)]

1. はじめに

FPGA の高性能化により，MicroBlaze[1] や Nios II[2] に代表されるソフトプロセッサが搭載されたコンピュータシステムが FPGA 上に実装され，様々な分野で活用されている．ソフトプロセッサが採用する ISA (Instruction Set Architecture) として，オープンアーキテクチャの一つである RISC-V が注目されている．

RISC-V は，RISC (Reduced Instruction Set Computer) ベースの ISA であり，過去の命令セットから得られた教訓を元に，普遍性と広範な拡張性を持つように設計されている．プロセッサの設計者はアプリケーションの要求に従って必要な命令セットを選択することができる．基本整数命令セットに追加できる命令セットとして，乗除算命令セットである M 拡張，単精度浮動小数点命令セットである F 拡張，モダンな OS をサポートするために必要な A 拡張などが定義されている．

32 ビットアドレッシングモードの場合，一部の頻出する 32 ビット命令を 16 ビット命令に置き換える圧縮命令拡張である C 拡張は，プログラムサイズを削減できることから特に組み込みシステムでの活用が期待されている．過去にも 2 種類の命令長の命令をサポートする RISC ベースの ISA は存在したが，それらと RISC-V の圧縮命令拡張が異なる点は，圧縮命令に対応するためのモード切替等の概念がないこと及び全ての命令が 32 ビット境界ではなく 16 ビット境界に整列することである．既存プロセッサを圧縮命令に対応させる場合，命令フェッチアーキテクチャに適切な変更を加えなければ大幅な性能低下を引き起こす．

本稿では圧縮命令に対応した効率の良い命令フェッチアーキテクチャと，そのアーキテクチャを用いて圧縮命令に対応させたソフトプロセッサ RVCoreP-32IC(RVP-c) を提案する．また，この提案プロセッサを Verilog HDL で実装し，IPC(Instruction Per Cycle) と動作周波数及びハードウェア量，プロセッサの性能を関連研究と比較する．

2. 関連研究

RVCoreP[3] は宮崎らが作成した FPGA をターゲットとした 5 段パイプラインの RISC-V ソフトプロセッサである．

[1] 東京工業大学
Tokyo Institute of Technology
[a)] kanamori@arch.cs.titech.ac.jp
[b)] miyazaki@arch.cs.titech.ac.jp
[c)] kise@c.titech.ac.jp

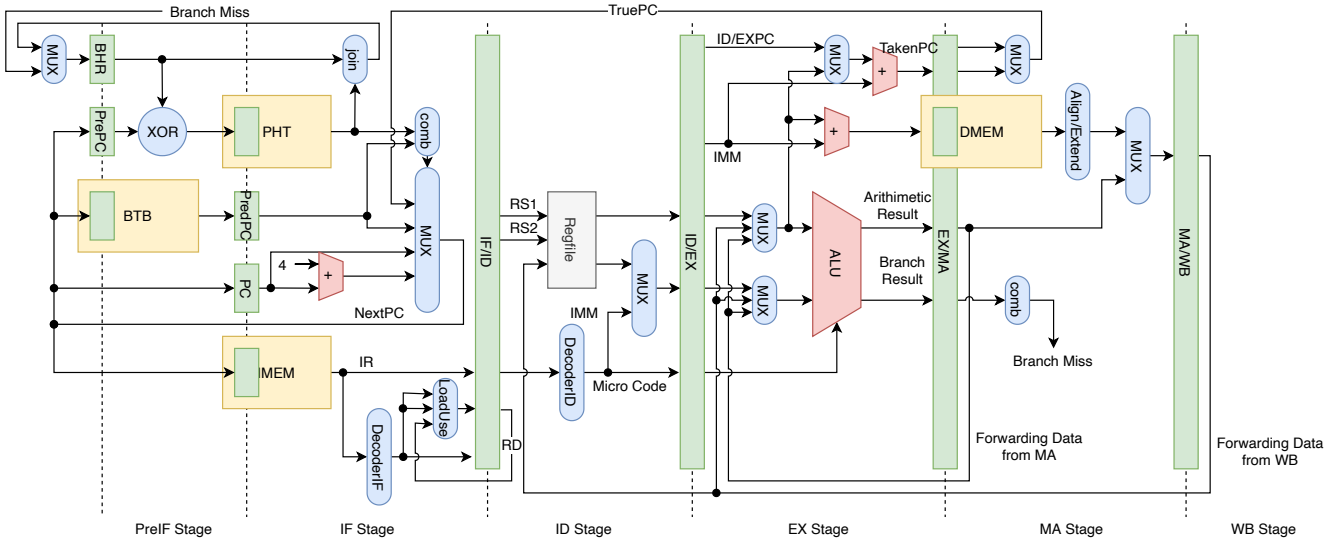

図 1　ベースラインプロセッサ RVCoreP-32I

RV32I に対応しており，Verilog HDL で記述されている．本稿では RVCoreP を圧縮命令に対応させるベースラインとし，圧縮命令に対応していない版を RVCoreP-32I(RVP) とする．

図 1 に RVCoreP-32I のブロック図を示す．緑色の四角はクロック信号の立ち上がりに同期して更新されるレジスタ，黄色の四角が Block RAM で構成されるモジュール，灰色の四角は読み出しを非同期で書き込みをクロックの立ち上がりに同期して行う LUT RAM で構成されるモジュール，赤色のモジュールは加算器や ALU，その他の青色のモジュールは組合せ回路を表す．

RVP は分岐予測に gshare[4] を採用しており，Pattern History Table (PHT) 及び Branch Target Buffer (BTB) の実装には Block RAM を用いている．また，動作周波数を向上させるために分岐予測機構はパイプライン化されている．[5]

圧縮命令に対応する場合に最も考慮しなければならない点は，命令メモリ中に 32 ビット命令と 16 ビット命令が混在し隙間なく配置されることである．図 2 に 32 ビット命令と 16 ビット命令が 32 ビット幅の命令メモリに配置されている状況を示す．以後，1 つの入出力ポートを用いてアクセスできるメモリの幅の単位をエントリと定義する．この図においては 1 エントリは 32 ビットで構成される．説明のためにメモリには 2 バイト毎にアドレスを割り当てる．

例えば橙色で示したアドレス 0x06 と 0x08 に配置される 32 ビットの命令 C は，Entry 1 と Entry 2 に跨って配置される．従って命令 C をフェッチする場合，Entry 1 と Entry 2 の両方にアクセスする必要がある．エントリを跨った 32 ビット命令をフェッチするために，逐次的に 2 回に分けて命令メモリにアクセスすると，プロセッサの IPC が大幅に低下する．

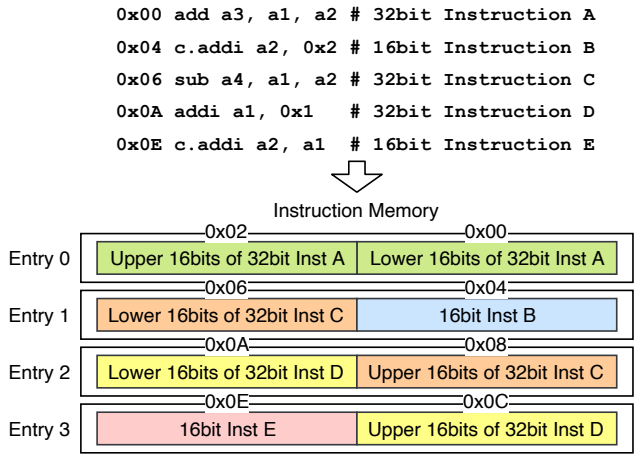

図 2　32 ビット命令と 16 ビット命令が混在する 32 ビット幅の命令メモリの図

この問題を解決するために，圧縮命令に対応する命令フェッチアーキテクチャでは，命令メモリからフェッチしたデータをバッファリングする手法が用いられることが多い．圧縮命令に対応したプロセッサである PULP Platform[6] の RI5CY[7] や Syntacore 社の SCR1[8] では，FIFO を用いたバッファを，VexRiscv[9] では最小限の 16 ビットのバッファを用いている．

しかしバッファを用いた手法では，分岐命令の分岐が成立した場合にバッファの値を消去する必要がある．従って 32 ビット境界に整列していない 32 ビット命令に分岐した場合に，命令の下位 16 ビットはフェッチできるが，命令の上位 16 ビットは同じアクセスでフェッチすることができないため IPC が低下する．

Gary の命令キャッシュアーキテクチャ [10] は CISC (Complex Instruction Set Computer) や VLIW (Very Long Instruction Word) ベースの ISA の可変長命令を効率よくフェッチ可能である．図 3 に RISC-V に最適化した最小構

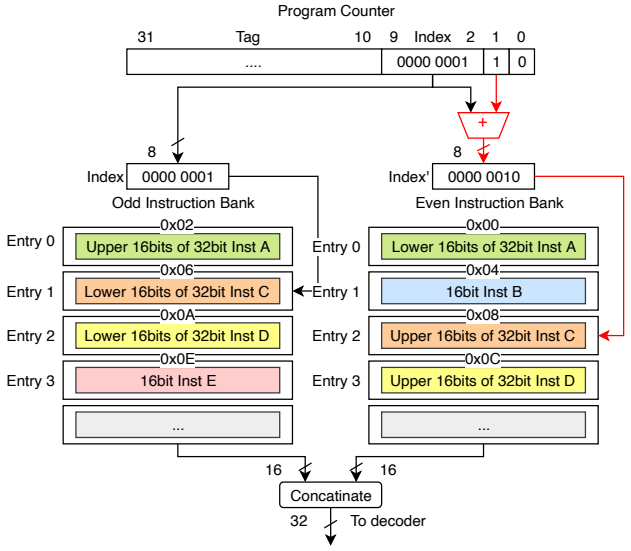

図 3　RISC-V に最適化した最小構成の Gray の命令キャッシュアーキテクチャ

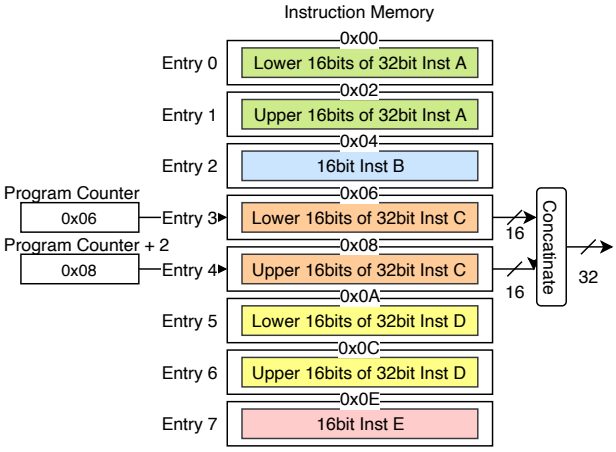

図 4　16 ビット幅の命令メモリへの 2 つのプログラムカウンタを用いた命令フェッチ

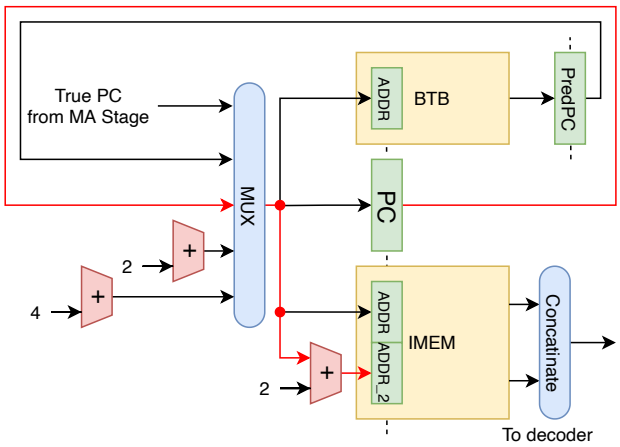

図 5　2 つのエントリへのアクセスを用いた命令フェッチの単純な実装のブロック図

成の Gray の命令キャッシュアーキテクチャで，命令 C がフェッチされる状況を示す．この図のエントリは 16 ビットで構成される．

Gray の命令キャッシュアーキテクチャでは，各命令はアドレスに応じて Odd Instruction Bank と Even Instruction Bank に分けて配置される．それぞれの Bank へ並列にアクセスするため，逐次的な 2 回のアクセスを必要とせず，いかなる命令でも効率よくフェッチできる．

しかしフェッチする命令のアドレスが確定したのちに，その値を加算に用い別のアドレスを計算するため，図中の赤色で示したパスが命令フェッチアーキテクチャのクリティカルパスとなる．このアーキテクチャではこの加算器に 1 を加算するだけの回路を用い，クリティカルパスを緩和している．しかしその回路は ASIC としての実装をターゲットとしており，FPGA への実装は想定されていない．

3. 提案手法

3.1 命令フェッチアーキテクチャ

提案する命令フェッチアーキテクチャでは 2 つのプログラムカウンタを用意し，Gray の命令キャッシュアーキテクチャと同様に 2 つのエントリに同時にアクセスする．ただし，Gray の命令キャッシュアーキテクチャのように命令メモリの分割はしない．FPGA の Block RAM は 2 つの入出力ポートを持つため，1 つの命令メモリの 2 つのエントリに同時にアクセスする．

ベースラインプロセッサの命令メモリの幅は 32 ビットである．この命令メモリの 2 つのエントリに常にアクセスする場合，少なくとも 32 ビットのデータが無駄となるため，命令メモリの幅を 16 ビットに変更する．以後命令メモリの 1 エントリは 16 ビットとする．

図 4 に 16 ビット幅の命令メモリに配置された命令 C を 2 つのプログラムカウンタを用いてフェッチする状況を示す．Program Counter が本来のプログラムカウンタ (PC) であり，Program Counter+2 が本来のプログラムカウンタの指すエントリの次のエントリを指すためのプログラムカウンタ (PC_2) である．提案手法はこれら 2 つのプログラムカウンタを用いて，常に連続した 2 つのエントリにアクセスする．従って 32 ビット境界に整列していない 32 ビット命令に分岐した場合でも，1 サイクルで命令をフェッチすることが可能である．

図 5 に，図 4 で示した命令メモリと 2 つのエントリへのアクセスを用いた命令フェッチをベースラインプロセッサに単純に実装した場合の命令フェッチアーキテクチャのブロック図を示す．この単純な実装は Gray の命令キャッシュと同様に，PC の値を計算したのちその値に 2 を加算して PC_2 の値を計算する．

しかしこのアーキテクチャをベースラインプロセッサに適用した場合，図中の赤で示したパスがクリティカルパスとなり，動作周波数を大幅に低下させる．ベースラインプ

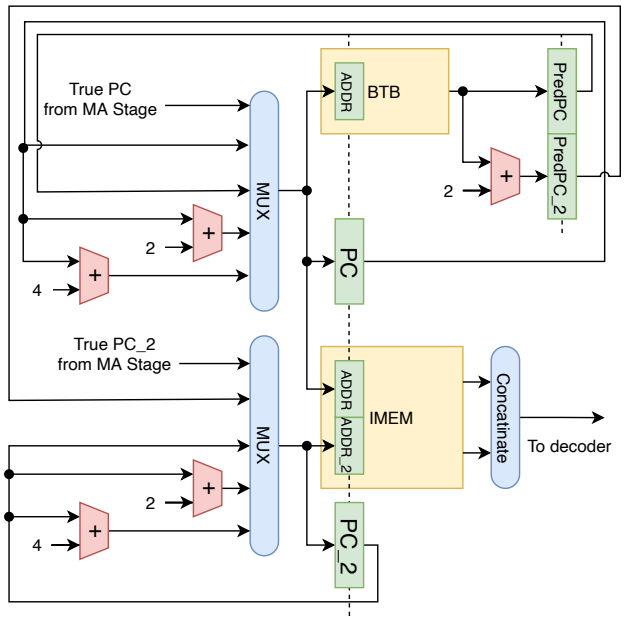

図 6　提案する命令フェッチアーキテクチャ

ロセッサのクリティカルパスは PC の値を計算するパスであるため，このパスへの回路の追加は最小限にしなければならない．

図 6 に提案する命令フェッチアーキテクチャのブロック図を示す．この命令フェッチアーキテクチャでは，PC_2 の値の計算を PC の値の計算と並列に行う．PC の値は IF ステージにおいて 5 つの候補から選択される．提案手法はこれら 5 つの候補全てに 2 を加算した値を予め計算し，PC の値の選択と同じタイミングで PC_2 の値を選択する．

RVCoreP-32IC における PC の値の候補は，現在の PC の値とそれに 2 もしくは 4 を加算した値，BTB から算出された分岐予測先の命令のアドレス (PredPC)，MA ステージから送られる分岐予測ミスを修正するための正しい分岐先の命令のアドレス (TruePC) の 5 つである．

現在の PC の値とそれに 2 もしくは 4 を加算した値のそれぞれに 2 を加算した値は，PC の値の選択ロジックと加算器を複製し，PC_2 の値を保存するレジスタを追加することで計算可能である．

PredPC の値に 2 を加算した値 (PredPC_2) は，BTB の直後に加算器を配置することで計算する．ベースラインプロセッサにおいて PredPC の値は BTB から読み出された直後にレジスタに書き込まれるため，このパスに加算器を追加してもクリティカルパスにはならない．

図 7 に TruePC の値に 2 を加算した値 (TruePC_2) をパイプライン中で計算する回路を示す．TruePC の候補は 2 つある．分岐成立を予測して分岐予測をミスした場合には，分岐命令の命令メモリ上での次の命令の PC (BelowPC) である．分岐不成立を予測して分岐予測をミスした場合には，実際に計算された正しい分岐先の命令の PC (TakenPC) である．これら 2 種類の値に 2 を加算した値をそれぞれ

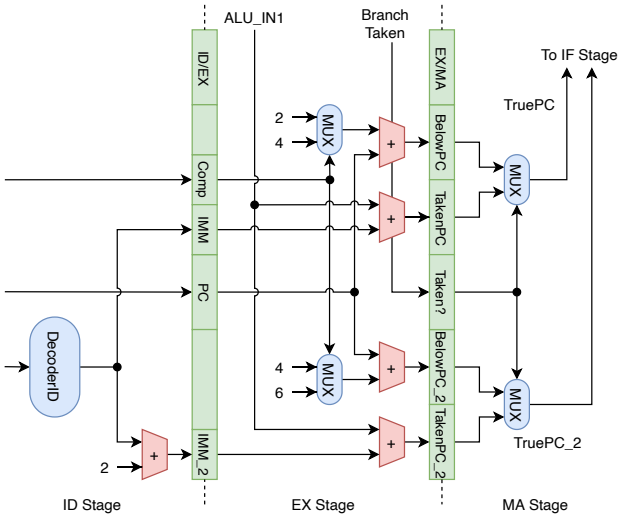

図 7　パイプライン中で TruePC_2 を計算する回路のブロック図

BelowPC_2 及び TakenPC_2 とする．これらの値をプロセッサのパイプライン中で計算する．

圧縮命令に対応したプロセッサにおいて，BelowPC は分岐命令が 16 ビット命令の場合は 2 を 32 ビット命令の場合には 4 を，分岐命令のアドレスに加算した値である．従って BelowPC_2 は，分岐命令が 16 ビット命令の場合は 4 を 32 ビット命令の場合には 6 を，分岐命令のアドレスに加算した値である．この計算は EX ステージにおいて行う．ID/EX パイプラインレジスタ中の圧縮命令であるかを示すレジスタ Comp を用いて 4 か 6 を選択し，分岐命令のアドレスと加算する．

TakenPC は，分岐命令の分岐先のアドレスである．RISC-V の分岐命令は Unconditional Jumps と Conditional Branches の 2 種類に分けられる．Unconditional Jumps の分岐先のアドレスは，分岐命令のアドレスもしくはオペランドレジスタの値のどちらかに命令からデコードされた即値 (IMM) を加算した値である．Conditional Branches の分岐先のアドレスは，分岐命令の PC の値に IMM を加算した値である．

すなわち RISC-V の分岐命令の分岐先は，分岐命令の PC の値と IMM を加算した値もしくはオペランドレジスタの値と IMM を加算した値のどちらかである．このどちらの候補にも 2 を加算した値を計算するために，IMM に 2 を加算した値 (IMM_2) を予め計算する．ベースラインプロセッサの ID ステージにおける即値生成のパスには比較的余裕があるため，Decoder_ID の直後に加算器を配置し IMM_2 を計算する．次の EX ステージで通常の分岐先である TakenPC の計算と並列に TakenPC_2 を計算する．最後に MA ステージで分岐成立を示す EX/MA パイプラインレジスタ中の Branch Taken を用いて，BelowPC_2 か TakenPC_2 を選択し，TruePC_2 として IF ステージに送る．

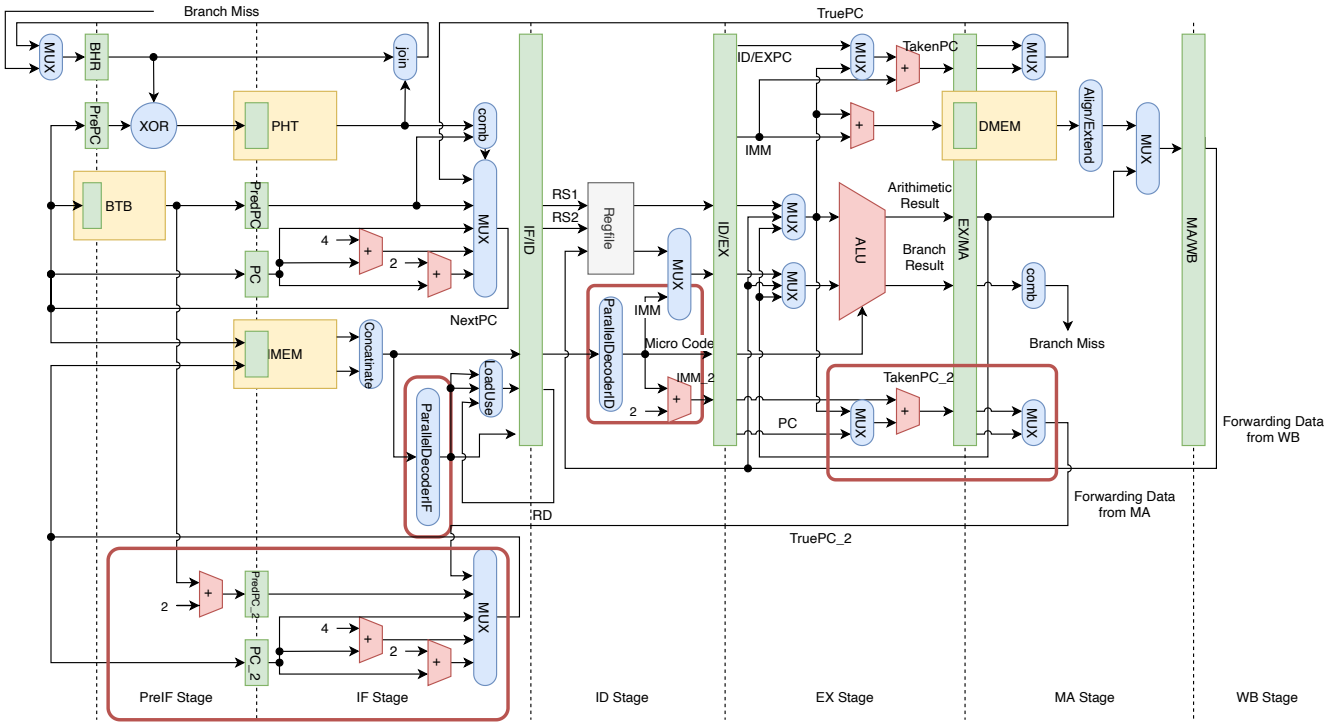

図 8　圧縮命令に対応した提案プロセッサ RVCoreP-32IC

3.2　組み込みシステムに適したソフトプロセッサ RVCoreP-32IC

図 8 に提案プロセッサである RVCoreP-32IC のブロック図を示す．赤色の枠で囲われている部分が RVCoreP-32I からの変更箇所である．提案した命令フェッチアーキテクチャの他にもデコーダと分岐予測機構の制御に変更を加えている．

ロード命令との依存関係の検出を IF ステージで行うために，ベースラインプロセッサである RVP のデコーダは IF ステージの DecoderIF と ID ステージの DecoderID に分割されている．RVP のデコードロジックを圧縮命令に対応させるために，DecoderIF の前に 16 ビット命令を 32 ビット命令に展開する回路を配置すると，このパスがクリティカルパスとなる．提案プロセッサである RVP-c はこの問題を解決するために，16 ビット命令を展開せずに 32 ビット命令のデコードと並列にデコードする ParallelDecoderIF を実装している．

ID ステージでは DecoderID で IMM を生成した後に IMM_2 を計算する．Decoder の前にさらに 16 ビット命令を 32 ビット命令に展開する回路を追加すると，このパスがクリティカルパスになり得る．従って ParallelDecoderIF と同様に，16 ビット命令を展開せずに 32 ビット命令のデコードと並列にデコードする ParallelDecoderID を実装している．

RVP に実装されているパイプライン化された分岐予測機構では，PHT 及び BTB への書き込みに用いるインデックスを生成するために，分岐命令の命令メモリ上での一つ前の命令のアドレスを用いる．圧縮命令に対応する場合，命令メモリ上での分岐命令の一つ前の命令のアドレスは分岐命令のアドレスから 2 または 4 を引いた値のどちらかになる．どちらの値が命令メモリ上での一つ前の命令のアドレスであるかを判別するために，RVP-c は分岐命令のパイプライン上での一つ前の命令の情報を用いる．しかし，分岐命令のパイプライン上での一つ前の命令も分岐命令であり，かつその一つ前の分岐命令が分岐成立と予測されている場合には，パイプライン上での一つ前の命令と命令メモリ上での一つ前の命令が一致しない．この場合に PHT 及び BTB への書き込みを禁止する回路を追加している．

4.　評価

RVCoreP-32IC の動作検証を行うために，Verilog シミュレーションを用いたプロセッサの挙動と，我々が C++ 言語で作成した RISC-V のプロセッサシミュレータである SimRV の挙動を比較した．RVCoreP-32IC の Verilog シミュレーション及び SimRV のどちらも，実行した命令のプログラムカウンタの値，命令レジスタの値，32 本のレジスタファイルの中身を同じフォーマットで出力することができる．各ベンチマークプログラム実行時のこれらのログを比較し一致することを確認した．

RVCoreP の比較対象に RISC-V Foundation 主催の 2018 年 RISC-V SoftCPU Contest[11] で優勝した VexRiscv を用いる．評価に用いた VexRiscv のソースコードは GitHub に公開されており，使用したバージョンは 2019 年 9 月 26 日にコミットされた *Spinal-HDL/VexRiscv@ca228a3* であ

表 1　Verilog シミュレーションによって得られた IPC と分岐予測精度の評価結果

Label	Dhrystone				Coremark				Average IPC
	IPC	prediction hit	prediction miss	hit rate	IPC	prediction hit	prediction miss	hit rate	
RVP	0.934	201,153	16,481	0.924	0.823	363,439	94,534	0.794	0.879
RVP-c	0.921	197,144	20,490	0.906	0.823	363,850	94,123	0.794	0.872
VR-nobp	0.661	N/A	N/A	N/A	0.591	N/A	N/A	N/A	0.626
VR-nobp-c	0.639	N/A	N/A	N/A	0.580	N/A	N/A	N/A	0.610
VR-bp	0.835	146,180	29,452	0.832	0.766	348,010	109,701	0.760	0.801
VR-bp-c	0.808	123,187	29,452	0.807	0.747	324,097	115,320	0.738	0.778

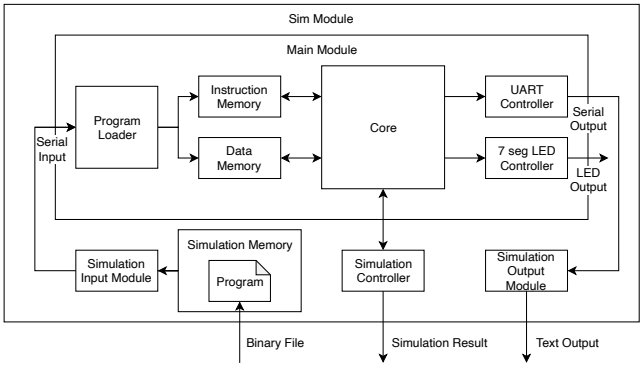

図 9　評価システムの構成

る．予備評価において，VexRiscv の分岐予測機構を搭載する版と搭載しない版の動作周波数と IPC を掛け合わせたパフォーマンス値が同等であったため，それぞれの版を用意した．

RV32I に対応した VexRiscv の分岐予測を搭載したプロセッサを VR-bp，分岐予測を搭載していないプロセッサを VR-nobp とする．また，それぞれ RV32IC に対応したプロセッサを VR-bp-c，VR-nobp-c とする．VexRiscv の各種パラメータは RVCoreP となるべく近い構成になるようにパラメータを設定した．

VR-bp 及び VR-bp-c は，分岐予測機構を有効にするために *prediction* パラメータを *DYNAMIC_TARGET* とした．テーブルのサイズは 512 エントリとした．RVP 及び RVP-c の PHT は 8,192 エントリ，BTB は 512 エントリである．VR-nobp-c 及び VR-bp-c は，*compressedGen* パラメータを *true* と設定し圧縮命令に対応させた．

図 9 に各プロセッサの評価に用いるシステム構成を示す．このシステムは，プロセッサを含み，命令メモリ，データメモリ，シリアル通信の送受信を行うモジュールとシリアル通信用のバッファを持つ．評価するプロセッサを変更する場合は，図中の Core の部分を変更する．FPGA 実機に実装する場合は，図中の Main Module をトップモジュールとして指定し，このシステムにプログラムのバイナリをシリアル通信で送信し，プログラムを実行させる．Verilog シミュレーションで動作させる場合は，図中の Sim Module をトップモジュールとして指定し，シミュレーション上のメモリに読み込んだプログラムを仮想のシリアル通信モジュールを用いて送信し，プログラムを実行させる．どちらの場合でも，同じバイナリファイルを用いて，プログラムを実行することができる．

IPC の評価は，Verilog シミュレーションで行う．RVP，VR-bp 及び VR-nobp は，圧縮命令を含まないプログラムを実行し，RVP-c，VR-bp-c 及び VR-nobp-c は，圧縮命令を含むプログラムを実行する．ベンチマークプログラムは，riscv-gnu-toolchain[12] にて公開されている RV32I 用の RISC-V クロスコンパイラでコンパイルした．用いたコンパイラのバージョンは 8.3.0 で，用いた最適化オプションは-O2 である．圧縮命令を含むプログラムはコンパイラのオプションを指定することでを生成した．ただしこのコンパイラの共有ライブラリのコードは RV32I でコンパイルされているため，リンクされる命令は圧縮されていない．圧縮命令を含むプログラムは，圧縮命令を含まないプログラムと命令の実行順序及び実行命令数が等しい．

ベンチマークプログラムには，Dhrystone[13] 及び Coremark[14] を用いた．Dhrystone のソースコードは riscv-tests[15] として公開されているコードを用いた．ループ回数のオプションである *NUMBER_OF_RUNS* は 2000 に設定した．この場合実行命令数は 909,443 命令である．Coremark のソースコードは RISC-V 向けに公開されたソースコード [16] を用いた．ループ回数のオプションである *ITERATIONS* は 2 に設定した．この場合実行命令数は 1,479,041 命令である．シミュレーションで用いる命令メモリとデータメモリのサイズは，ベンチマークプログラムのサイズから 32KB に設定した．

圧縮前のプログラムのサイズは，Dhrystone は 15.95KB，Coremark は 15.90KB であり，圧縮後では，Dhrystone は 14.40KB，Coremark は 10.28KB である．プログラムのサイズの削減率は，Dhrystone で 9.78%，Coremark で 35.32% である．Dhrystone はリンクされる標準ライブラリに含まれる圧縮されない命令の割合が高いため，プログラムの削減率が低い．

Xilinx Artix-7 FPGA ファミリの xc7a100tcsg324-1 を搭載する Digilent Nexys 4 DDR[17] をターゲットとして最大動作周波数及びハードウェアリソース量を評価する．設計ツールには，Xilinx Vivado 2017.2 を用いた．論理合成には *Flow_PerfOptimaized_high* ストラテジを，配置配線に

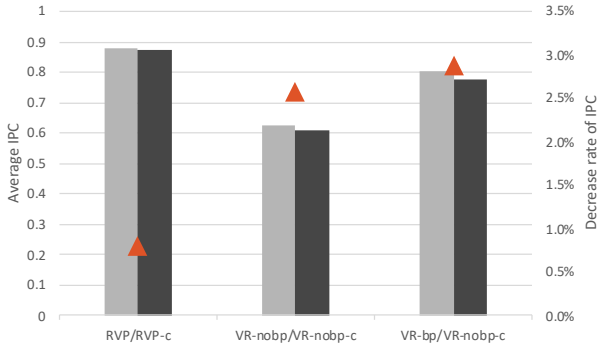

図 10 各プロセッサの平均 IPC と圧縮命令の対応前と対応後で比較した IPC の低下率

は *Performance_ExplorePostRoutePhysOpt* ストラテジを指定した．クロックサイクルを 5MHz 単位で変化させ論理合成及び配置配線を行い，制約を満たす一番高い周波数をプロセッサの動作周波数とした．ハードウェア量は，最高動作周波数の制約での配置配線結果の値を用いた．命令メモリとデータメモリのサイズは，最大動作周波数を測定するために 1 つの Block RAM のサイズである 4KB に固定して評価を行った．評価システムの動作周波数を安定させるため，FPGA の使用するクロックリージョンを一つに限定して配置配線を行った．

各プロセッサの性能をパフォーマンス値として評価する．パフォーマンス値は Dhrystone と Coremark での IPC 値の平均と動作周波数を掛け合わせた値とする．

表 1 に各プロセッサが各ベンチマークプログラムを実行した場合の IPC と分岐予測精度の評価結果を示す．prediction hit は分岐予測の成功回数，prediction miss は分岐予測の失敗回数，hit rate は分岐予測の成功率を示している．RVP-c はプログラムの削減率の低い Dhrystone においては RVP と比較して IPC が低下しているが，プログラムの削減率の高い Coremark ではほとんど変化はなかった．圧縮命令に対応するに当たって，RVP-c の PHT 及び BTB の読み書きインデックスの生成方法を，全ての命令が圧縮命令である場合に最もエントリの有効活用ができるように変更した．従って Dhrystone では分岐予測精度の低下がみられ，Coremark ではほとんど変化がみられない結果となった．VR-nobp-c 及び VR-bp-c のどちらも圧縮命令対応前と比較して平均 IPC の低下が見られた．

図 10 に各プロセッサの平均 IPC と圧縮命令対応後の平均 IPC の低下率を示す．薄い色の棒グラフが圧縮命令に対応していないプロセッサの平均 IPC，濃い色の棒グラフが圧縮命令に対応しているプロセッサの平均 IPC である．赤色の三角点は圧縮命令対応後の平均 IPC の低下率を表している．VR-nobp-c と VR-bp-c のどちらも，分岐先が 32 ビット境界に整列していない 32 ビット命令である場合に 1 サイクルで命令をフェッチできないため，RVP-c と比較して IPC の低下率が高いことがわかる．

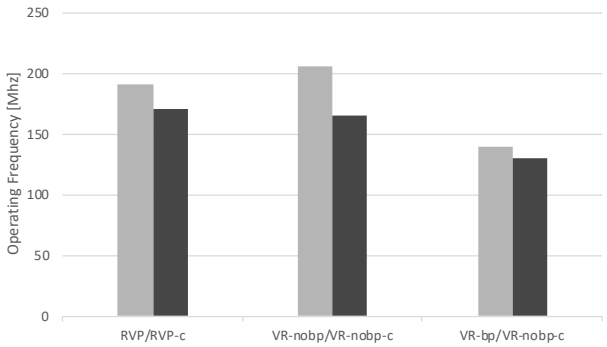

図 11 各プロセッサの動作周波数

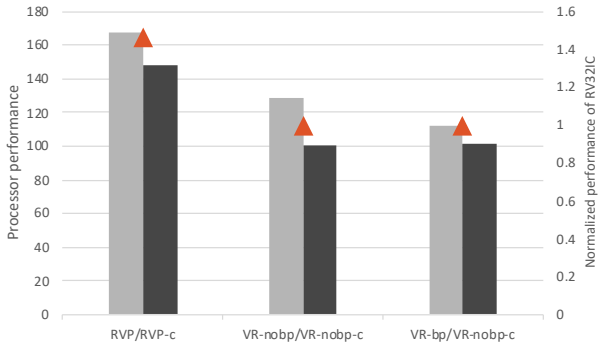

図 12 各プロセッサのパフォーマンス値と VR-bo-c のパフォーマンス値で 1 に正規化された圧縮命令に対応した各プロセッサのパフォーマンス値

表 2 に各プロセッサの動作周波数，ハードウェア量及びパフォーマンス値を示す．Normalized performance は圧縮命令に対応していないプロセッサでは VR-nobp のパフォーマンス値で 1 に正規化された値である．また，圧縮命令に対応したプロセッサでは VR-bp-c のパフォーマンス値で 1 に正規化された値である．Slice LUT，Slice Register 及び Slice の全ての項目において RVP-c が最も資源を消費している．

図 11 に各プロセッサの動作周波数を示す．薄い色の棒グラフが圧縮命令に対応していないプロセッサの動作周波数，濃い色の棒グラフが圧縮命令に対応しているプロセッサの動作周波数である．RVP-c の動作周波数は 170MHz であり，VexRiscv の 2 つの構成で動作周波数の高い VR-nobp-c の 165MHz よりも高い結果となった．VexRiscv は 32 ビット命令に対応したデコードロジックしか持たないため，IF ステージにて 16 ビット命令を 32 ビット命令に展開する．VR-nobp-c のクリティカルパスを解析した結果，VR-nobp-c ではそのパスがクリティカルパスとなり，著しい動作周波数低下を引き起こしていることを確認した．一方 RVP-c は 16 ビット命令のデコードを 32 ビット命令と並列におこなうため，デコードが動作周波数に及ぼす影響を最小限に抑えている．クリティカルパスは RVP と同様に分岐予測機構でのパスである．

図 12 に各プロセッサのパフォーマンス値のグラフを示

表 2　Artix-7 への実装によって得られた各プロセッサの動作周波数とハードウェア量及びパフォーマンスの評価結果

Label	Operating frequency	Slice LUT	Slice register	Slice	Average IPC	Processor performance	Normalized performance
RVP	190	1,073	764	397	0.879	167.01	1.301
VR-nobp	205	936	562	284	0.626	128.33	1.000
VR-bp	140	944	611	300	0.801	112.14	0.874
RVP-c	**170**	1,360	916	490	0.872	148.24	**1.466**
VR-nobp-c	165	1,001	563	310	0.610	100.65	0.995
VR-bp-c	130	1,064	673	363	0.778	101.14	1.000

す．薄い色の棒グラフが圧縮命令に対応していないプロセッサのパフォーマンス値，濃い色の棒グラフが圧縮命令に対応しているプロセッサのパフォーマンス値である．赤の三角は VR-bp-c のパフォーマンス値で 1 に正規化された圧縮命令に対応した各プロセッサのパフォーマンス値である．RVP-c は VexRiscv の 2 つの構成でパフォーマンス値の高い VR-bp-c の約 1.47 倍のパフォーマンス値を達成した．

5. まとめ

本稿では RISC-V 圧縮命令に対応した効率的な命令フェッチアーキテクチャと，その手法を用いて圧縮命令に対応した組み込みシステムに適する RVCoreP-32IC を提案した．

RISC-V 圧縮命令にはコードサイズの削減というメリットがあるが，既存プロセッサを圧縮命令に対応させる場合に性能低下を引き起こす要因を主にプロセッサの命令フェッチアーキテクチャに抱えている．

本稿で提案した命令フェッチアーキテクチャは，常にプログラムカウンタの指す命令メモリのエントリとその次のエントリを並列にフェッチすることで効率の良い命令フェッチを実現する．

評価では，Verilog シミュレーション及び FPGA への実装を行い，提案プロセッサ RVCoreP-32IC と関連研究である VexRiscv を比較した．提案命令フェッチアーキテクチャを用いることで，提案プロセッサは圧縮命令に対応する場合の IPC 低下率を VexRiscv よりも抑えることができた．また，提案プロセッサの性能は VexRiscv の約 1.47 倍の値を達成した．

参考文献

[1] Xilinx: *MicroBlaze Processor Reference Guide*, v2018.2 edition (2018).
[2] Intel: *Nios II Processor Reference Guide* (2018).
[3] H. Miyazaki, T. Kanamori , M. Ashraful Islam and K. Kise: RVCoreP: An optimized RISC-V soft processor of five-stage pipelining, *arXiv preprint arXiv:2002.03568* (2020).
[4] McFarling, S.: Combining branch predictors, Technical report, Technical Report TN-36, Digital Western Research Laboratory (1993).
[5] K. Matsui, M. Ashraful Islam and K. Kise: An Efficient Implementation of a TAGE Branch Predictor for Soft Processors on FPGA, *2019 IEEE 13th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, pp. 108–115 (online), DOI: 10.1109/MCSoC.2019.00023 (2019).
[6] Integrated Systems Laboratory (IIS) of ETH Zürich, Energy-efficient Embedded Systems (EEES) group of the University of Bologna: PULP Platform, https://pulp-platform.org/index.html.
[7] Andreas Traber, Michael Gautschi, P. D. S.: RI5CY: User Manual (2019).
[8] Syntacore: *SCR1 User Manual*, 1.1.0 edition (2019).
[9] SpinalHDL: VexRiscv: A FPGA friendly 32 bit RISC-V CPU implementation, https://github.com/SpinalHDL/VexRiscv.
[10] Vondran Jr, Gary L: Efficient I-cache structure to support instructions crossing line boundaries,US Patent 6,480,938 (2002).
[11] RISC-V Foundation: RISC-V SoftCPU Contest, October 8, 2018, https://riscv.org/2018/10/risc-v-contest/.
[12] RISC-V Foundation: riscv-gnu-toolchain, https://github.com/riscv/riscv-gnu-toolchain.
[13] Weicker, Reinhold P.: Dhrystone: A Synthetic Systems Programming Benchmark, *Commun. ACM*, Vol. 27, No. 10, pp. 1013–1030 (online), DOI: 10.1145/358274.358283 (1984).
[14] EEMBC: CoreMark — CPU Benchmark – MCU Benchmark, https://www.eembc.org/coremark/.
[15] RISC-V Foundation: riscv-tests, https://github.com/riscv/riscv-tests.
[16] UC Berkeley Architecture Research: Setup scripts and files needed to compile CoreMark on RISC-V, https://github.com/riscv-boom/riscv-coremark.
[17] Digilent, Inc.: *Nexys4 DDR Reference Manual*, rev.c edition (2016).