社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2009-ARC-181／2009-EMB-11 (14)
2009/1/14

並列／分散処理環境における組込み仮想マシンの実現可能性

矢野 裕章† 中西 正樹†
三輪 忍† 中條 拓伯†

現在，東京農工大学共生科学技術研究院を中心に共生情報工学プロジェクトが推進されている．今後のマルチコアプロセッサや組み込み機器で構成される並列／分散ネットワーク環境において、仮想マシンを構築することで汎用的な計算環境を実現することを目的とする．その大規模分散環境において，利用するプロセッサの台数の増加にしたがって性能を向上させるための組み込み仮想プロセッサの概念を提案する．そして，その実験評価システムについて示し，実装の方向性について述べる．

Feasibility of an Embedded Virtual Machine under Parallel or Distributed Processing Environment

Hirofumu Yano,† Masaki Nakanishi,† Shinobu Miwa† and Hironori Nakajo†

Currently, Symbio-Infomation Technology Project has been conducted in Institute of Symbiotic Science and Technology, Tokyo University of Agriculture and Technology. Under environment of parallel or distributed network using current and future multicore processors and the numerous number of processors coupled into embedded devices or electrical applicances, we aim to implement general purpose computing environment configuring virtual machines in the environments. In the vast distirbuted environment, we propose embedded virtual machines with the increasing number of processors towards higher performance. We describe our experimental evaluation system and directory of our implementation.

1. はじめに

1.1 背景：溢れるプロセッサパワー

1.1.1 マルチコアの現状と今後

現在．単一プロセッサにおける周波数向上の限界から，プロセッサはマルチコア化に向かい，研究レベル，商用レベルにおいてさまざまなプロセッサが提案，商品化されている．

数コア程度のものであれば，これまでの SMP の延長として，従来のソフトウェア資産の継承性からも汎用の PC においてかなり普及しつつある．さらなる処理性能を目指して，コア数は数十コアとなり，やがて 3 桁といったものに移行していくことは容易に想像可能である．

実際，Intel の Larrabee では，1 個のプロセッサで 16〜48 個のコアを計画しており，Intel はそれに先立ち 80 コアの試作にも成功している．また，Tensilica 社の Xtensa を百コア以上搭載したネットワークプロセッサも稼動状態にある．

しかしながら，このような溢れる数のコアを効率よく利用するためには，システムソフトウェアに依存するところが大きく，また．そのシステムソフトウェアをサポートするための有効なハードウェア支援も必要不可欠であり，その技術開発については，さまざまな研究が続けられてはいるが，メモリアーキテクチャなど，まだ多くの課題をかかえている．

1.1.2 モバイル機器の普及

次に，世界的に爆発的な普及をしているプロセッサを搭載したモバイル機器の代表として携帯電話がある．現在，第三世代 (3G) が利用可能であり，老若男女を問わず，個人で複数台を所有する場合も稀ではなく，その出荷台数は 2008 年 12 月時点においては，国内で 1 億台数を超えており，世界規模では，今後さらに普及していくものと考えられる．その計算能力を集約し，利用する環境を整えることが新たな並列分散処理の可能性が出てくる．

1.1.3 その他のプロセッサ

さらに，プロセッサを搭載した電子機器として，2011 年 7 月に完全移行する地上ディジタルテレビ (地デジ TV) があり，今もその完全移行の時期に向けて普及は始まっている．周知の通り，地デジ TV の機能としてネットワーク機能や，それを通じた双方向インタラクションがあり，そこにはこれまでネットワーク分野で培われてきたコンピューティング技術が結集されており，その処理に当然のようにプロセッサが搭載され，

† 東京農工大学
Tokyo University of Agriculture and Technology

各製品は他社との競争を勝ち抜くためにその技術革新の開発に余念がない．実際，東芝社の地デジ TV には．マルチコアプロセッサである Cell Broadband Engine (Cell) が搭載されており，SD 画質を HD 画質へアップコンバートする処理や複数の動画を同時に行う処理を担っている．

また．乗用車の安全性，機能性を高めるために，その車内には多数のプロセッサが搭載されており，実際メルセデス・ベンツ C クラスには 153 個の組み込みプロセッサが搭載されていると言われている．これらは単独で動作するのではなく，FlexRay と呼ばれる車内ネットワークが開発され，これらのプロセッサが通信を行いながらデータの授受が行える環境が確立されつつある．さらに，最近の乗用車では，カーナビゲーションシステムや ETC などといったように，外界とも接続され，その経路を通じて外部サーバとのデータの授受が可能なものとなっているのである．

1.2 余剰プロセッサパワーの有効活用

余剰プロセッサを活用するものとして，Grid Computing があるが，PC とネットワークを活用し，さらに普及の著しいゲーム機をも活用した形で，一部のアプリケーションにおいて成功を収めている．

さらに，現在は Clund Computing として，Google 社，IBM 社，Microsoft 社，HP 社を中心にその概念が確立されつつあり，その実現性とビジネスへの展望についても急激な発展が示されている．

クラウド・コンピューティングにおけるサービスは以下の 3 つの種類に分類される．

- SaaS(Software as a Service) - インターネット経由のソフトウェア機能の提供（グループウェア機能の提供など）
- PaaS(Platform as a Service) - インターネット経由のプラットフォーム機能の提供（仮想マシンの提供など）
- HaaS(Hardware as a Service) - インターネット経由のハードウェアリソースの提供（仮想ディスクの提供など）

この中で，SaaS，HaaS については，一般利用が浸透しつつあるが，計算機パワーの提供ということでは，Paas について克服すべき課題は多い．現状では，巨大かつロバストなデータセンターを保有する Google 社や IBM 社などの特定の巨大企業に負うところが大きい．

以上のように，PC だけでなく，世の中にプロセッサが溢れており，それらのプロセッサが常にすべてが稼動状態にあるわけではなく，休止状態となっているプロセッサパワーを結集し，それを統合するような方式が確立されれば，今後の計算処理，データ管理，その他プロセッサの処理を要するさまざまなコンピューティング環境に変革をもたらすことが可能ではないかと考えた[1]．

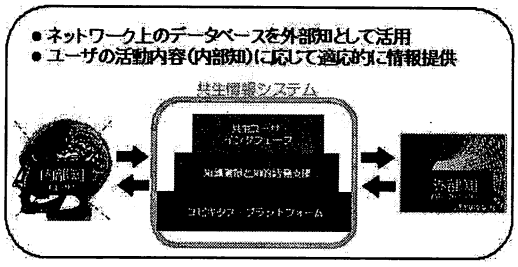

図 1 共生情報工学プロジェクトの概念
Fig. 1 Concept of Symbio-infomation Technology Project.

2. 共生情報工学プロジェクト

このプロジェクトは，平成 18 年度から，文部科学省の特別教育研究経費として，東京農工大学大学院共生科学技術院先端情報科学部門に対して補助された研究費を元に立ち上げたものであり，平成 22 年度まで 5 年間続く予定で，現在も推進を続けている．

人と機器の共生という一貫した概念のもと，直感的で思考を妨げないユーザ・インタフェース，ならびにユーザ適応する専用ハードウェアの開発によって，単なる情報アクセスにとどまらず，人と調和・融合して，ネットワーク上のデータベースを外部知として活用し，人間の知的能力を最大限に発揮させる，真の意味でのユビキタス情報環境を実現することを本プロジェクトの目標としている．

真にユビキタスな情報環境とは，単にいつでもどこでも利用できるというだけでなく，利用するユーザの知的活動を阻害せず，逆に必要性に応じて適応的に情報を提供し，その能力を最大限に発揮させるものであるべきである．しかし，真に人間の知的活動を支援するためには，機能制限によるうわべの使いやすさではなく，必要な情報を必要な形で提供するという，情報処理の本質にかかわる機能が求められる．

すなわち，状況に応じてユーザが必要とする情報をわかりやすい形で提供し知的活動を支援する環境こそが真のユビキタス情報環境であり，その実現のためには，既存のシステムにとらわれない，知的活動支援の観点に立ったハードウェアからインタフェースまで一貫した設計・開発，ならびに心理学や生理学に根ざしつつ工学的視点に立ったシステム評価が必要である．以上の概念を図 1 に示す．

本プロジェクトでは，上記のようにユーザの思考を阻害せず，状況に応じて必要とする情報を必要な形で提供し，人間の知的活動を支援してその能力を最大限に発揮させることができる，人と機器の共生情報環境の実現を目指すことを特徴とする．

現在 3 つ大きな研究グループに分かれ，それぞれの各研究サブテーマを以下に示す．

(1) ユビキタス・プラットフォーム：
- 【超大規模並列処理のための通信機構とプロセッサ・アーキテクチャ】
- 【再生同期制御により臨場感あふれる動画の同時視聴システム】
- 【MAC プロトコルの制御情報を用いたクロスレイヤ制御方式】
- 【知的なセンサによるアプリケーション機能保証のための枠組み開発】

(2) 共生ユーザ・インタフェース：
- 【ユビキタス環境実現のための画像による3次元空間認識】
- 【ユーザコンテキストを用いた一体感のあるオンラインコミュニケーションの実現】
- 【脳波センシングとユビキタス情報提示の融合】
- 【思考にやさしい手書きユーザインタフェースのための手書き認識技術と新しい利用分野の検討】

(3) 知識獲得と知的活動支援：
- 【人の感性から学ぶ人工知能】
- 【ユビキタス語彙学習支援サービスの公開に向けての取り組み】
- 【ヒトと人工物の持続的相互適応】

それぞれのサブテーマについてはネットワーク，ヒューマンインタフェース，ヴァーチャルリアリティの分野において多数の研究成果が発表されつつある．

当初，携帯電話を主要な計算主体とし，その可能性を探ってきたが，現状の携帯機器の今後の方向性において，汎用計算能力よりも動画，音声等のマルチメディアに特化した形で，汎用計算能力の向上については今後は期待できない可能性がある．また，携帯の処理能力を補う形で外部にアクセラレータを搭載する方向で検討を行ってみたが[2]，外部インタフェースとして取り出せるものとしては，SD カード部分や Bluetooth といったバンド幅の限られたものとなり，アクセラレータを接続しても，携帯機器本体との通信に問題が生じる．そこで，携帯機器からさらに計算資源を地上ディジタルテレビやカーエレクトロニクスに搭載された組み込み用のプロセッサに範囲を広げた形での検討に現在修正を行ってる．

3. 目標とする並列／分散処理環境

3.1 並列に処理する粒度

従来，並列処理において最も重要なことに処理粒度があり，利用するシステムの通信バンド幅や通信遅延，利用するプロセッサの処理能力に応じてさまざまな粒度がある．これらをどのように分配し，効率を高めるかといったことがシステム全体の性能向上に対する鍵となる．

そこでまず，多数のプロセッサコアや計算ノードに分配する処理粒度を非現実なものも含め，以下の8段階に分ける．

(1) タスク（プログラム + データ）：広く一般に利用される最も大きな粒度である．

(2) データのみ：個々のノードにおいては，タスクが実行されており，処理を行うデータのみを分配する．

(3) サブルーチン (スレッド)：上記二つはそれぞれ処理としては完結しているものの，ここでの粒度はタスクを小さな処理単位に分割したものとなる．

(4) 命令単位：上記サブルーチンが複数の命令単位であるのに対し，ここでの粒度は個々のノードは単一の命令のみを実行する．

(5) マイクロコード：命令を実装する上においてマイクロコードを用いることにより，過去のソフトウェア資産の継承や，仮想マシンの実装に役立ってきた．ここでは，その個々のマイクロコードの個々の命令を粒度とする．

(6) パイプラインステージ：実行性能を上げるためには，プロセッサの個々の命令を，デコードや実行，メモリアクセスなど，複数のパイプラインの分割して，その各ステージを同時に実行することにより並列性を高めることは周知であるが，その個々のステージを粒度とすることも考えられる．

(7) 計算リソース：プロセッサが備える ALU や浮動小数点ユニットなどの計算資源を粒度とするもの．

(8) ゲートレベル：ハードウェアを構成するゲートレベルを粒度とする．

(9) トランジスタレベル：ゲートを構成する個々のトランジスタを粒度とする．

以上の中で，ゲートレベル，トランジスタレベルについては非現実的ではあるが，その上にあるものは，通信遅延を無視した議論ではあるものの，今後のネットワークの高速化とキャッシュ技術等により，あながち不可能と決め付けるのは性急であると考える．

これらの実現可能性については，現状のノードの処理性能とノード間の通信バンド幅，通信遅延などを考慮した上で決めていく必要がある．

3.2 組み込み仮想マシン

現在，さまざまな計算機ベンダにおいては，サーバ仮想化を中心に，計算機システムの仮想化が，企業のコスト低減，消費電力の削減，セキュリティの向上の観点からも必要不可欠となってきている．仮想化についての歴史は古く，メインフレームにおいてはごく当たり前のものとして利用されてきた．

その仮想化が経てきた過程については二つの大きな方向がある．その過程の一つは，1台のコンピュータ

を仮想化により，論理的に「分割」することによって，複数の OS が実行されたり，あたかも複数台のマシンが稼動しているような仕掛けを投じた方向性である．もう一つは，複数台のコンピュータをシステム内外のネットワークを通じて結合し，仮想的に一つの並列コンピュータに見せかける仮想化への方向がある．ここでは，この後者の方向性について検討を行うものである．この，複数台のコンピュータを仮想化して結合する手法としては，前述のグリッド・コンピューティングがあり，また，CPU 以下の層で仮想化を施したものに仮想マルチプロセッサがある[3]．仮想マルチプロセッサは，単一システムイメージを提供するための仮想マシンモニタであり，ネットワークで結合された複数のマシン上に，共有メモリ型マルチプロセッサマシンを仮想的に構築するものであるすなわち，N 台のシングルプロセッサマシン上に，N プロセッサからなる仮想マルチプロセッサマシンを構築することができ，ユーザが，この仮想マシン上に，マルチプロセッサ対応の OS をインストールしたり，さらに，その OS 上で並列アプリケーションを記述・実行することが可能となる．

また，言語による仮想マシンをハードウェア化したものもある．古くは Pascal の p-code インタプリタをハードウェア化した は　Western Digital 社の MCP-1600 チップセットがある．Java については，Sun Microsystems 社が Java バイトコードをネイティブに実行するプロセッサのサブセット的実装を行った．ARM アーキテクチャの中には Jazelle と呼ばれる Java バイトコード実行モードに切り換える事ができるものもある．

3.3 仮想マシン評価のためのプログラミング言語

Ruby は，まつもとゆきひろ氏により開発されたオブジェクト指向スクリプト言語であり，従来 Perl などのスクリプト言語が用いられてきた領域でのオブジェクト指向プログラミングを実現する．

機能として，クラス定義，ガベージコレクション，強力な正規表現処理，マルチスレッド，例外処理，イテレータ・クロージャ，Mixin，演算子オーバーロードなどがある．

構文は，ALGOL 系を継承しながら，可読性を重視している．Ruby においては整数や文字列なども含めデータ型はすべてがオブジェクトであり，純粋なオブジェクト指向言語といえる．

YARV (Yet Another Ruby VM) は，笹田耕一氏が開発する Ruby 言語処理系であり，Ruby インタプリタの高速化を目指し，開発が進められている．世界最高速の Ruby 処理系を目標としている[4][4]．

Ruby インタプリタの一からの書き直しはせず，Ruby の拡張モジュールとして，既存処理系へのパッチという形で開発が進められ，Ruby 1.9 で採用された．バイトコードインタプリタとして実装されており，Ruby プログラムをバイトコードにコンパイルし，仮想計算機上で実行する．

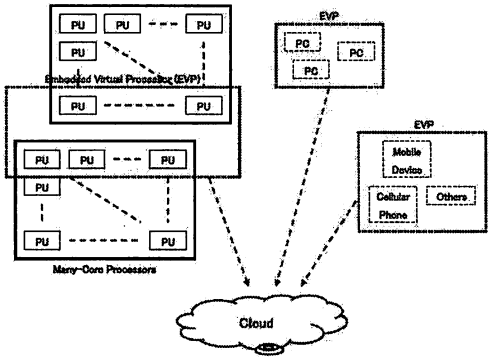

図 2　組み込み仮想プロセッサ
Fig. 2　Embedded Virtual Processors.

そこで，本研究では，まず YARV のハードウェア実装を手がかりとして進めていくこととした．その理由としては，Ruby のユーザが徐々に増えてきたことと，スタックマシンをベースにした形であり，コンパクトな実装が可能であると考えられる点がある．

以下に，我々の目指す組み込み仮想プロセッサの概念を示す．

組み込み仮想プロセッサは，携帯機器や地上ディジタル TV に内蔵されるプロセッサ，車載プロセッサ等から構成されるだけでなく，マルチコア，メニィコアのプロセッサの，複数のコアを，チップをまたがった形でさえ構築可能な方向でその仕様策定を進めている．

4. 実験評価システム

図 3 に現在実装を進めている実験評価システムのブロック図を示す．

YARV は 以下の 5 つの仮想的なレジスタによって制御される[6]．

- PC (Program Counter)
- SP (Stack Pointer)
- CFP (Control Frame Pointer)
- LFP (Local Frame Pointer)
- DFP (Dynamic Frame Pointer)

PC は現在実行中の命令列の位置を示す．SP はスタックトップの位置を示し，CFP，LFP，DFP はそれぞれフレームの情報を示す．

ターゲット実験評価システムは大きくソフトウェア部とハードウェア部の二つに分かれ，それぞれの構成要素について以下に説明する．

- ＜Software 側＞
  - 【Method Dispatch Listener ＋ Method Dispatch List】：
    Ruby の一つの特徴に，Method Dispatch というものがある．これは，send 命令にメゾッド

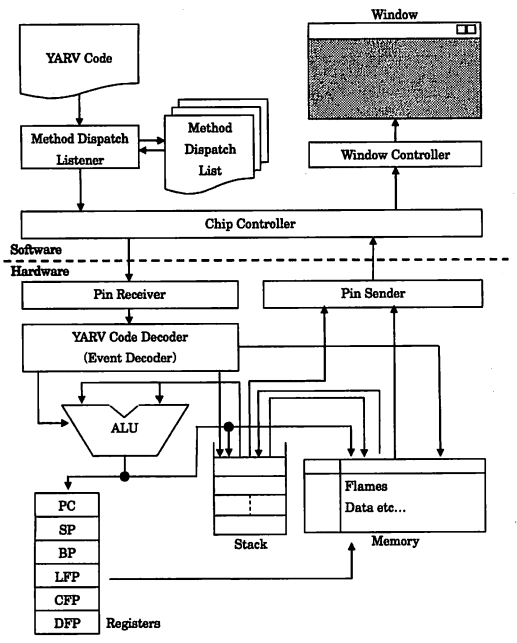

図 3 実験評価システム

Fig. 3 Experimental Evaluation Environment.

名を指定することにより，メゾッドを起動するというスタイルである．たとえば，

```
puts "Hello World!"
```

という 1 行のプログラムがあった場合，

```
0000  putnil
0001  putstring "Hello World"
          #Stack に文字列を追加
0003  send :puts, 1, nil, 8, <ic>
          #puts メゾッドの呼び出し
0009  leave
```

というようになり，puts というメゾッドが，send 命令により呼び出される．これが Method Dispatch である．

プログラムには複数のメゾッドが存在する．そのため，Method Dispatch List にその一覧を保存し，send 命令が行われる時，Method Dispatch Listener が List を参照し，必要なメゾッドの命令を呼び 出せるようにする．

- 【Chip Controller】：
  YARVChip とのアクセスを行う．YARV コードの送信だけではなく，Chip の状態も同時に監視する．

• ＜Hardware 側＞
- 【Pin Receiver】：
  Chip Controller から送信された YARV コードを受信し，YARV Code Decoder に流す．
- 【YARV Code Decoder】：
  Pin Receiver から受け取った YARV コードをデコードし，ALU，Stack，Memory，Registers を操作し，命令を実行させる．
- 【Memory】：
  メモリ領域．ここでは主にスタックフレームを格納する．スタックフレームには 3 種類あり，メゾッド呼び出し時に生成される"メゾッドフレーム"，ブロックを起動したときに生成される"ブロックフレーム "，クラスやモジュールを定義する場合に生成される"クラスフレーム"がある．それぞれのフレームでは，引数や変数が格納されている．

YARV のフル実装を目標に，Ruby を複数ノードでハードウェアにより実行する環境を目指すが，まずは命令を限定した形での実装を進めている．

以下に現在実装を進めている命令群を示す[6]．

この実装には，CoWare 社が提供している Processor Designer により，アーキテクチャ記述言語である LISA による実装と，Verilog-HDL による実装の二つの方向で進めている．

5. 関連研究

Mobile Supercomputer が Michigan 大学を中心に，2004 年に提唱された[7]．高性能化する携帯電話やスマートフォンの計算能力に着目し，今後の計算処理を単なる実行時間や MIPS 値，FLOPS 値だけで判断するのではなく，電力消費を考慮に入れた形で設計すべきであると述べている．提唱はされたものの，その後どのような形で研究が推進されているかについては不明である．

他には，Lawrence Berkeley 国立研究所の Department of Energy において，i-Pod Supercomputer が提案された[8]．これは現状のスーパコンピュータよりも 1000 倍高速なものを目指したものであるが，気象予報のモデリングをターゲットとしている．そのために，既存の消費電力の大きなプロセッサを用いるのではなく，組み込み機器の中でも音楽プレーヤである i-Pod に搭載されているような低消費電力プロセッサを膨大な個数として集約することで性能向上をはかることを目指した．

その構築には 2000 万個の小規模プロセッサを集め，200Peta Flops を目指し，Tensilica 社の Xtensa を活用する方向で研究が進められている．これは，分散した組み込み機器をネットワークで結合するものではなく，低消費電力の小規模プロセッサを集約するという点において，Mobile Supercomputer とは異なるが，組み込み用プロセッサを活用するという点においては我々の提案と類似するところはあり，今後の動向は興味深い．

6. おわりに

本稿では，組み込み機器の現状に触れ，共生情報工学プロジェクトについて紹介し，組み込み仮想プロセッサの概念を提案した．そして，その実験評価システムについて示し，実装の方向性について述べた．

2008 年度末に実験評価システムの完成を目指し，Ruby をベースとして評価データを収集していく予定である．

ネットワークに接続された環境における問題点としては，セキュリティは避けられず，今後その点についても研究を進めていきたい．

謝辞 本研究の一部は文部科学省 共生情報工学推進経費による．

参考文献

1) 中條拓伯，並木美太郎："次々世代携帯端末による超大規模並列処理のためのアーキテクチャ"，情報処理学会研究報告 ARC157 (デザインガイア 2005), pp.67–72 (2005)
2) Satoshi Watanabe, Yoshiyasu Ogasawara, Ippei Tate, Hirofumi Yano and Hironori Nakajo: "Toward Parallel and Distributed Processing on High-Density Network with Mobile Devices", *The 2007 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'07) Vol.2*, pp.794–800 (2007)
3) 金田憲二，大山恵弘，米澤明憲: "単一システムイメージを提供するための仮想マシンモニタ"，情報処理学会論文誌, Vol.47, SIG3 (ACS13), pp.27–.39 (2006)
4) 笹田耕一，松本行弘，前田敦司， 並木美太郎："Ruby 用仮想マシン YARV の実装と評価"，情報処理学会論文誌, Vo.47, SIG2 (PRO 28), pp.57–73 (2006)
5) 笹田耕一，松本行弘，前田敦司， 並木美太郎："Ruby 用仮想マシン YARV における並列実行スレッドの実装"，情報処理学会論文誌, Vo.48, SIG10 (PRO 33), pp.1–16 (2007)
6) 笹田耕一: "http://www.atdot.net/yarv/yarvarch.ja.html"
7) Todd Austin, David Blaauw, Scott Mahlke, Trevor Mudge, Chaitali Chakrabarti and Wayne Wolf: "Mobile Supercomputers", *IEEE Computer, Vol.37, No.5*, pp.81–83 (2004)
8) Michael Wehner, Leonid Oliker and John Shalf: "Towards Ultra-High Resolution Models of Climate and Weather", *International Journal of High Perormance Computing Applications, Vol.22, No.2*, pp.149–165 (2008)

表 1 実装する命令群
Table 1 Instructions to be implemented.

種類	命令名	説明
メモリアクセス	setlocal	スタックトップをローカル変数に保存
	getlocal	ローカル変数をスタックに置く
	setinstancevariable	スタックトップをインスタンス変数 (@val) に保存
	getinstancevariable	インスタンス変数をスタックに置く
	setclassvariable	スタックトップをクラス変数 (@@val) に保存
	gerclassvariable	クラス変数をスタックに置く
メソッドディスパッチ	send	メソッドディスパッチ命令
ALU 命令（最適化計算命令）	opt_plus	スタックトップ 2 つの数値を加算してスタックに置く
	opt_minus	スタックトップ 2 つの数値を減算してスタックに置く
	opt_mul	スタックトップ 2 つの数値を乗算してスタックに置く
	opt_dev	スタックトップ 2 つの数値を除算してスタックに置く
	opt_mod	スタックトップ 2 つの数値の剰余をスタックに置く
	opt_eq	スタックトップ 2 つの数値が等しいか比較し真偽をスタックに置く
	opt_lt	スタックトップ 2 つの数値が小さいか比較し真偽をスタックに置く
	opt_le	スタックトップ 2 つの数値が小さいもしくは等しいか比較し真偽をスタックに置く
	opt_neq	スタックトップ 2 つの数値が等しくないか比較し真偽をスタックに置く
	opt_gt	スタックトップ 2 つの数値が大きいか比較し真偽をスタックに置く
	opt_ge	スタックトップ 2 つの数値が大きいもしくは等しいか比較し真偽をスタックに置く
	opt_ltlt	スタックトップ 2 つの数値をシフト演算しスタックに置く
スタック操作	putobject	定数をスタックに置く
	putnil	nil（null）値をスタックに置く
	pop	スタックトップを破棄
	dup	スタックトップをコピーし，スタックに置く
ジャンプ命令	jump	ジャンプ命令
	branchif	スタックトップが真ならばジャンプ
	branchunless	スタックトップが偽ならばジャンプ
定義命令	defineclass	クラスを定義
	definemethod	メソッドを定義
その他	leave	メソッド，プログラム等の終了