論文

命令セットアーキテクチャの多様化に関する研究

Diversity of Instruction Set Architecture

043714　澤田　豊志
豊橋技術科学大学
知識情報工学専攻

概要

Abstract

1. はじめに

近年，コンピュータやインターネットは一般に普及している．しかし，一方で悪意ある第三者 (攻撃者) からの攻撃は脅威を増している．こうした攻撃者のウィルスやワームによる攻撃や，リバースエンジニアリングによるソフトウェアの盗用，改竄は社会問題となっている．例えば，ウィルスやワームはバッファオーバーフローを利用してインジェクション攻撃を行なうが，これらの攻撃に対する防御は近年ますます重要になっている．

バッファオーバーフローとは，システムが想定しているサイズ以上の多量のデータを送りつけることにより，入力バッファとして確保しているメモリ領域を溢れさせて，バッファの後にあるメモリ領域を上書きし制御を奪う攻撃方法である．バッファオーバーフローにより，関数などの戻りアドレスを攻撃者が用意した不正な命令コードへのアドレスに書き換えることが可能になる．不正な命令コードをメモリに注入 (インジェクション) し，バッファオーバーフローを利用して実行する攻撃方法をインジェクション攻撃と呼ぶ．インジェクション攻撃は，特定のプロセッサアーキテクチャ (例：x86 など) を想定しているため，他の命令セットでは動作しない．

個々のプロセッサごとに命令セットが異なっていれば攻撃者は攻撃を行なうことが難しくなる．つまり，プロセッサごとに命令セットが違うなどの個性を持たせて，多様化することができれば攻撃を受けにくくなる．また，リバースエンジニアリングによるソフトウェアの盗用や改竄も，多様化されている命令セットの知識がなければ意図した成果は得られない．

プロセッサに多様性を持たせる方法として命令セットアーキテクチャ (ISA) の多様化が考えられる．ISA には，命令セットや使用するレジスタ等が定義されている．ISA の多様化として ISA の構成要素である命令セットを多様化する手法が考えられる．別な方法として，プログラムカウンタの増加値を多様化することを考えることができる．プログラムカウンタは次の命令へのアドレスを指し示すため，固定長命令の命令セットならば命令長ごとに単調増加している．プログラムカウンタの増加値を多様化することで，それに対応してメモリの命令列の記述順序をシャッフルすることができる．攻撃者は，命令列だけでなくプログラムカウンタの更新値を調べなければ攻撃を行なうことができない．

命令セットの多様化は現行の汎用コンピュータにも適用可能であるが，プロ

セッサの実装自体を個体毎に行なう必要がある．そのため，要求によりシステム構成を変更することが多い組込みシステムへの適用が向いている．FPGA (Field Programmable Gate Array) は再構成が可能な論理 LSI であり，単品生産が可能なため本手法の実装には最適であると考えられる．

本研究では，まず命令セットを多様化する新手法を提案し，命令セットにどれだけの自由度があるか，手法に適した命令セットの条件，手法に対する攻撃について検討する (3 章)．また，別な案として，プログラムカウンタの増加値を多様化する手法についても考察する (4 章)．最後に FPGA 上に命令セットの多様化手法の実装を行ない，既存手法との比較を行なう (5 章)．

2. 関 連 研 究

ウィルスやワームによる攻撃やリバースエンジニアリングによるソフトウェアの盗用や改竄に対する研究は，ソフトウェア，ハードウェアの両面で多く存在する．

ソフトウェア面での対策に難読化という手法がある．難読化は，ソフトウェアに含まれる保護したい対象を理解のしにくい難解なものに書き換えることで，攻撃者の解析や改竄を困難にさせる手法である[1)]．門田らはプログラムを難読化する方法として，ループを使用しているプログラムをより難解なループを持つプログラムにする手法を提案した[2)]．また，門田らは任意の命令を異なる命令に偽装する命令とその命令を復元する命令，再び偽装する命令をオブジェクトファイルに追加することで，実行時においてのみ元の命令に復元する手法を提案している[3)]．攻撃者は偽装されている命令を特定する必要があり，それには全体を見なくてはならなくなるために部分的な盗用が難しくなる．

また，ハードウェアにおいてはメモリの暗号化や MAC (Message Authentication Code) を使用する方法がある．メモリの暗号化では XOM (eXecute-Only Memory) と呼ばれる，実行される命令部分だけが格納された暗号化されたメモリを用意する手法がある[4)]．XOM では，命令部分が暗号化されているため，実行時には復号化して実行する．一方，MAC と呼ばれる認証コードを各命令コードに付属させることで実行時に変造がないか認証する手法がある[5)]．しかし，これら手法は暗号化されたメモリの復号化処理や MAC の認証処理が生じるため，メモリアクセスの遅延が大きくなるという性能オーバヘッドを伴っている．

命令セットの多様化に関連する研究としては，まず，Barrantes らの RISE (Randomized Instruction Set Emulation) がある[6)]．彼らは，命令列に対して擬似乱数列を XOR したものをメモリに格納し実行時に元に戻すことを提案し RISE と名づけた．しかし，この手法で行なわれていることは厳密な命令セットの多様化ではなくメモリのランダム化といえる．また彼らは，メモリの復元をソフトウェアでエミュレーションすることにより行なうと提案しており，ハードウェアによる実装は考えていない．

一方 Kc らは，命令に対して鍵を XOR する手法と命令ワードのビットを任意に転置する手法を提案した[7)]．彼らは Barrantes らと違い 2 つの手法をハードウェアで実装の検討をしている．ここで行なわれている鍵を XOR する方法は，

Barrantes らと同じくメモリのランダム化と言える．また，命令ワードのビットを転置する手法は，命令セットの多様化と言えるが固定長の命令セットでしか適用できない．

門田らは，FSM を導入して命令の符号化の解釈を FSM の状態によって切り替える手法を提案した[8]．FSM を導入することにより同じ命令を複数の命令に偽装することが可能になるが，その半面分岐命令による状態の差異が起きてしまう．彼らは，ダミー命令を挿入し正しい状態に遷移させることで問題を防いでいるが，逆にダミー命令が解読の糸口になってしまう問題を挙げている．命令の解釈を FSM で切り替えるという方法は命令セットの多様化というよりは，命令セットの難読化と考えることができる．また，ハードウェアへの実装は行なっていない．

Kc らの鍵を XOR する手法は，Barrantes らの手法と本質的に大きな相違はないため，本研究では既存手法との比較に Kc らの手法を用いることにする．

3. 命令セットの自由度

この章では，命令セットを多様化する手法について説明し，各命令セットにどれだけの自由度が得られるかを評価する．また，攻撃の可能性について述べる．

3.1 命令セットアーキテクチャ

命令セットアーキテクチャ (ISA) には命令セット，使用されるレジスタ等が定義されている．命令セットには命令が定義されており，各命令では処理内容やオペランドが定義されている．また，命令のメモリ表現は命令形式によって定義されており，例えば，MIPS[9] には R, J, I の 3 つの命令形式 (図 1) が定義されている．命令形式では，命令を区別する opcode フィールドを始め function フィールドなど各種フィールドが定義されている．命令セットによって命令形式の形や数，定義されるフィールドは異なっている．

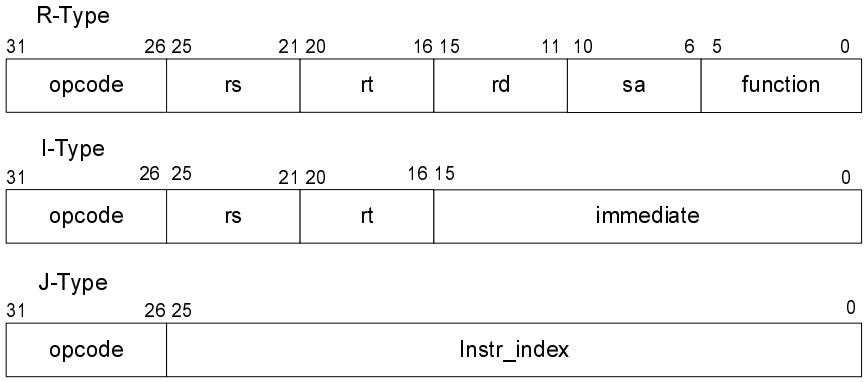

図 1 例：MIPS 命令セットの命令形式

3.2 命令セットの多様化手法について

命令を区別している opcode フィールドの値を入れ替えることで，命令セットを多様化することが可能である．例えば図 2 の場合，命令 A と命令 B の opcode フィールドの値を入れ替えた ISA を定義する．opcode A で定義される命令 A と opcode B で定義される命令 A は，異なったメモリ表現であるが命令の処理内容は変わらない．命令 B も同様で，これにより命令 A と B のメモリ表現が異なった命令セットを作り出すことができる．生成された ISA は，opcode を入れ替える前の ISA と同等の機能を持つ．ここで，このようなメモリ表現の異なった命令セットを命令セットの**パーソナリティ**と呼ぶことにする．また，命令セッ

トからどれだけのパーソナリティを生み出せるかを命令セットの自由度と呼ぶ．本手法は，このような命令コードの符号化の自由度を利用して命令セットを多様化する．

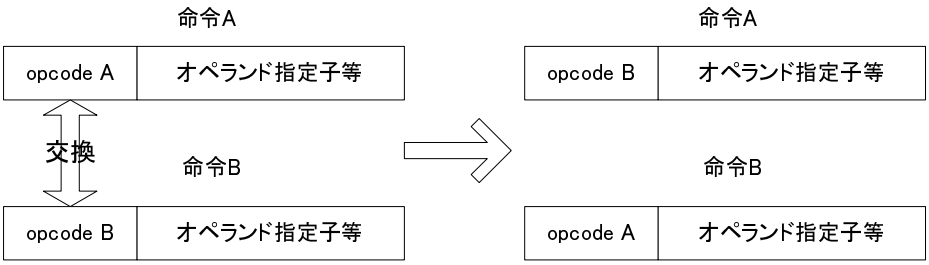

図 2　パーソナリティの生成：opcode フィールドの書き替え

3.2.1　命令セットの多様化手法の利点

本手法の利点として，まず攻撃者が不正な命令列を実行しにくい点を挙げる．これは，同じ命令セットのように見えても命令のメモリ表現が異なっているために，攻撃者が想定した命令が実際には別な命令になりその処理が変わってしまうためである．攻撃者が攻撃を実行するには命令の変換規則を解読する必要がある．

同じようにリバースエンジニアリングによるアルゴリズムの盗用の面では，メモリ表現が異なっているために，そのまま盗用しても同じ動作を得ることはできない．そのため，攻撃者が命令列を解読する必要がある．一方で命令の改竄は可能である．しかし，命令のメモリ表現が異なっているために，改竄による攻撃者が想定した効果は得られにくい．

また，命令セットのマニュアルを公開してもセキュリティが低下しない点も利点である．マニュアルを公開することで攻撃者は命令の仕様を知ることができるが，実際のメモリ表現がわからなければ攻撃者は攻撃を実行しにくくなる．

命令のメモリ表現を変更するだけで命令の処理内容は変更されないため，命令デコーダの論理が若干変更になるだけで，メモリの暗号化や MAC による認証手法のようなメモリアクセス遅延の増加は殆ど無いと考えられる．この点に関しては，後ほど 5 章で評価結果を示す．

本手法は，関連研究で挙げた Barrantes らや Kc らの手法と直交しており併用することが可能である．

3.3 命令セットごとの評価

命令セットのパーソナリティの生成方法には，まず 3.2 章で説明した opcode フィールドを入れ替える方法 (図 2) が考えられる．この方法は，同じ長さの opcode フィールドであれば入れ替えることが可能となる．ここで得られる命令セットの自由度は，opcode フィールドを入れ替えることのできる命令数の階乗である．

2 つ目のパーソナリティの生成方法としては，命令形式にある特別な拡張フィールドを使用する方法がある．例えば，MIPS の R 形式では opcode フィールドだけではなく function フィールドと呼ばれる拡張フィールドによって命令が区別されている．そこで，図 3 のように拡張フィールドの値の入れ替えによってパーソナリティを生成することが可能である．この方法は，同じ命令形式であり同じ命令長のときのみ可能である．例えば，MIPS の function フィールドは R 形式にしかないため，入れ替えは R 形式の命令群にのみ可能である．ここで得られる命令セットの自由度は，拡張フィールドを入れ替えることが可能な命令数の階乗である．この生成方法は命令セットによっては複数存在する．

両方の方法で得られた自由度を掛け合わせたものを命令セット全体の自由度として評価を行なう．

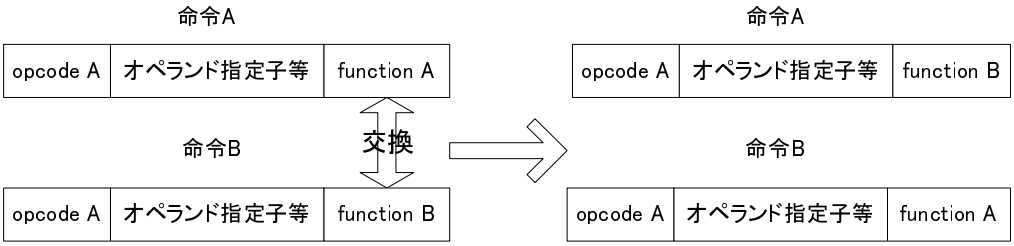

図 3　パーソナリティの生成：拡張フィールドの書き替え

また，上記 2 種類の方法以外にもオペランドやオペランド指定子といったフィールドを入れ替えることによる自由度もあるが本研究では扱わないものとする．

これらを踏まえた上で以下の評価条件を決めた．

- 命令形式の構造を変えない．
- 拡張フィールドを使用した生成方法では，その命令形式内でのみ考える．
- オペランドやオペランド指定子の入れ替えは考えない．
- 未定義命令を入れ替えの自由度に考えない．

本手法を評価する上で重要な条件として命令形式の構造を変えないことがある．命令形式の構造を変えることによってもパーソナリティを得ることは可能である．例えば，拡張フィールドを入れ替える方法では命令形式を超えて拡張フィールドを入れ替えることを考えることができる．しかし，その場合にはパーソナリティごとに命令形式を新たに定義しなおす必要がある．多様化したISAの自動生成まで考えれば，命令形式の定義までを多様化するのは現実的ではない．この点はオペランドやオペランド指定子の制限も同様である．

また，未定義命令は動作が不定であることからユーザが使用することは無いと考えられる．ユーザが使用しない命令を入れ替えても自由度にはならない．

これらの評価条件のもと各命令セットからどれだけの自由度が得られるかを調査した．また，各命令セットの命令種別ごとにどれだけの自由度が得られるかも調査を行なった．

3.3.1 MIPS

MIPS32-ISA[9)10)]は32ビットCPUで使用される命令セットである．MIPS32-ISAには命令が244命令ある．今回は，MIPS32 Release1を使用した．Release1で定義されている命令数は170命令である．図4にMIPSの命令形式を示す．命令は全てが32ビットの固定長であり，その命令形式は3つある．また，opcodeフィールドは6ビットで構成されており，opcodeにはオペランド指定子が含まれていない．

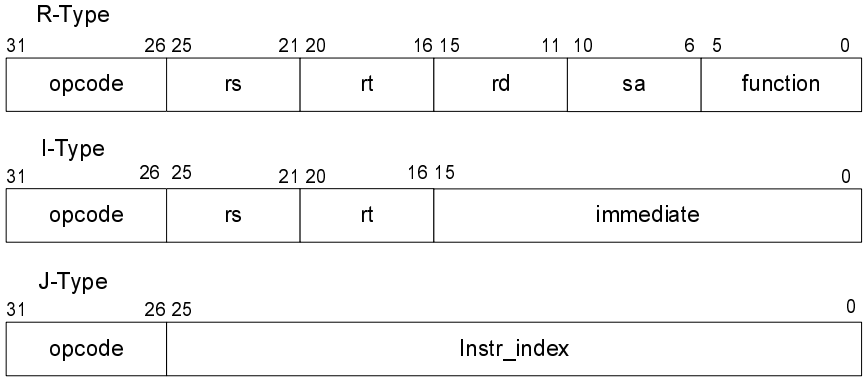

図4　MIPS命令セットの命令形式

opcodeフィールドの入れ替えを行なえる命令は48命令あり，opcodeフィールドの入れ替えによって生まれる自由度は$48! = 1.24 \times 10^{61}$になる．また，拡

張フィールドの入れ替えではそれぞれopcodeがCOP0の場合で$3! \times 7!$，opcodeがCOP1の場合で$8! \times 4! \times 17! \times 2! \times 17! \times 2! \times 2!$，opcodeがCOP2の場合で$6! \times 4!$，opcodeがSPECIALの場合で$38! \times 2!$，opcodeがSPECIAL2の場合で$8!$，opcodeがREGIMMの場合で$14!$の自由度が得られる．拡張フィールドの入れ替えによる各自由度を掛け合わせると1.88×10^{105}になる．MIPS命令セット全体の自由度は，両方の自由度を掛け合わせて2.34×10^{166}になる．

3.3.2 SH-3

命令セットSH-3[11]には命令が全部で188命令定義されている．その命令長はMIPSと同じく固定長であるがMIPSと違いSH-3は16ビットしかない．命令形式は図5のようになっており，命令形式の数は12個ある．opcodeフィールドには4ビット割り当てられており，opcodeにはオペランド指定子は含まれていない．

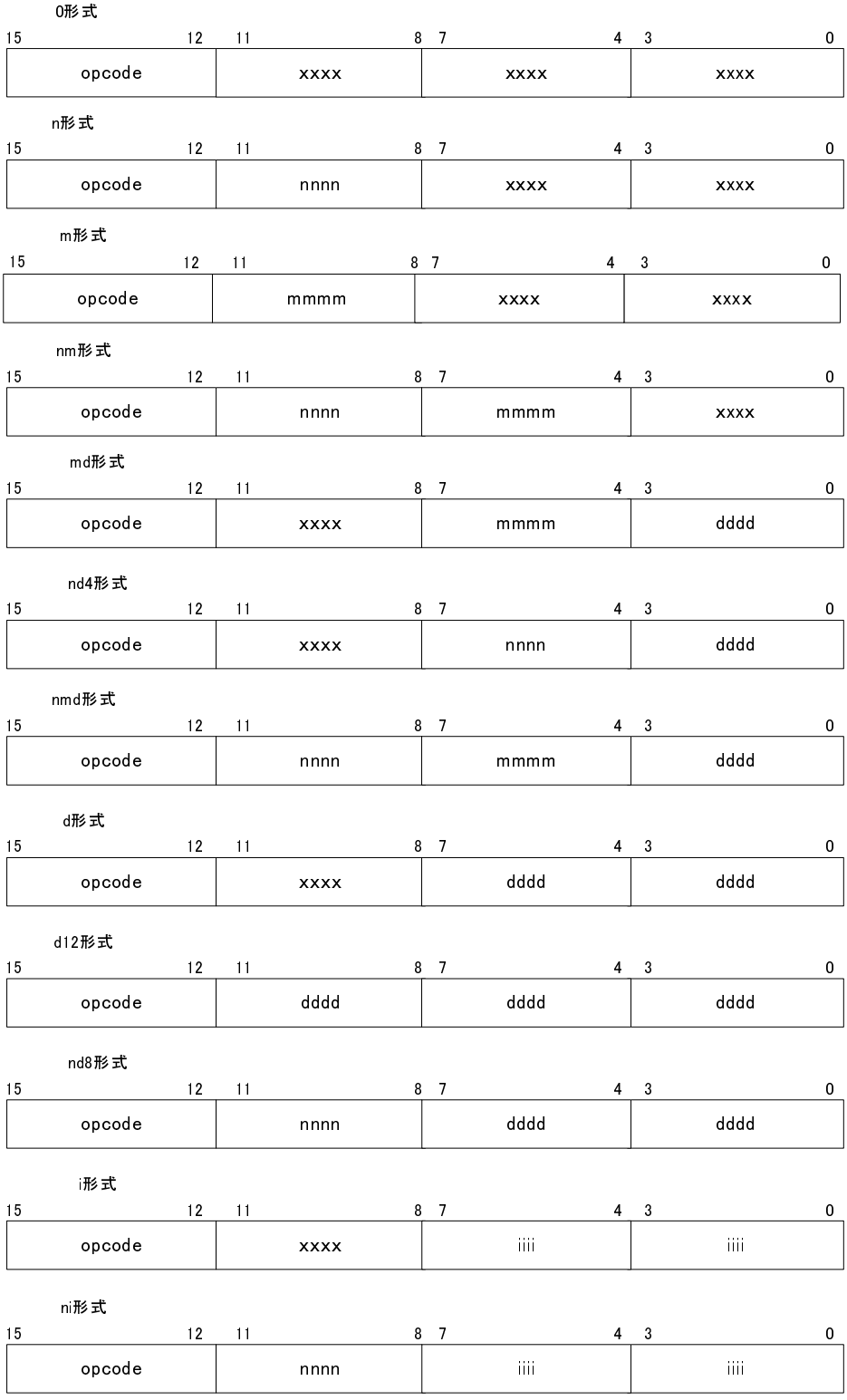

図 5　SH-3 命令セットの命令形式

opcode フィールドを入れ替えることができる要素が 15 個のため，opcode フィールドの入れ替えで生まれる自由度は $15! = 1.31 \times 10^{12}$ になる．また，拡張フィールドの入れ替えはそれぞれ opcode が 0000 の場合で $11! \times 3! \times 8!$，opcode が 0010 の場合で 15!，opcode が 0011 の場合で 14!，opcode が 0100 の場合で $18! \times 2! \times 2! \times 2! \times 3!$，opcode が 0110 の場合で 16!，opcode が 1000 の場合で $2! \times 2! \times 4!$，opcode が 1100 の場合で $7! \times 9!$ の自由度が得られる．拡張フィールドの入れ替えで生まれる自由度は全部で 1.24×10^{78} になる．SH-3 命令セット全体の自由度は，両方の自由度を掛け合わせて 1.63×10^{90} になる．

3.3.3 JavaVM

JavaVM は全部で 201 命令定義されている．その命令長は MIPS や SH-3 と違い可変長であり，命令形式は図 6 のようになっていて 1 つだけである．opcode フィールドは 8 ビットあり，オペランド指定子は含まれていない．命令形式は 1 つであり全ての命令が opcode フィールドだけで区別されているため，パーソナリティは opcode フィールドの入れ替えのみでしか生成できない．入れ替え可能な opcode は 201 個あるため，JavaVM の自由度は $201! = 1.59 \times 10^{377}$ になる．

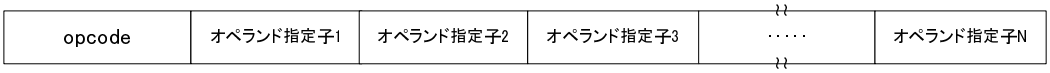

図 6 JavaVM 命令セットの命令形式

3.3.4 8080

命令セット 8080[12] には全部で 112 命令定義されている．その命令長は JavaVM と同じく可変長命令である．命令形式は図 7 のようになっており，命令長の違う 3 つの命令形式がある．opcode フィールドは 8 ビットあり，これまでの命令セットと違い opcode 部分にオペランド指定子を含むものがある．

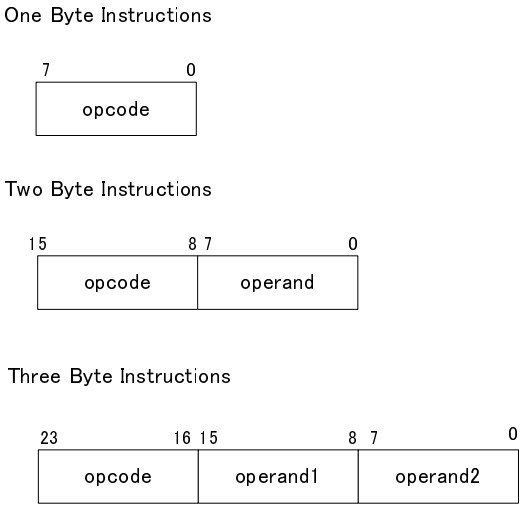

図 7 8080 命令セットの命令形式

opcode フィールドにオペランド指定子が無い命令で入れ替えが可能なものが 86 要素ある．opcode にオペランド指定子がある命令のうち 4 要素と 8 要素がそれぞれ入れ替え可能である．opcode フィールド以外はオペランドなため，拡張フィールドの入れ替えは行なえない．命令セット全体の自由度は $86! \times 4! \times 8! = 2.34 \times 10^{136}$ になる．

3.3.5 Z80

命令セット Z80[13] は命令セット 8080 を元に作られた命令セットである．総命令数は 202 命令と 8080 よりも数が増えている．命令長は 8080 と同じく可変長である．しかし，8080 では命令長が 3 種類だったのに対して Z80 では 32 ビットの長さの命令が存在する．また opcode の長さが 8 ビットと 16 ビットの 2 パターンあり，opcode にはオペランド指定子が含まれているものもある．

Z80 のマニュアルに命令形式が確認できなかった．代わりに図 8 で示されるアドレシングモードと呼ばれる形があり，さらにモードを組み合わせたものが存在することがわかった．今回は，アドレシングモードとそれを組み合わせた

形のものを命令形式として考え評価を行なった．opcode フィールドが可変であるため，opcode フィールドの入れ替えは同じアドレシングモードの命令のみで考えた．

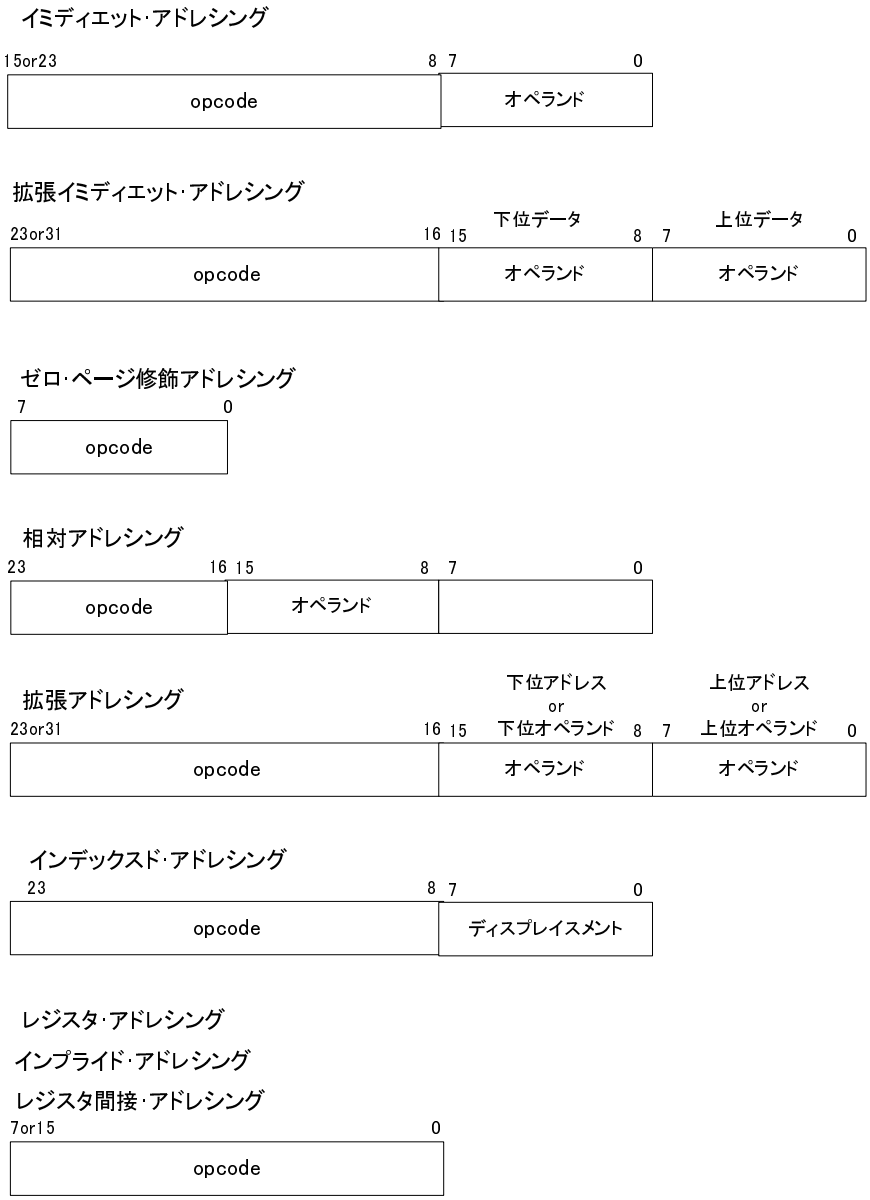

図 8　Z80 命令セットのアドレシングモード

各アドレシングモードの自由度は，イミディエットアドレシングで 8!，拡張イミディエットアドレシングで $2! \times 2!$，相対アドレシングで 6!，インデックスドアドレシングで $10! \times 7!$，レジスタアドレシングで $8! \times 2 \times 3! \times 3! \times 3! \times 7!$，

インプライドアドレシングで $15! \times 4!$，レジスタ間接アドレシングで $12! \times 16!$，ビットアドレシングで $3! \times 3! \times 3!$，レジスタ＋インプライドで $4!$，レジスタ＋レジスタ間接で $4! \times 2 \times 2 \times 2$，レジスタ＋インデックスドで 2，レジスタ＋拡張アドレシングで $4! \times 2 \times 2$，レジスタ＋イミディエットで 2 になる．命令セット全体の自由度は，各アドレシングモードの自由度を掛けたものになり 2.24×10^{73} となる．

今回の評価では前述のとおり，命令形式を使って評価が行なえていないため評価条件に適しない．そのため，Z80 の評価は試験的な評価としておく．

3.4 評価結果と考察

各命令セット (但し Z80 を除く) の評価結果を表 1 にまとめた．評価結果と各命令セットの特性から，命令セットの多様化にとって重要な命令セットの要素を考察していく．

表 1 命令セット全体の自由度

	総命令数	自由度	情報量 [bit]
MIPS	170	2.34e+166	553
SH-3	188	1.63e+90	300
Java VM	201	1.59e+377	1253
8080	111	2.34e+136	453

3.4.1 命令長が固定長の命令 (MIPS，SH-3)

本節では命令長が固定長である命令セット MIPS と SH-3 について比較する．

まず，命令数と自由度の差が見て取れる．命令数は，SH-3 の方が MIPS よりも多いのだが自由度は MIPS の方が多くなった．原因として opcode の長さが MIPS で 6 ビットであるのに対して SH-3 が 4 ビットであることが挙げられる．それぞれ opcode フィールドだけで 64 命令，16 命令の命令を定義することができる．しかし，実際には未実装命令があるため少なくなっている．仮にそれだけの命令が定義されていた場合，MIPS は SH-3 の $64!/16! = 6 \times 10^{75}$ 倍の自由度があることになる．

もう一つの原因として命令形式がある．MIPS は，命令形式が 3 つと簡素な作りになっているのに対して SH-3 では命令形式が多く 12 個も存在する．拡張フィールドの入れ替えは，命令形式ごとに入れ替えを考えるため，少ない命令

形式により多くの命令が集まったMIPSの方が多くの自由度を得られる結果になった．例えば，ある10個の命令が拡張フィールドで入れ替え可能であった場合を考える．命令形式が1つで10個の命令を入れ替えられる場合と命令形式が2つで5個ずつの命令が入れ替えられる場合では，10!と$5! \times 5!$となりその自由度の差 $(3.63 \times 10^6 - 1.44 \times 10^4)$ は大きくなる．

3.4.2 命令長が可変長の命令 (JavaVM，8080，Z80)

本節では命令長が可変長な命令セットであるJavaVM，8080，Z80について比較する．

まず，JavaVMが飛びぬけて自由度が高いことがわかる．同じような命令数のZ80よりも多くの自由度を得ている．これは，JavaVMの命令形式が簡素であることが挙げられる．JavaVMには命令形式が1つしかないためで多くの命令が自由度に影響した．

また，JavaVMと8080ではどちらもopcodeフィールドの入れ替えの自由度だけであった．しかし，opcodeフィールドにオペランド指定子が含まれる8080では全ての命令で入れ替えを行なうことができず自由度は総命令数の階乗にはならなかった．つまり，opcodeフィールドに対してオペランド指定子を含まない命令セットであれば自由度は生まれやすいことがわかる．

3.4.3 命令長が固定長と可変長の命令 (MIPS，SH-3，JavaVM，8080，Z80)

本節では命令長が固定長と可変長の命令セットを比べる．

MIPSと8080を見てみると命令長が固定長でも可変長でも自由度に違いは見られない．但し，固定長命令では定義できる総命令数に限りがあるため，命令長が短ければ自由度が低くなりやすい．また，可変長命令では拡張フィールドによる自由度が増える可能性がある．8080ではopcodeフィールドの入れ替えのみのため，拡張フィールドが定義されれば自由度は増える．しかし，実際に8080を拡張させたZ80では命令長の種類が増え，命令数も増えたにも関わらず自由度が減ってしまっている．これは，命令形式が簡素であることが大きく関係している．JavaVMとSH-3を見ても分かるとおり命令数が殆ど変わらなくても自由度には大きな差が出ている．それぞれ命令形式が1つと12個の違いがある．8080には命令形式が3つであるのに対してZ80には自由度が得られたものだけでも13個ある．

3.4.4　8080 とその拡張命令セットである Z80

本節では 8080 とその拡張命令セットである Z80 を比較する．

Z80 に拡張されて命令数は倍近く違うが，自由度は拡張される前の 8080 の方が多い結果になった．原因として，opcode フィールドが固定長である 8080 は入れ替えの制限を受けにくかったことがある．Z80 には opcode フィールドの長さが 2 種類あるため，入れ替えを行なえる命令数が減ってしまった．

2 つ目の原因としては，今回 Z80 では簡素な命令形式が見つからなかったため，アドレシングモードを使用していることが挙げられる．アドレシングモードとは命令形式をさらに細かく分類したものであり，そのために Z80 では命令が各アドレシングモードに分散する結果となり大きな自由度が生まれなかった．

3.4.5　自由度が多くなる条件

各命令セットの比較から本手法において自由度の多くなる条件がわかった．

- opcode フィールドが十分な長さを持っている．
- opcode にオペランド指定子を含んでいない．
- 命令形式が簡潔である．
- 可変長命令のほうが有利である．

まず，opcode フィールドが十分な長さを持っていること．opcode フィールドの長さが長ければそれによって区別される命令数が増え，入れ替えが可能となる命令数が増える．つまりは，opcode フィールドの入れ替えによる自由度が増加する．

次に opcode に対してオペランド指定子を含まないことが望ましい．今回の評価条件ではオペランド指定子の入れ替えを考えないことにしているため，オペランド指定子を含む opcode は含まないものと入れ替えなかった．

3 つ目に命令形式が少ない，命令形式が簡潔であることが求められる．JavaVM のように 1 つであることが最も望ましいが，命令形式が少なければ拡張フィールドによる自由度は増える．

4 つ目に命令長は可変長のほうが自由度は多くなる．可変長の方が割り当てることのできる命令数は固定長のものよりも多くなる．命令数が多ければ自由度は生み出しやすい．但し，必ずしも命令数が増えることが自由度の増加に繋がるとは言えない．Z80 では 8080 よりも命令数が増えたが，opcode フィールドの長さが可変になり，逆に自由度が減る結果になった．

以上の条件を満たす命令セットが本手法における最高の命令セットである．

3.4.6 命令種別ごとの自由度

各命令セットの命令種別ごとに自由度を評価した．各命令セットの代表的な命令種別を集めて表 2 にした．ここで表示されている命令種別は厳密なものではない．例えば，SH-3 の PREF 命令の区分はデータ転送命令とシステム制御命令の両方だがデータ転送命令に数えている．

命令種別ごとなので，命令セット全体の自由度よりはかなり低くなった．その中でも命令数の多い算術演算命令群とデータ転送命令群は，自由度がある程度得られることがわかった．

表 2 命令種別ごとの自由度（一部）

	算術演算	データ転送	分岐	システム制御	特権
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	–

3.4.7 命令セットの多様化によって得られる自由度はどうなのか

本手法は組込みシステムへの適用を考えている．本手法によって得られる自由度は，組込みシステムの製品として生産するのに十分な量の自由度がある．例えば，自由度の少なくなった SH-3 において 10 億個の製品それぞれに 100 億のパーソナリティを割り振ることができる．

Kc らの手法[7] と本手法はどちらも本質的には換字暗号であり，自由度が大きければ大きいほど総当り攻撃に耐性を持つ．Kc らのビット転置の手法の自由度は 2.63×10^{35} なのに対して本手法で一番少ない自由度だった Z80 でも 2.24×10^{73} の自由度がありその差は大きいことがわかる．また本手法は Kc らの手法と直交しているため併用が可能であり，両者を併用することでより耐性を強くすることもできる．

また，命令種別ごとの自由度の評価より命令種別ごとにパーソナリティを変更する応用も想定できる．例えば，特権命令のパーソナリティを変えれば特権命令に対してのみ制限を掛け，特権命令以外は普通に使える ISA も作ることが可能になる．

3.5 攻撃の可能性

本手法に対しての攻撃としてはまず総当り攻撃が考えられる．総当り攻撃とは，暗号解読方法の一つで可能な組み合わせを順番に当てはめて解読を試みる方法である．総当り攻撃は，解読に対してヒントがない場合に用いられる力任せの攻撃であり，本手法のように膨大な組み合わせが考えられる場合には現実的ではない．しかし，攻撃を続けられればいつかは解読されてしまう．総当り攻撃は，定期的にパーソナリティを変更することで防ぐことができる．

次に頻度分析攻撃が挙げられる．頻度分析攻撃は，文章中における文字の頻度の傾向を分析して解読を行なうものである．本手法の場合には，命令列中の命令頻度を分析することが考えられる．例えば，算術演算命令はどのような命令列でも必ずといっていいほど使用される．MIPS の R 形式には多くの算術演算命令が指定されており，この R 形式を示す opcode 自体は簡単に特定することができる．また，命令コードに特徴があるものを狙うことが考えられる．例えば，MIPS の R 形式にある sa フィールドの値が 0 である演算命令 (ADD 命令, SUB 命令等) の特徴がある．これらを手がかりにすることによる頻度分析攻撃は可能である．そのため，命令が解読されれば攻撃者によるインジェクション攻撃やアルゴリズムの盗用，改竄の危険がある．

4. プログラムカウンタの増加値の多様化

本章では，プログラムカウンタを利用したISAの多様化の手法を考察する．

4.1 プログラムカウンタについて

プログラムカウンタ (PC) は，次に実行する命令へのメモリアドレスを指し示す．分岐命令ではない場合，メモリの0番地から順に命令を指していく．PCは基本的に単調増加するが，分岐命令によって分岐が起きたときには分岐先の命令のアドレスを指す．PCの増分は命令セットで決まる．固定長の命令ならば命令長がPCの増加値である．例えば，MIPS命令セットならば固定長32ビットであるため，PCは4バイトずつ増えていく．

4.2 本手法の説明

本手法は，通常は単調増加していくPCの増加値をパーソナリティ毎に異なった増加値に変える手法である．

図9の命令列 (左) をシャッフルした命令列 (右) を生成する．通常のPCならば実行順序が変わってしまい，2つの命令列で動作は異なる．例えば，通常ならば最初のADD命令が実行された後，次のADD命令を指し示すためにPCは4バイト分増加される．シャッフルされた命令列ではPCが4バイト増加されると次の命令としてAND命令が実行される．シャッフルされた命令列でシャッフル前の順序のとおりにADD命令を実行させるには，PCを12バイト増加させる必要がある．

ここで，シャッフルされた命令列をシャッフルされる前の命令列と同じ順序で実行できるようにするためのPCの増加値を表にした更新値表を作成する．例えば，最初のADD命令から次のADD命令へはPCの増加値は+12バイト，ADD命令の次のAND命令を実行するための増加値は-8バイトとなる．本手法では，PCを単調増加ではなく更新値表の値を使用することでアドレスの更新を行なっていく．

このように，PC更新値表を用いることにより，PCの増分を任意に定めることができる．

4.2.1 実 現 方 法

本手法の概要図を図 9 に示す．今回は，固定長の MIPS 命令セットでの実現を考えた．

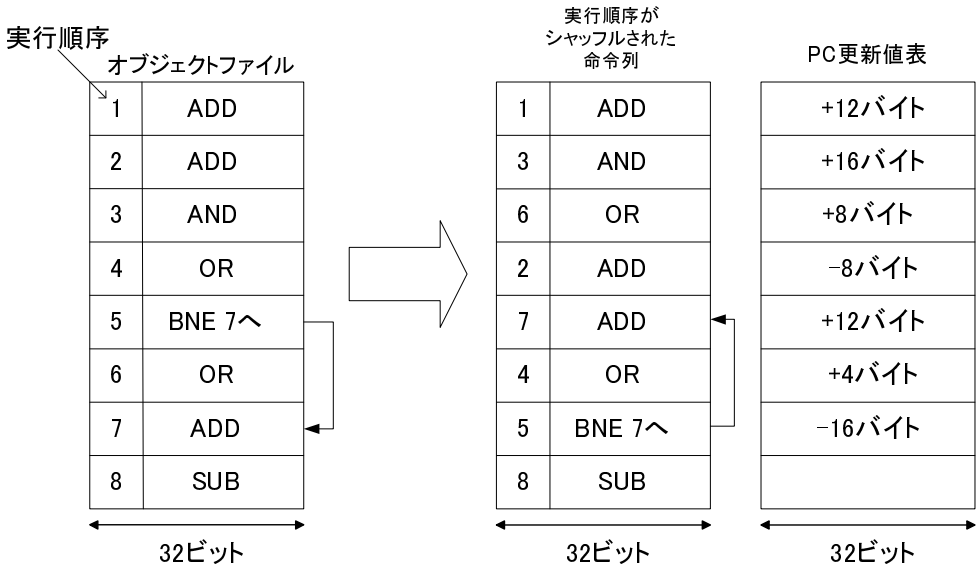

図 9　PC の増加値の多様化の方法

まず，元のオブジェクトファイルから命令列をシャッフルしたものと，それを正しい順序で実行するための PC の更新値表をソフトウェア側で生成する．更新値表は，RAM や ROM で実装を行なう．また，更新値表には，命令長と同じだけのメモリ幅が必要である (MIPS 命令セットならば 32 ビット)．メモリ上の更新値表の先頭アドレスをレジスタに保持しておき，PC+レジスタ値が更新値表の対応する増分を指すようにする．

本手法では分岐命令の際に問題がある．図 9 では，5 番目に条件分岐命令があり 7 番目の ADD 命令へ分岐する可能性がある．分岐命令で分岐が発生しなかった場合は，PC は更新値表の値で更新される．この場合は，PC に対して-16 バイト加算され PC は次の 6 番目の OR 命令を指すアドレスになる．分岐命令によって分岐が発生した場合は，PC は命令列の飛び先のアドレスを示す必要があるため，7 番目の ADD 命令のアドレスを示さなければならない．しかし，命令列のシャッフルによって飛び先の命令コードのアドレスが変わってしまっている．そのためにシャッフルした命令列を生成する際には分岐命令のオフセット部分を書き換える必要がある．また，分岐が発生したときも PC を共有してい

るため，更新値表も飛び先の命令の更新値 (この場合+12 バイト) を指すことができる．

4.3 本手法の考察

本手法の利点として，攻撃者がインジェクション攻撃を試みたときに不正コードが任意に実行されない点がある．命令の実行順序がシャッフルされているため，攻撃者が任意の順序では実行できない．ただし，攻撃者が更新値表を書き換えることで可能となる．また，ソフトウェアの盗用の面では，命令列の実行順序が変わっているために更新値表の知識なしには正しい実行順序はわからない．

本手法は命令列が長ければ長いほど自由度が大きくなる．自由度は順序を入れ替える命令の数の階乗分ある．また，今回は固定長命令で考えたが可変長の命令にも適用することは可能である．ただし，可変長命令ではコンバータ等の実装が複雑になり NOP 命令によって更新値表に冗長性が増す可能性が高い．

本手法は，3 章で述べた命令セットの多様化手法や，Kc らの手法などとも併用が可能である．しかし，問題点として更新値表に命令列と同等のメモリ資源を要することや，本手法単体だけでは命令コードや更新値表が隠蔽されないため，何らかの対策をとらないと容易に解読されてしまうことがある．

攻撃に対する安全性が高いと言える手法ではないため，本手法の改良は今後の課題である．

5. 命令セットの多様化手法の FPGA による実装

本章では，命令セットの符号化の自由度による多様化手法の FPGA 上での実装を考え，Plasma を使用した実装例とその評価結果について述べる．

5.1 FPGA 実装

本手法の評価には，3 章で行った命令セットの自由度の評価より，組込みシステムでよく用いられ，命令形式が簡潔で固定長命令な MIPS 命令セットを選択した．

実装プラットホームには Xilinx 社の Spartan3[14] を使用し，実装対象として MIPS 命令セットのソフトコアプロセッサである Plasma[15] を採用した．測定条件を表 3 に示す．

表 3 測定条件

CPU	Athlon XP 2500+
メモリ	1GB
OS	Windows XP SP2
CAD software	Xilinx ISE 8.2i
デバイス	xc3s2000
パッケージ	fg456
Speed Grade	4

Plasma には MIPS 命令セットのサブセットとして 94 の命令が定義されている．そのうち Plasma で実装されている命令が 62 命令，未実装の命令が 32 命令ある．ここでの未実装な命令は，MIPS 命令セットでは未実装ではないため自由度として考える．Plasma の自由度は，3.36×10^{112} になる．

本手法の実装例として特殊化デザインとメモリマッピングデザインの二つの実装を行なった．また，比較のためにオリジナルの Plasma と Kc ら手法の命令のビット転置デザイン，命令に鍵を XOR するデザインの実装を行なった．

5.1.1 オリジナルデザイン

オリジナルデザインとはPlasmaのことである．ブロック図を図10に示す．

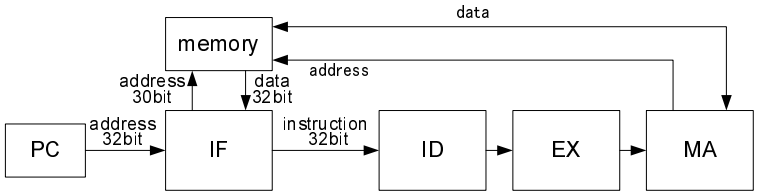

図 10　Plasma オリジナルデザイン

プログラムカウンタ (PC) から得られたアドレスによって，命令フェッチ部分 (IF) でメモリ (memory) から命令がフェッチされ，命令デコード部分 (ID) で命令コードの解釈を行なって，実行部分 (EX) で命令を実行して，メモリアクセス部分 (MA) でメモリへの書き込みや読み込みが行なわれる．

5.1.2 特殊化デザイン

特殊化デザインは，opcodeをランダムに入れ替えた命令コードを実行できるようにするために，オリジナルのID部分を変更したデザインである．

Plasma (図10) のID部分では命令コードのopcodeフィールドのバイナリ値またはopcodeフィールドと拡張フィールドのバイナリ値によって命令の解釈を行なっている．ここで選択された処理がEX部分で実行される．3章で説明したとおりopcodeフィールドや拡張フィールドのバイナリ値を入れ替えることでパーソナリティを変えたものを作ることが可能である．

特殊化デザインを図11に示す．まず，デザインの生成作業を行ないやすくするためにID部分のバイナリ値をシンボリック表現に置き換えた．例えば，ジャンプ命令であるJ命令を示すopcodeフィールドのバイナリ値 **000010** を **INSTRUCTION_J** のように全ての命令の置き換えを行なった．また，シンボリック表現の定義ファイルとしてDefinition Fileを作成した．Definition Fileには，各シンボリック表現に対するバイナリ値が記述されている．

シンボリック表現に対するバイナリ値をランダムに書き換えたDefinition Fileを作り，これを製品ごとで変えることでパーソナリティが異なった製品になる．そのため，パーソナリティを変えた製品を作るにはDefinition Fileを変える必要があり，それにはその都度デザインの論理合成を行なう必要がある．

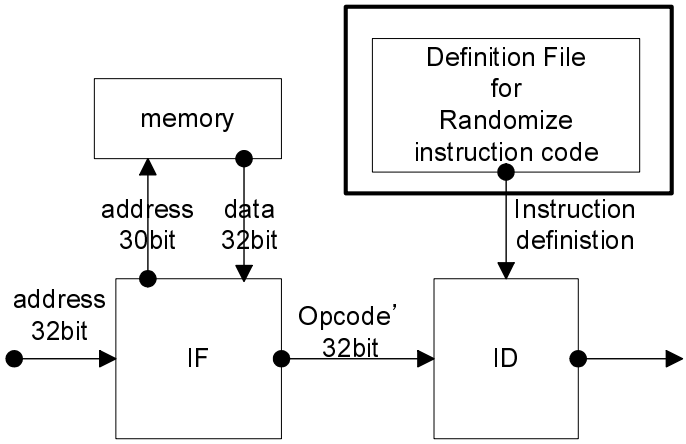

図 11　Plasma 特殊化デザイン

5.1.3　メモリマッピングデザイン

メモリマッピングデザインは，特殊化デザインの Definition File の部分を RAM で実装を行なったデザインである．

オリジナルの IF 部分と ID 部分の間にマッピング部分を追加し，マッピング部分では命令コードを RAM にある変換テーブルによって変換する．

MIPS 命令セットには，opcode フィールドの入れ替えと拡張フィールドである function フィールドと rt フィールドによる入れ替えによってパーソナリティを生成することが可能である．そのため，opcode フィールドによって決まる命令群，opcode と function フィールドによって決まる命令群，opcode と rt フィールドによって決まる命令群の 3 つに別れる．この 3 つの命令群は，opcode によって判定が可能である．

ram にはそれぞれ各フィールド用の変換テーブルが格納されている．ram1 は opcode フィールド，ram2 は function フィールド，ram3 は rt フィールドの変換テーブルが格納されている．メモリマッピングデザインでフェッチされた命令コードは，前述の 3 つのフィールドが対応する ram へのメモリアドレスになっている．

repack 部では次の 3 つの命令コードを生成する．

- case1：opcode フィールドのみを変換した命令コード
- case2：opcode フィールドと function フィールドを変換した命令コード
- case3：opcode フィールドと rt フィールドを変換した命令コード

opcode によってどの case か判別できる．そのため，ram1 には repack 部分で生

成される各 case を選択するためのセレクトビットが格納されている．セレクトビットによって選択された case の命令コードが ID 部分に送られる．

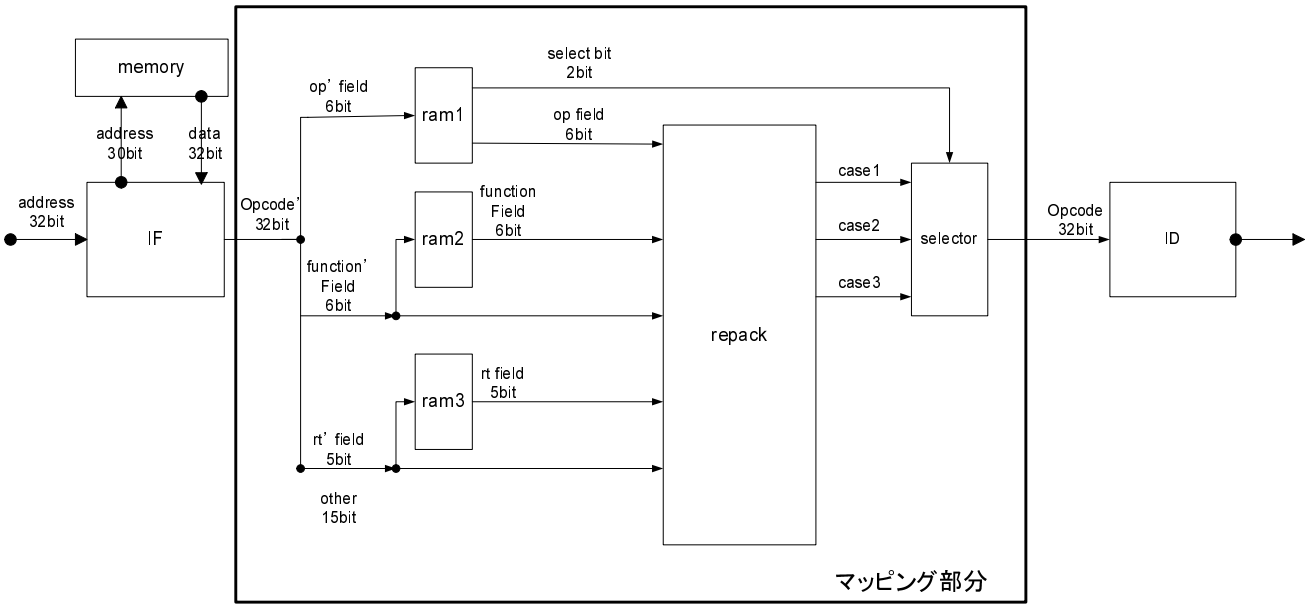

図 12　Plasma メモリマッピングデザイン

メモリマッピングデザインでは，特殊化デザインと違い Definition File を RAM で実装しているため，パーソナリティを変えた製品を作るために論理合成をしなおす必要はない．その点で生産性に優れているデザインと言える．

また，RAM の容量を増やして複数のパーソナリティを備えた製品を作ることも可能である．プロセスごとにパーソナリティの切り替えを行なうことで，ユーザの制限や攻撃に対する耐性を高めることができる．

その点で特殊化デザインより応用の利くデザインと言える．

しかし，特殊化デザインよりマッピング部分や RAM による性能のオーバヘッドは増える．

5.1.4　ビット転置デザイン

ビット転置デザインは，オリジナルの IF と ID の間にビットの転置を元に戻す処理を追加したデザインである[7]．

図 13 のように 32 組の 32:1 のマルチプレクサとセレクト信号用の 160 ビットのレジスタを用意する．各マルチプレクサで転置された命令コードをそれぞれ 5 ビットのセレクト信号により 32 ビットから 1 ビットを選択する．この処理に

より転置された命令コードを元の命令コードに戻す.

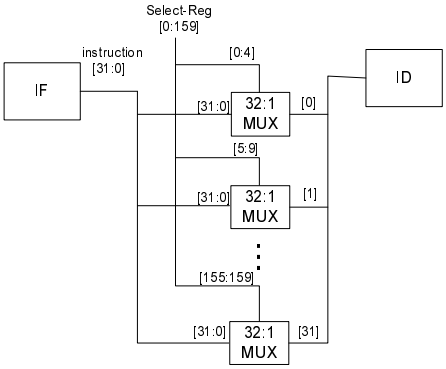

図 13　Plasma ビット転置デザイン

5.1.5　鍵を XOR するデザイン

鍵を XOR するデザインは，オリジナルの IF と ID の間で命令コードに対して 32 ビットの鍵を XOR する処理を追加したデザインである[7)]．

32 ビットの命令コードに対して 32 ビットの鍵を XOR する処理を行なう．これにより，鍵を XOR された命令コードが元の命令コードに復号される．

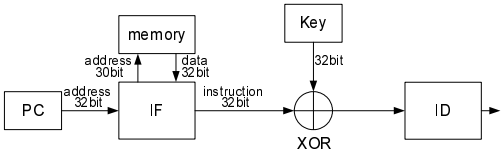

図 14　Plasma 鍵を XOR するデザイン

5.2 評価結果

本節では，各デザインの論理合成の結果について述べる．論理合成の結果を表 4 と 5 に示す.

表 4 Plasma の論理合成結果 (Speed 優先)

	Slice (SliceM)	Block RAM	動作周波数 [MHz]	合成時間 [s]
オリジナル	1911 (128)	4	35.3	437
特殊化 (最大値)	2117 (130)	4	37.9	-
特殊化 (最小値)	1868 (128)	4	31.1	-
特殊化 (平均値)	2009 (128)	4	34.1	479
メモリマッピング	1979 (128)	4	31.2	461
ビット転置	2139 (192)	4	33.2	522
鍵を XOR する手法	1887 (128)	4	34.2	426

表 5 Plasma の論理合成結果 (Area 優先)

	Slice (SliceM)	Block RAM	動作周波数 [MHz]	合成時間 [s]
オリジナル	1481 (128)	4	19.3	302
特殊化 (最大値)	1538 (130)	4	19.5	-
特殊化 (最小値)	1505 (128)	4	16.5	-
特殊化 (平均値)	1522 (128)	4	17.9	336
メモリマッピング	1526 (128)	4	17.4	380
ビット転置	1675 (192)	4	18.1	380
鍵を XOR する手法	1513 (128)	4	18.1	369

スライス数は，論理規模を表し，スライス L とスライス M の合計値である．スライス L は論理記述のみに使用され，スライス M は論理記述や分散 RAM やシフトレジスタで使用される．メモリマッピングデザインでは，分散 RAM を使用している．

特殊化デザインでは，Definition File によって性能にばらつきがあったため，100 個のデザインでの平均値も測定している．これは，命令セットでの命令の定義は最適化されているためと考えられる．また，スライス数，ブロック RAM 数，動作周波数の最大値と最小値を併記した．

また，論理合成の最適化オプションとして動作周波数を優先する Speed 優先とスライス数を優先する Area 優先を使用した．それぞれのオプションの間にはオリジナルのスライス数で 29%，動作周波数で 45%の差がある．そのため，最適化オプションは有効に働いていると言える．

動作周波数を優先した場合 (表 4) ではオリジナルに対して，スライス数で特殊化 (平均値) で 5%, メモリマッピングで 3%, ビット転置で 12%のオーバヘッドがみられた．鍵を XOR する手法ではオリジナルよりも 1%性能が向上した．動作周波数では特殊化 (平均値) で 3%, メモリマッピングで 12%, ビット転置で 6%，鍵を XOR する手法で 3%のオーバヘッドがみられた．鍵を XOR する手法においてスライス数がオリジナルよりも小さくなった点は，1%程度のためおそらく最適化オプションによるものだと思われる．

スライス数を優先した場合 (表 5) ではオリジナルに対してスライス数では特殊化 (平均値) で 3%, メモリマッピングで 3%, ビット転置で 13%，鍵を XOR する手法で 2%のオーバヘッドがみられた．動作周波数では特殊化 (平均値) で 7%, メモリマッピングで 11%, ビット転置で 7%，鍵を XOR する手法で 7%のオーバヘッドがみられた．

5.3 評 価 考 察

最適化オプションをつけたどちらの場合でも，オリジナルに対する性能の低下は少なくなった．スライス数で 3%〜13%，動作周波数で 3%〜12%の性能低下がある．

本手法のデザイン 2 つと Kc ら手法のデザイン 2 つ共にオリジナルに対する性能の低下は十分に少ない．しかし，命令セットの自由度の点では本手法のほうが優れている．

本手法の実装例である 2 つでは，やはりメモリマッピングデザインが特殊化デザインと比べると多少性能が低下する．しかし，今回提案した実装例はまだ改善の余地があるため，性能低下はまだ抑えられるのでないかと考えられる．例えば，メモリマッピングデザインではマッピング部分を IF と ID の間で実装したが，RAM のマッピングによる変換部分は ID に含めることも可能である．

特殊化デザインでは，パーソナリティを変えた製品ごとに論理合成を行なう必要性があるが，メモリマッピングは 1 度の論理合成でよい点など両者のデザインには特性に違いがある．両デザインの性能低下の差は少ないため，用途に合わせて選択することが可能であると考える．

6. おわりに

ISA を多様化するための手法として，命令の符号化の自由度を利用した命令セットの多様化手法を提案した．本手法を 4 つの命令セットで評価し，多くの自由度が得られることがわかった．この自由度は，Kc らの手法で得られる自由度よりもとても多くなり，自由度の点で Kc ら先行手法よりも優れていることを示せた．また，本手法は既存手法に対して直交性をもち併用が可能な手法である．

他の手法の提案として，プログラムカウンタの増加値をプログラムカウンタの更新値表を備えることで多様化する手法を提案した．命令セットの多様化手法や他の手法などと併用ができる手法であるが，その安全性は高いとは言えないため手法の改良を今後の課題としたい．

また，命令セットの多様化手法を FPGA 上に実装した．Plasma に対して，本手法の実装例を 2 つと Kc ら手法の 2 つのデザインを評価した．本手法と Kc らの手法ともに，性能のオーバヘッドは最大で 13%程度と少なくなることがわかった．本手法の特殊化デザインとメモリマッピングデザインでは性能オーバヘッドに余り差がなかったため，用途に合わせて選択することが可能であると考えている．

命令セットの多様化手法のみでは頻度分析攻撃による解読の危険性があるため，より安全性を高めるためにもソフトウェアの難読化や他の手法などとの併用が望ましいと考える．実装の評価より命令セットの多様化手法は，併用を考えることができるだけの低い性能オーバヘッドで実装できることがわかっている．

本論文で ISA を多様化する手法として自由度が多く得られ，低い性能オーバヘッドで実装が容易な新手法を提案することができた．

References

1) 門田暁人，Thomborson, C.：ソフトウェアプロテクションの技術動向 (前編)–ソフトウェア単体での耐タンパー化技術–，情報処理， Vol.46, No.4, pp.431–437 (2005).

2) 門田暁人，高田義広，鳥居宏次：ループを含むプログラムを難読化する方法の提案，電子情報通信学会論文誌， Vol. J80–D–I, No. 7, pp. 644–652 (1997).

3) 神崎雄一郎，門田暁人，中村匡秀，松本健一：命令のカムフラージュによるソフトウェア保護方法，電子情報通信学会論文誌， Vol.J87–A, No.6, pp. 755–767 (2004).

4) Thekkath, D. L.C., Mitchell, M., Lincoln, P., Boneh, D., Mitchell, J. and Horowitz, M.: Architectural Support for Copy and Tamper Resistant Software, *International Conference on Architectural Support for Programming Languages and Operating Systems*, pp.168–177 (2000).

5) Drinic, M. and Kirovski, D.: A Hardware-Software Platform for Intrusion Prevention, *International Symposium on Mecroarchitecture*, pp. 233–242 (2004).

6) Barrantes, E.G., Achley, D.H., Forrest, S. and Stefanovič, D.: Randomized instruction set emulation, *ACM Transactions on Information and System Security*, Vol.8, No.1, pp.3–40 (2005).

7) Kc, G.S., Keromytis, A.D. and Prevelakis, V.: Countering code-injection attacks with instruction-set randomization, *CCS'03*, pp.272–280 (2003).

8) Monden, A., Monsifrot, A. and Thomborson, C.: Tamper-Resistant Software System Based on a Finite State Machine, *Trans. IEICE*, Vol.E88–A, No.1, pp.112–122 (2005).

9) MIPS Technologies Inc.: *MIPS32^{TM} Architecture For Programmers VolumeI: The MIPS32^{TM} Introduction to the MIPS32^{TM} Architecture*,, md00082 revision 2.00 edition (2003).

10) MIPS Technologies Inc.: *MIPS32^{TM} Architecture For Programmers VolumeII: The MIPS32^{TM} Introduction to the MIPS32^{TM} Architecture*,, md00082 revision 2.00 edition (2003).

11) 株式会社ルネサステクノロジ：SH-3, SH-3E, SH3-DSP プログラミングマニュアル (2000).

12) intel: *8080A/8080A-1/8080A-2 8BIT N-CHANNEL MICROPROCESSOR* (1986).

13) ZiLOG: *Z80 Family CPU User Manual* (2005).

14) Xilinx Inc.: Spartan-3 FPGA ファミリ データシート，ds099–1 v1.5 edition (2005).

15) Rhoads, S.: 'Plasma - most MIPS I (TM) opcodes: overview (2006).

http://www.opencores.org/projects.cgi/web/mips/.