SCIS 2007 The 2007 Symposium on Cryptography and Information Security
Sasebo, Japan, Jan. 23-26, 2007
The Institute of Electronics, Information and Communication Engineers

命令セットの冗長性とプロセッサの多様化

Redundancy in Instruction Set and Diversification of Processors

市川 周一* 澤田 豊志* 畑 尚志*
Shuichi Ichikawa Takashi Sawada Hisashi Hata

あらまし プロセッサアーキテクチャを多様化することにより，ソフトウェアの盗用や解析，実行制御の乗っ取りに対して一定の耐性を持たせることができる．本研究では，特に FPGA を用いた組込みシステムを前提として，プロセッサを多様化する手法について検討する．提案手法では，先行研究で実現されていなかった (真の意味の) 命令セットランダム化が実現できる．さらに 4 つの命令セットの評価結果から，本手法で得られる自由度は先行研究より大きいことがわかった．FPGA 上で実際に設計・評価した結果，RAM を利用した実装では特殊化による論理規模の増加は 3.5%，性能低下も 11.6%に留まることが示された．

キーワード FPGA，組込みシステム，命令セットのランダム化，セキュアプロセッサ

1 はじめに

近年，ソフトウェアの保護は重要な課題になっている．悪意のコピーによる著作権侵害をはじめとして，リバースエンジニアリングによるノウハウの流失，ウイルスなど悪意のプログラムによる攻撃は，いずれも重大な社会問題として認知されている．

Forrest ら [1] は，多様性によりシステムの信頼性を高めることを提案した．システムが均一であるとソフトウェアのポータビリティは向上するが，同時に悪意の複製や解析も容易になる．多くのウイルスはスタックオーバーフロー等を利用してバイナリコードのインジェクションを行なうが，これはプロセッサアーキテクチャを仮定した攻撃である．個々のシステムで命令セットが異なっていれば，単にソフトウェアを複製しても動作しないばかりでなく，命令セット全体の知識なしでリバースエンジニアリングをすることも難しくなる．また，バイナリコードのインジェクションを受けても，感染を免れることができる．

このように多様化には多くの利点があるが，実際には PC やサーバの世界では，プロセッサアーキテクチャの寡占化が進みつつある．ユーザにとって多様化の利益より寡占化の利益 (ポータビリティ) が大きかったことも一因であるが，それ以上に製造側にとってアーキテクチャの寡占化は必然であった．近年のソフトウェアは非常に高い処理能力をプロセッサに要求するようになっており，プロセッサは常に最新のプロセス技術を利用して競争力を保たねばならない．しかし膨大な投資が必要な最先端プロセスを利用するには同一チップの大量生産が必須で，必然的にアーキテクチャの寡占化が促進される構造になっている．

ところが組込みシステムにおいては，少なからず事情が異なっている．組込みシステムは応用によって要求が千差万別であり，本質的に少量多品種生産である．必ずしも高度な処理能力を必要とせず，その一方でコストや消費電力など性能以外の要求事項が多い．部品点数を減らすために，周辺回路をプロセッサと共に集積化した SoC (System-on-Chip) を使う場合もある．結果として，組込みシステムの世界では，用途や特徴に応じて多様なプロセッサが使用されている．

組込みソフトウェアは，一般に製造時にシステムに組込まれて出荷されるため，ポータビリティへの要求は高くない．むしろ，コピー商品やノウハウ流出が大きな問題になる．さらに産業用機器や輸送用機器では，ソフトウェアの改変により重大な結果を招く場合があるため，変造防止技術のメリットは大きい．

近年，組込みシステムの実装には，FPGA (Field Programmable Gate Array) が広く用いられている．FPGA は再構成可能な論理 LSI であり，(2006 年現在) 90〜65nm の最先端プロセスを使用して，1 chip に大規模論理 (100 万ゲート以上)・大容量メモリ (1M バイト以上)・高速演算器などを搭載可能である．FPGA は本質的に単品生産可能なので，組込み用途の少量多品種生産には最適である．FPGA 上にマイクロプロセッサを実現

* 豊橋技術科学大学・知識情報工学系, 441-8580 豊橋市天伯町雲雀ヶ丘 1-1, Department of Knowledge-based Information Engineering, Toyohashi University of Technology, Toyohashi 441-8580, Japan. Email: ichikawa@tutkie.tut.ac.jp

すること (ソフトコアプロセッサ) も広く行なわれているので，FPGA を用いればシステムの多様性を生み出すことは容易である．

本研究では，特に組込みシステムを適用分野に仮定し，命令セットの自由度を利用してプロセッサを多様化する手法について検討する．実現可能な自由度の大小を幾つかのアーキテクチャについて評価し，既存手法と定量的に比較する．さらに FPGA 上で実装評価して，実装コストと性能について定量的に議論する．

2 関連研究

プログラムの改竄を防止する仕組みについては，既に多くの研究がある．ソフトウェア技術としては難読化技術などがあるが，組込みシステムにおいてはハードウェアによる耐タンパ性が必須であると考えられる．

ハードウェアによる耐タンパ性支援については幾つかの先行研究がある．典型的な手法は，実行時にメモリを暗号化してソフトウェアを秘匿する手法と，実行時に MAC (message authentication code) の検証により変造を検出する手法である．XOM (execute-only memory) [2]，AEGIS [3]，SP-processor [4] は，メモリの暗号化を採用している．メモリの MAC を検証してソフトウェアの検証を行なうシステムには，命令ブロックごとに MAC をつけて検証する SPEF (secure program execution framework) [5]，基本ブロックごとの MAC を表にして保存する BBST [6]，FPGA による Secure Hardware Component をシステムに付加して動的にチェックを行なう SAFE-OPS [7] などがある．

動的なメモリ暗号化や MAC 検査は，メモリアクセスの遅延を増大させるため性能オーバヘッドを伴うという欠点がある．この欠点を克服するために，One-Time-Pad を利用して XOM の遅延を掩蔽すること [8][9] や，メモリのプリフェッチと同様な発想でメモリの Predecryption を行なうこと [10] が提案されている．また，SPEF [5] については，命令の投機的実行と同時に MAC 検査を行い，検査結果が合わなければ実行結果をコミットしないという手法 [11] が提案されている．

多様化やランダム化により計算機システムのセキュリティを高めることは，Forrest ら [1] により提案された．また Shacham ら [12] は，主としてソフトウェアによる実現を前提として，アドレス空間のランダム化による利益を検討している．本研究に類似した先行研究としては，命令セットのランダム化によりプロセッサを多様化する手法が提案されている．命令セットのランダム化には色々なレベルや方法があるが，ソフトウェアの解読や複製が難しくなり，インジェクション攻撃が困難になるという利点は本研究と同じである．

Barrantes ら [13][14] は，命令列を擬似ランダム数列と xor してメモリ上に格納し，実行時に元の命令列に戻して実行することを提案した．ここで実際に行なわれているのは**メモリのランダム化**であって**命令セットのランダム化**ではないが，Barrantes らはこの手法を RISE (Randomized Instruction Set Emulation) と名づけた．彼らは既存プロセッサでソフトウェアを実行することを大前提としているので，メモリのスクランブル解除はエミュレータで行なうとした．そのため実行時の性能オーバヘッドが大きい．またハードウェア化に関する検討は全く行なわれていない．一方，本研究ではハードウェアによる命令セット多様化を前提としており，5 章で示すとおりコスト増加や性能オーバヘッドも大きくない．

Kc ら [15] は，ハードウェアによる命令セットのランダム化を検討し，2 つの方法を提案した．1 つは命令流に秘密鍵を xor する方法，もう 1 つは命令ワードのビットを任意に転置する方法である．命令流に秘密鍵を xor する方法は，RISE と同様にメモリのランダム化であって，本質的には命令セットのランダム化ではない．命令ワードを任意にビット転置する手法は，固定長命令であれば命令セットのランダム化といえるが，5 章で示すように本研究の手法より実装コストが大きい．Kc の手法は可変長命令セットに適用することが困難であるが，本研究の手法は可変長命令でも適用可能である．Kc の手法はいずれも単純な換字暗号であるため brute-force 攻撃で解読される恐れがあるが，本研究の手法は 4 章で示すとおり Kc の方法より自由度が大きいので，Kc の手法よりは brute-force 攻撃に強い．

3 本研究の特徴

命令セットのランダム化では，極端な場合，個々のプロセッサでレジスタ構成から命令形式・データ表現まで変更することもできる．しかしその場合，命令セットの定義をランダムに生成し，そこからプロセッサの論理設計と実装を自動生成するシステムが必要になる．さらに，その命令セットをサポートするコンパイラ〜アセンブラ〜ローダ〜ライブラリ〜OS カーネルまで，全てを同時に自動生成しなければならない．その上，生成されたシステムが，一定の信頼性と性能を発揮することを保障しなければならない．これは非常に困難で，(仮に可能であるとしても) コストの高い方法なので，本研究では採用しない．

本研究では，より現実的な代替案として，命令形式を変更せずに符号化の方法だけを変更することにする．例えば，加算命令の命令コード (opcode) が 1 で，減算命令の命令コードが 2 であるプロセッサ A を考える．ここで加算命令と減算命令の命令コードを入れ替えたプロセッサ B を考えたとしても，プロセッサ A と B の命令セットアーキテクチャには何ら本質的な相違は発生しない．単に命令列の表現 (符号化) 方法が異なるだけで，命令列の長さ，データ表現，メモリアクセス方法，条件分

岐の方法，例外処理方法など，全てが同じである．以後の説明で，プロセッサ A と B の相違を，**マシンパーソナリティ**と呼ぶことにする．別な言い方をすれば，A と B は同じ命令セットアーキテクチャから**特殊化**されて生成されたものであるともいえる．

異なるマシンパーソナリティの命令列は，単純な変換規則によって相互に変換することができるので，プロセッサ毎に固有のバイナリコンバータを作成すれば，OS やコンパイラを別途生成する必要はない．最終的なソフトウェアの信頼性も変わらない．プロセッサ設計の立場では命令デコーダの論理が少々変更になるが，論理に本質的な相違はないので，論理規模と動作速度に大きな差はないものと考えられる．(この点は 5 章で実験により検証する．)

一方ハードウェア側から見ると，命令デコーダの論理はパーソナリティ毎に異なる．即ち，プロセッサの論理機能はパーソナリティ毎に異なる．しかし命令コードの写像を RAM/ROM で実装して書き換え可能にすれば，同じハードウェアのままパーソナリティを変更できる．この場合，ハードウェアとしてのプロセッサ設計は 1 つだけあればよいので，製造工程の面で非常に有利になる．さらに RAM/ROM の容量を増やして複数の写像を切り替え可能にすれば，複数パーソナリティを内蔵するプロセッサを実現し，実行時にプロセス毎にパーソナリティを変更することも可能である．プロセス毎にランダム化の方法を変更する機能は，Barrantes [13][14] や Kc [15] も言及しているが，本研究の手法でも同等の機能を実現することができる．

本研究の利点の 1 つは，命令毎にランダム化の有無を決められることである．例えば特権命令だけをランダム化すれば，ユーザプログラムのポータビリティを確保しつつ，OS カーネルのバイナリだけをマシンパーソナリティ依存にできる．これは OS のソフトウェアライセンシングに応用可能である．同様に浮動小数点演算命令だけをランダム化すれば，整数命令による浮動小数点演算エミュレーションは共通に動作可能でありながら，高性能な浮動小数点演算応用はパーソナリティ依存にして利用を制限することができる．もちろん全ての命令をランダム化すれば，あらゆるプログラムがパーソナリティ依存となる．

本研究の方法によるランダム化は，FPGA などのソフトプロセッサに最適であるが，Java VM にも技術的には適用可能であるし，Transmeta 社の Code Morphing [16] のような実行時エミュレーションにも適用可能である．また本研究の手法は，Barrantes ら [13][14] の手法や Kc ら [15] の手法と完全に直交しており，必要に応じて併用することが可能である．メモリ暗号化や MAC 検査と組み合わせることもできる．そうした意味で非常に独立性が高く，有効性の高い手法であるといえる．

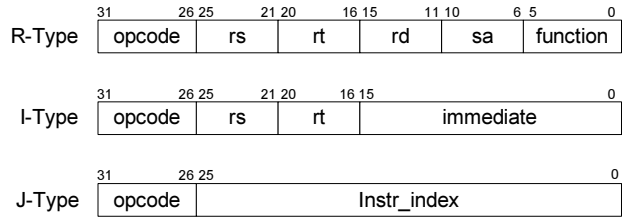

図 1: MIPS instruction format [17].

オペランド指定子の符号化方法や，オペランド指定フィールドの入れ替えによりランダム化することも考えられるが，本研究では扱わないものとする．

4 命令セットの自由度

本章では，1 つの命令セットアーキテクチャから生成可能なパーソナリティの数を検討する．命令セットに自由度が大きければ，大きな多様性を発生させることが可能である．即ち自由度の大きさは，本手法の提供する手法の安全性の尺度となる．また，アーキテクチャの自由度がわかれば，パーソナリティを指定する情報量の下限が判明するので，実装上のコストの目安にもなる．本章の評価には，32-bit アーキテクチャの MIPS [17]，16-bit の SH-3 [18], 8-bit の Intel 8080 [19] と Java VM [20] を取り上げた．

図 1 は，MIPS の命令形式を示したものである [17]. MIPS においては命令コード (opcode) の場所と長さが一定になっているが，命令形式ごとに命令コードの長さが異なる命令セットもある．また，命令コードの一部を用いて命令形式を区別する命令セットもある．そこで本研究では，同じ命令形式の同じフィールドのコード間でのみ，入れ替えを行うことにする．この制限はハードウェア設計の面からみて自然な仮定であり，また複数の命令セットについて同じ基準で自由度を評価するために必要な制限である．[1]

同様に，アーキテクチャ仕様書で未定義あるいは未使用の命令コードは，自由度の計算に含めないものとする．未定義命令は一般に結果が予測不能であるため，ユーザは利用しない．従って命令コードを入れ替えても多様性の増大に実質的な意味を持たない．

各命令セットの自由度と対応する情報量の評価結果を，表 1 にまとめる．さらに表 2 には，基本的な命令クラスごとの自由度の内訳を示した．4 つの命令セットで提供されている命令数に大きな差はないが，自由度には大きな差があることがわかる．命令長が長いほど自由度が大きいわけではない．今回の評価基準では，命令形式が簡単で，ひとつの形式に多くの命令コードが割り当てら

[1] この制限をはずして，命令形式を横断した命令コードの入れ替えを行うことも不可能ではない．その場合，機種にも依存するが，以下の評価結果より大きな自由度を取り出せる可能性はある．

表 2: Redundancy of various instruction classes

	Arithmetic	Data transfer	Branch	Control	Privileged
MIPS	4.68e+21	1.29e+39	4.05e+18	2.00e+00	8.64e+03
SH-3	6.53e+28	9.06e+25	1.15e+03	5.76e+03	2.40e+01
Java VM	2.65e+32	8.50e+101	2.59e+22	1.00e+00	–
8080	2.30e+10	1.55e+25	3.05e+29	4.03e+04	–

表 1: Redundancy in four instruction sets.

	Number of instruction	Redundancy	Information [bit]
MIPS	170	2.34e+166	553
SH-3	188	1.63e+90	300
8080	111	2.34e+136	453
Java VM	201	1.59e+377	1253

れているアーキテクチャの自由度が大きくなる．例えば Java VM では，全ての命令が 1 バイトで命令形式が 1 つしかないため，自由度は $201! \approx 1.59 \times 10^{377}$ と極めて大きくなる．単純な命令セットでも大きな自由度 (安全性) が得られることは，組込み応用では大きなメリットである．

Kc ら [15] は，32 bit の命令ワードを任意にビット転置することでランダム化する手法を提案したが，その自由度は $32! \approx 2.63 \times 10^{35}$ であり，情報量に換算すれば 118 bit 足らずである．本研究の手法も本質的には換字暗号であるが，いずれの命令セットにおいても Kc らのビット転置より大きな自由度をもっており，brute-force 攻撃に対する耐性も大きくなっている．

5 実装と評価

本章では，本研究の手法を実際に FPGA 上で設計・実装し，そのコストと性能オーバヘッドについて評価した結果を示す．評価環境は表 3 にまとめた．実装プラットホームには Xilinx Spartan3 FPGA [21] を採用し，以下の各設計は Xilinx Spartan3 Starter Kit (XC3S200 デバイス搭載) を用いて実機で正常動作することを確認した．

実装対象としては，MIPS 命令セット [17] のサブセットを実装するソフトコアプロセッサ Plasma [22] を採用した．Plasma のブロック図を図 2 に示す．MIPS アーキテクチャは組込み用途でも一定数量利用されており，命令セットも簡潔である．また Plasma はソースが公開されていて自由に利用可能であり，評価に好都合である．

現状の Plasma では 94 命令が宣言されているが，そのうち実際に実装されているのは 62 命令だけで，32 命令は未実装である．これは Plasma の実装上の都合であって，4 章で述べた未定義命令とは異なるので，今回は全 94 命令をランダム化の対象とした．これらの命令について Plasma の自由度を計算すると 3.36e+112 となる．

表 3: Evaluation environment.

CPU	AMD Athlon XP 2500+
Memory	1GB
OS	Windows XP SP2
CAD software	Xilinx ISE 8.2i
Target device	XC3S2000 -4 FG456

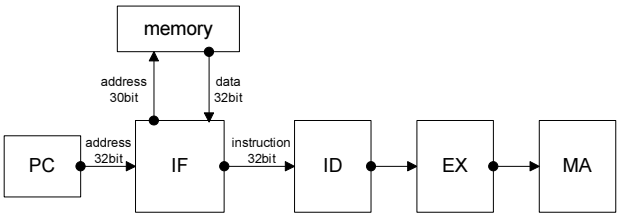

図 2: Block diagram of the original Plasma design.

本章の評価では，以下の 4 つの設計を用いる．

Original 図 2 に示したオリジナルの Plasma.

Specialized Original の命令コードをランダムに入れ替えて特殊化した設計.

RAM-mapped 命令コードの写像を RAM で実装し，パーソナリティの書き換えを可能にした設計．写像部分の論理構成は，図 3 に示されている．

Bit-shuffle 32 bit の命令ワードを任意にビット転置する論理を，先行研究 [15] で提案されたとおり 32 組の 32:1 マルチプレクサと 160-bit レジスタにより実装し，Original の Plasma に付加したもの．ビット転置の概要は，図 4 に示されている．

表 4 に，各設計の論理合成の結果をまとめた．表中，Slice は Spartan3 FPGA 上の実装規模をあらわす単位である．Slice は論理資源 (SliceL) とメモリ資源 (SliceM) からなるが，表には総 Slice 数のうちの SliceM 数も明示した．Spartan3 FPGA には，SliceM の他に BlockRAM というメモリ資源も搭載されているので，その使用量も表 4 に示した．ちなみに今回の評価では，図 3 の RAM1, RAM2, RAM3 は SliceM で実装されており，Plasma の

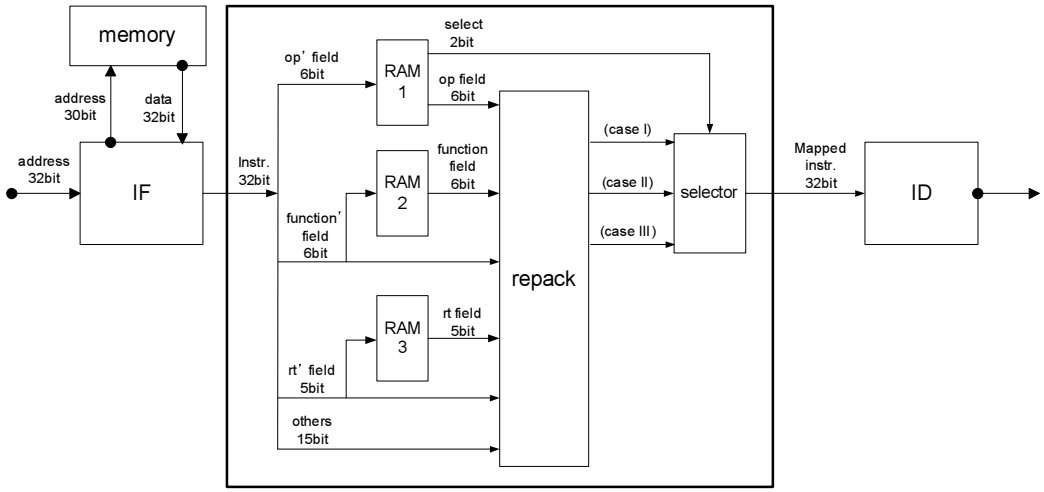

図 3: Partial block diagram of a RAM-mapped Plasma design.

表 4: Evaluation results of various designs.

	Slice (SliceM)	BlockRAM	Freq. (MHz)	Synthesis (s)
Original	1911 (128)	4	35.3	437
Specialized (avg.)	2009 (128)	4	34.1	479
RAM-mapped	1979 (128)	4	31.2	461
Bit-shuffle	2139 (192)	4	33.2	522

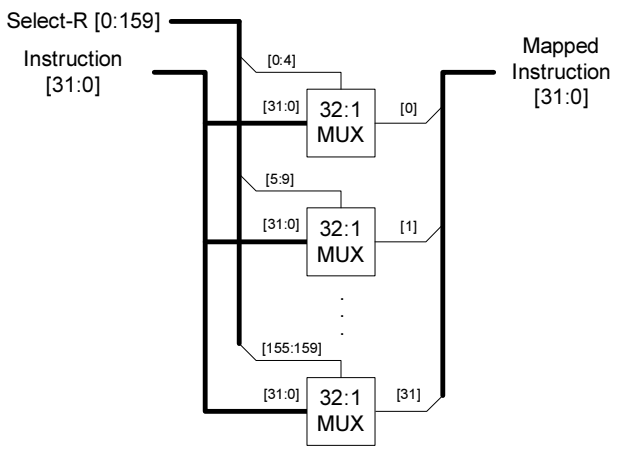

図 4: Block diagram of bit-shuffling logic.

主記憶 (図 2 の memory) は BlockRAM で実装されている．

表 4 中，Specialized の結果はランダムに特殊化された 100 個のプロセッサの平均値を示している．ちなみに Slice 数の最大は 2117, 最小は 1868 なので，特殊化により論理規模に 12%ほどのばらつきが発生した．動作周波数の最大は 37.9 MHz, 最小は 31.1 MHz なので，動作速度にも 20%ほどのばらつきがあった．RAM-mapped は，論理規模が Specialized の平均より僅かに (1.5%) 小さく，動作速度は 8.5%ほど遅い．

表 4 から，Specialized も RAM-mapped も，Original に比べてコスト・性能の両面でほとんど遜色ないことがわかる．Bit-Shuffle の論理規模は，Original の 12%増，RAM-mapped と比べても 8%の増で，実装コストは提案手法より高い．論理合成時間は 7〜9 分で大差ないが，Specialized では個別に論理合成が必要であるのに対し，他の設計は一度合成するだけでよいことに注意すべきである．

6 おわりに

本研究では，命令コードの割り当てを変更することにより，命令セットアーキテクチャを保存しつつプロセッサの多様化を実現する手法について提案した．その結果，先行研究で実現されていなかった (真の意味の) 命令セットランダム化を実現し，その自由度は先行研究の手法より大きいことが示された．さらに，FPGA 上で実際に設計評価した結果，RAM による実装では，特殊化による論理規模の増加は 3.5%にとどまり，性能低下も 11.6%に留まることが示された．

以上の評価結果より，命令セットの冗長性を利用したプロセッサの多様化は，FPGA を用いた組込みシステム適したソフトウェア保護手法として極めて有望であることがわかった．

謝辞

本研究の一部は，21 世紀 COE プログラム“ インテリジェントヒューマンセンシング ”および科学研究費補助金・基盤研究 (C)(2)16500029 の援助により行われた.

参考文献

[1] S. Forrest, A. Somayaji, and D. Ackley, “Building diverse computer systems,” HotOS ’97: Proceedings of the 6th Workshop on Hot Topics in Operating Systems (HotOS-VI), Washington, DC, USA, p.67, IEEE Computer Society, 1997.

[2] D.L.C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell, and M. Horowitz, “Architectural support for copy and tamper resistant software,” ASPLOS-IX: Proceedings of the ninth international conference on Architectural support for programming languages and operating systems, New York, NY, USA, pp.168–177, ACM Press, 2000.

[3] G.E. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas, “AEGIS: architecture for tamper-evident and tamper-resistant processing,” ICS ’03: Proceedings of the 17th annual international conference on Supercomputing, New York, NY, USA, pp.160–171, ACM Press, 2003.

[4] R.B. Lee, P.C.S. Kwan, J.P. McGregor, J. Dwoskin, and Z. Wang, “Architecture for protecting critical secrets in microprocessors,” ISCA ’05: Proceedings of the 32nd Annual International Symposium on Computer Architecture, Washington, DC, USA, pp.2–13, IEEE Computer Society, 2005.

[5] D. Kirovski, M. Drinić, and M. Potkonjak, “Enabling trusted software integrity,” ASPLOS-X: Proceedings of the 10th international conference on Architectural support for programming languages and operating systems, New York, NY, USA, pp.108–120, ACM Press, 2002.

[6] M. Milenković, A. Milenković, and E. Jovanov, “A framework for trusted instruction execution via basic block signature verification,” ACM-SE 42: Proceedings of the 42nd annual Southeast regional conference, New York, NY, USA, pp.191–196, ACM Press, 2004.

[7] J. Zambreno, A. Choudhary, R. Simha, B. Narahari, and N. Memon, “SAFE-OPS: An approach to embedded software security,” Trans. on Embedded Computing Sys., vol.4, no.1, pp.189–210, 2005.

[8] G.E. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas, “Efficient memory integrity verification and encryption for secure processors,” MICRO 36: Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture, Washington, DC, USA, p.339, IEEE Computer Society, 2003.

[9] J. Yang, Y. Zhang, and L. Gao, “Fast secure processor for inhibiting software piracy and tampering,” MICRO 36: Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture, Washington, DC, USA, p.351, IEEE Computer Society, 2003.

[10] B. Rogers, Y. Solihin, and M. Prvulovic, “Memory predecryption: hiding the latency overhead of memory encryption,” SIGARCH Comput. Archit. News, vol.33, no.1, pp.27–33, 2005.

[11] M. Drinić and D. Kirovski, “A hardware-software platform for intrusion prevention,” MICRO 37: Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture, Washington, DC, USA, pp.233–242, IEEE Computer Society, 2004.

[12] H. Shacham, M. Page, B. Pfaff, E.J. Goh, N. Modadugu, and D. Boneh, “On the effectiveness of address-space randomization,” CCS ’04: Proceedings of the 11th ACM conference on Computer and communications security, New York, NY, USA, pp.298–307, ACM Press, 2004.

[13] E.G. Barrantes, D.H. Ackley, T.S. Palmer, D. Stefanovic, and D.D. Zovi, “Randomized instruction set emulation to disrupt binary code injection attacks,” CCS ’03: Proceedings of the 10th ACM conference on Computer and communications security, New York, NY, USA, pp.281–289, ACM Press, 2003.

[14] E.G. Barrantes, D.H. Ackley, S. Forrest, and D. Stefanović, “Randomized instruction set emulation,” ACM Trans. Inf. Syst. Secur., vol.8, no.1, pp.3–40, 2005.

[15] G.S. Kc, A.D. Keromytis, and V. Prevelakis, “Countering code-injection attacks with instruction-set randomization,” CCS ’03: Proceedings of the 10th ACM conference on Computer and communications security, New York, NY, USA, pp.272–280, ACM Press, 2003.

[16] J.C. Dehnert, B.K. Grant, J.P. Banning, R. Johnson, T. Kistler, A. Klaiber, and J. Mattson, “The Transmeta Code Morphing^{TM}Software: using speculation, recovery, and adaptive retranslation to address real-life challenges,” CGO ’03: Proceedings of the international symposium on Code generation and optimization, Washington, DC, USA, pp.15–24, IEEE Computer Society, 2003.

[17] MIPS Technologies, Inc., MIPS32^{TM}Architecture for Programmers (Volume I: Introduction to the MIPS32^{TM}Architecture), June 2003. Rev.2.00.

[18] Renesas Technology, SH-3/SH-3E/SH3-DSP Software Manual, May 15 2006. REJ09B0317-0400.

[19] Intel Corp., Intel 8080A/8080A-1/8080A-2 8-bit N-channel Microprocessor, Nov. 1986.

[20] T. Lindholm and F. Yellin, Java Virtual Machine Specification, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[21] Xilinx, Inc., Spartan-3 FPGA Family: Complete Data Sheet, 26 April 2006. DS099 (v2.1).

[22] S. Rhoads, “Plasma - most MIPS I (TM) opcodes: Overview,” November 2006. http://www.opencores.org/projects.cgi/web/mips/.