9 情技応第 888 号

高度情報化支援ソフトウェア育成事業

自己反映計算に基づく Java 言語用の
開放型 Just-in-Time コンパイラ OpenJIT の研究開発

結合試験仕様書

平成 11 年 1 月

富士通株式会社

目次

第 1 章　概要 **1**

1.1 目的 1

1.2 試験対象 3

1.2.1 OpenJIT フロントエンドシステム 8

1.2.2 OpenJIT バックエンドシステム 23

第 2 章　試験方針 **34**

2.1 試験方針 34

2.2 用語 35

第 3 章　試験環境 **36**

3.1 OpenJIT フロントエンドシステム 36

3.2 OpenJIT バックエンドシステム 38

第 4 章　試験項目 **39**

4.1 OpenJIT フロントエンドシステム 39

4.1.1 OpenJIT コンパイラ基盤機能 39

4.1.2 OpenJIT バイトコードディスコンパイラ機能 42

4.1.3 OpenJIT クラスファイルアノテーション解析機能 47

4.1.4 OpenJIT 最適化機能 51

4.1.5 OpenJIT フローグラフ構築機能 55

4.1.6 OpenJIT フローグラフ解析機能 57

4.1.7 OpenJIT プログラム変換機能 59

4.2 OpenJIT バックエンドシステム 61

4.2.1 OpenJIT ネイティブコード変換機能 61

4.2.2 OpenJIT 中間コード変換機能 64
4.2.3 OpenJIT RTL 変換機能 67
4.2.4 OpenJIT Peephole 最適化機能 69
4.2.5 OpenJIT レジスタ割付機能 71

第 5 章 試験方法 73

5.1 OpenJIT フロントエンドシステム 73
5.1.1 OpenJIT コンパイラ基盤機能 73
5.1.2 OpenJIT バイトコードディスコンパイラ機能 88
5.1.3 OpenJIT クラスファイルアノテーション解析機能 103
5.1.4 OpenJIT 最適化機能 116
5.1.5 OpenJIT フローグラフ構築機能 123
5.1.6 OpenJIT フローグラフ解析機能 142
5.1.7 OpenJIT プログラム変換機能 159
5.2 OpenJIT バックエンドシステム 172
5.2.1 OpenJIT ネイティブコード変換機能 172
5.2.2 OpenJIT 中間コード変換機能 195
5.2.3 OpenJIT RTL 変換機能 220
5.2.4 OpenJIT Peephole 最適化機能 237
5.2.5 OpenJIT レジスタ割付機能 254

付録 A 試験方法補足 267

A.1 OpenJIT フロントエンドシステム 267
A.1.1 ディスコンパイル機能用テストドライバ 267
A.1.2 OpenJIT クラスファイルアノテーション解析機能 279
A.1.3 最適化機能用テストドライバ 283
A.1.4 プログラム変換機能用テストドライバ 285
A.1.5 OpenJIT フロントエンド用テストドライバで使用されているその他のクラス 292
A.2 OpenJIT バックエンドシステム 389
A.2.1 メソッド情報取得試験用クラス 389
A.2.2 バイトコードアクセス試験用クラス 390

A.2.3 生成コードメモリ管理試験用クラス 391
A.2.4 中間言語変換試験用クラス 392
A.2.5 バックエンド中間コード試験用クラス 393
A.2.6 コントロールフロー解析試験用クラス 394

第 1 章

概要

1.1 目的

デジタル技術が普遍性を持つ今日，従来の計算技術は急速に陳腐化し，新たな計算環境に適した汎用性のある技術を我が国が研究開発することが大いに求められている．例えば，広域ネットワーク，マルチメディア環境，NC，並びに電子商取引などの新しいアプリケーションにおいては，可搬性の高いプログラムが要求されており，各国ともその技術開発に凌ぎを削っている．特に，インターネット及びイントラネットを中心とした互換性が要求される環境，あるいは組み込み機器のように計算資源が限定されている環境においては，Java 言語に代表される機器間で高い可搬性を有する言語が重要視されてきている．

Java 言語では，バイトコードのコンパクトでかつ可搬性のあるプログラムの中間形式から，必要な部分を実行時にネイティブコードにコンパイルし，実行速度を向上させる Just-In-Time (JIT) コンパイラが開発されているが，技術フレームワークの欠如，JIT 自身の可搬性の欠如，最適化技術の未発達を含む問題が指摘されている．たとえば，「最適化」は通常速度の最適化であり，限られたメモリやその他の計算資源のもとで，最大の効率を得るという，組込み型のアプリケーションで必要な "Resource-Efficient"(資源効率の高い) 計算の最適化はなされない．さらに，今後様々な計算環境へ適合するため，(1) 個々のアプリケーション及び計算環境に特化した最適化と，(2) 計算環境とアプリケーションに応じたコンパイルコードの拡張が必要になってくるが，従来型の JIT は (1) は汎用性のある最適化しか行わず，また，(2) に対しては，言語の拡張や新規の機能に対応してコード生成の手法を変えるようなカスタム化は不可能でり，Java の利便性と性能に関して大いに妨げとなっている．

本開発では従来型のコンパイラ技術とは異なる，自己反映計算 (リフレクション) の理論に基づいた “Open Compiler”(開放型コンパイラ) 技術をベースとして，アプリケーションや計算環境に特化した言語の機能拡張と最適化が行える JIT コンパイラのテクノロジ “OpenJIT” を研究開発する．開発のターゲットは実用性や広範な適用性を考慮して Java 言語とするが，技術的には他の同種のプログラム言語にも適用可能である．本研究開発により，我が国がこの分野でリーダーシップをとり，我が国が得意とする組み込み機器，マルチメディア機器，並列科学技術計算などにおいて次世代の基盤技術を持つことを目標とする．

結合試験の目的は，上記の目標に基づき開発した OpenJIT コンパイラシステムが構造仕様書で記述した仕様を満たしていることを，そのそれぞれのサブプログラムを結合して試験を行うことで確認することとする．

1.2 試験対象

本システムの構成を図 1.1 に示す．

本システムは大きく OpenJIT フロントエンドシステムと OpenJIT バックエンドシステムの二つに分けられる．OpenJIT フロントエンドシステムでは，Java のバイトコードを入力とし，高レベルな最適化を施して再びバイトコードを出力する．OpenJIT バックエンドシステムでは，OpenJIT フロントエンドシステムによって得られたバイトコードに対して，より細かいレベルでの最適化を行いネイティブコードを出力する．

図 1.2，図 1.3に OpenJIT フロントエンドシステムと OpenJIT バックエンドシステムを構成する機能の一覧を示す．ただし，OpenJIT バックエンドシステム内の OpenJIT SPARC プロセッサコード出力モジュール，OpenJIT ランタイムモジュールは契約の対象外である．

これらは更に次のに示す機能より構成されている．

- OpenJIT フロントエンドシステム
  - OpenJIT コンパイラ基盤機能
  - OpenJIT バイトコードディスコンパイラ機能
  - OpenJIT クラスファイルアノテーション解析機能
  - OpenJIT 最適化機能
  - OpenJIT フローグラフ構築機能
  - OpenJIT フローグラフ解析機能
  - OpenJIT プログラム変換機能
- OpenJIT バックエンドシステム
  - OpenJIT ネイティブコード変換機
  - OpenJIT 中間コード変換機能
  - OpenJIT RTL 変換機能
  - OpenJIT Peephole 最適化機能

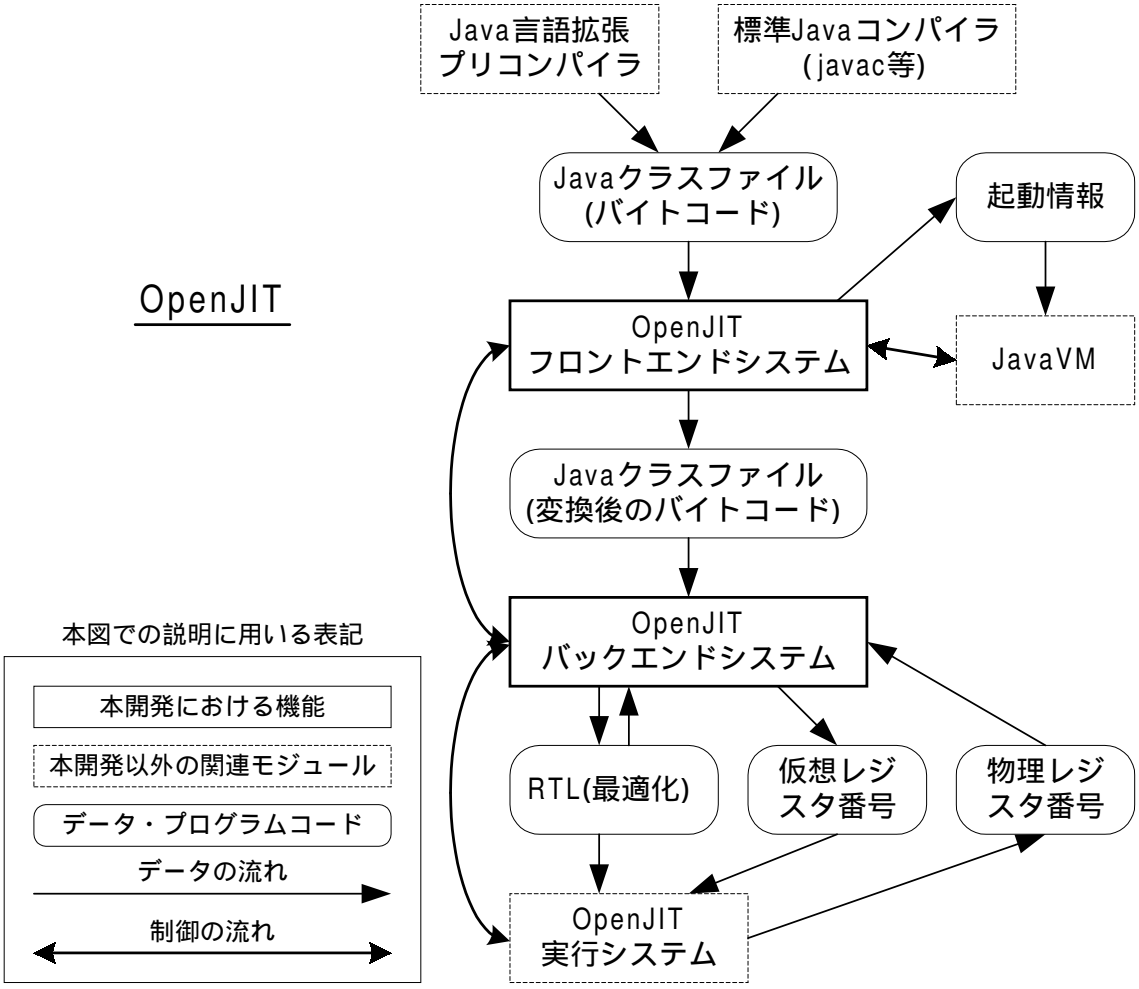

図 1.1: OpenJIT コンパイラシステムの構成図

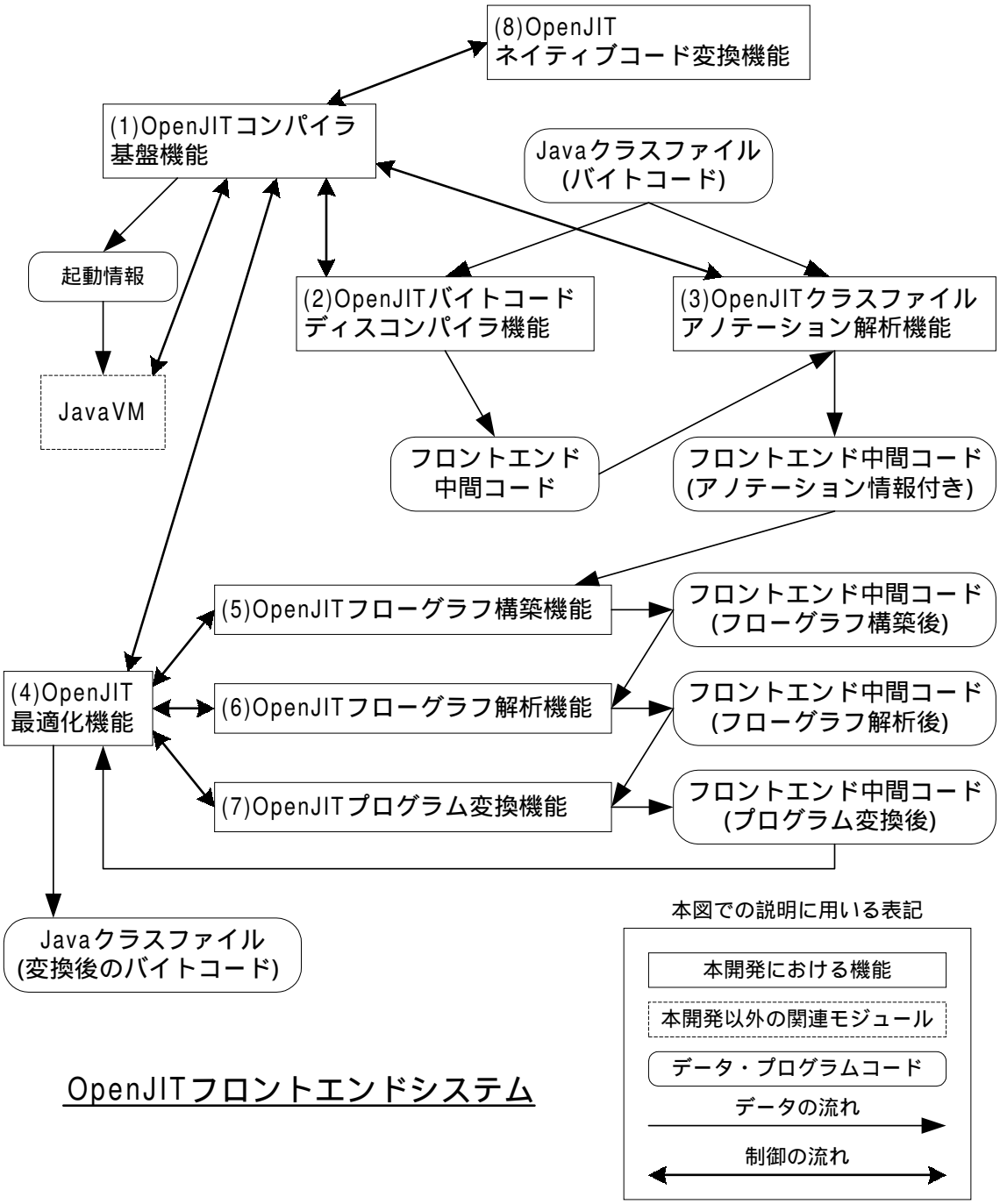

図 1.2: OpenJIT フロントエンドシステム

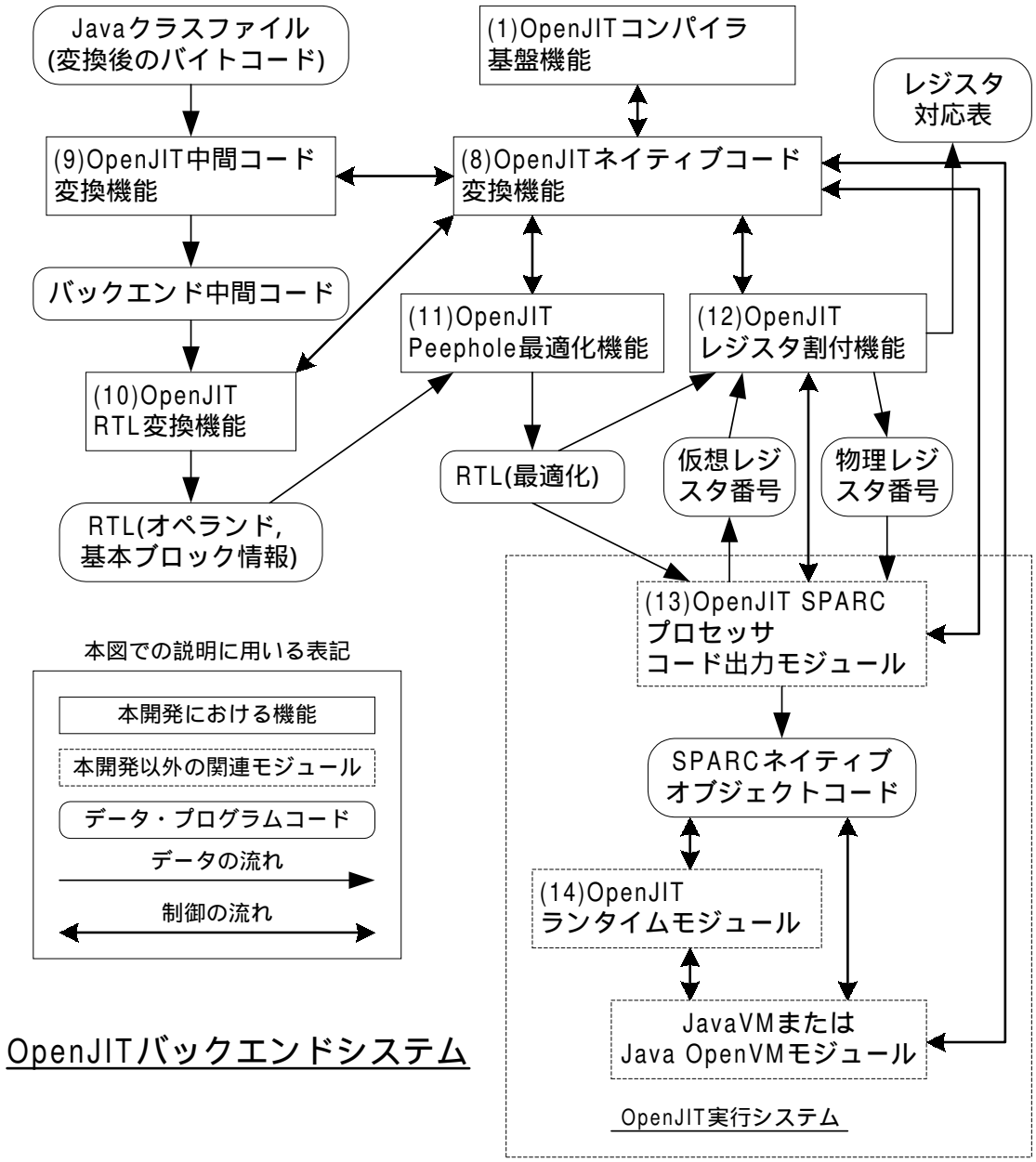

図 1.3: OpenJIT バックエンドシステム

- OpenJIT レジスタ割付機能

以下では，これらの機能概要および機能構成を示す．

1.2.1 OpenJIT フロントエンドシステム

OpenJIT フロントエンドシステムでは，基本的に与えられたバイトコードから，最適化および拡張を施したバイトコードへの変換を行う．クラスファイルに内在する Java のバイトコードを入力とし，高レベルな最適化およびプログラム変換を施して，再びバイトコードを出力する．

まず，OpenJIT バイトコードディスコンパイラ機能は，与えられたバイトコード列をフロー解析して，逆変換することにより，AST を得る．この際には，与えられたバイトコード列から，元のソースプログラムから生成されるコントロールグラフのリカバリを行う技術を開発する．

同時に，OpenJIT クラスファイルアノテーション機能により，このクラスファイルのアトリビュート領域に何らかのアノテーションが付記されていたときに，その情報を得る．たとえば，バイトコードへコンパイルしたときの高レベルな解析情報が，クラスファイルに付記されていることが考えられる．特に重要なのは，クラスファイル自身では得ることが難しいグローバルな解析情報であり，具体的には各コールサイトにおけるディスパッチ可能なクラスが挙げられる．この情報は，AST 上の付加情報として用いられる．

次に，得られた AST に対し，OpenJIT 最適化機能によって，最適化が施される．最適化に必要な情報は，OpenJIT フローグラフ構築機能，OpenJIT フローグラフ解析機能により抽出される．最適化時のプログラム変換は，OpenJIT プログラム変換機能が司って実施され，変換後のバイトコードがバックエンドシステムに出力される．

(1) OpenJIT コンパイラ基盤機能

OpenJIT コンパイラ基盤機能 (図 1.4) は，OpenJIT 全体の基本動作を司る．

Sun の JDK においては，Java Native Code API(Application Programmer's Interface) というコンパイラに対するインタフェースが用意されている．この API は JVM のインタプリタにネイティブコード生成を組み込むために用意されたものである．今回開発する OpenJIT コンパイラでは，この API に基づくことにより JDK に準拠の VM に OpenJIT コンパイラを組み込むことができる．この JIT コンパイラは JVM から必要なときに読み込まれ動作する．

また，OpenJIT コンパイラ基盤機能は以下の小機能から構成される．

- OpenJIT 初期化部

  OpenJIT フロントエンドシステムの各機能，バックエンドシステムの各機能の初期化を指示し，OpenJIT システムの初期化を行う

- OpenJIT コンパイラフロントエンド制御部

  OpenJIT フロントエンドシステムの各機能の起動および制御を行う．

- OpenJIT JNI API 登録部

  バックエンド用の JNI (Java Native Interface) の API を登録する．

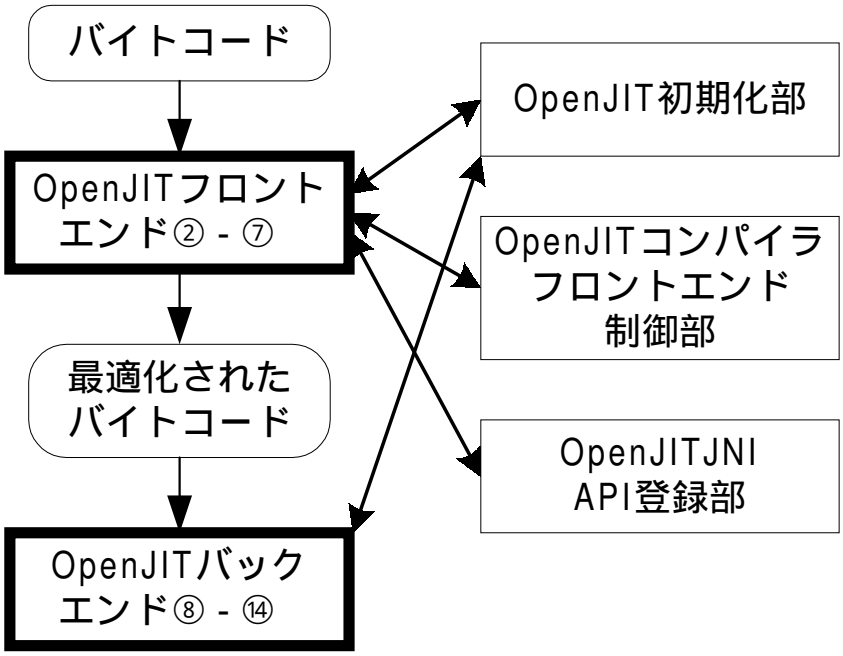

図 1.4: OpenJIT コンパイラ基盤機能

(2) OpenJIT バイトコードディスコンパイラ機能

OpenJIT バイトコードディスコンパイラ機能 (図 1.5) は，Java のクラスファイルに含まれるバイトコードを，いわゆる discompiler 技術により，コンパイラ向けの中間表現に変換する．

OpenJIT バイトコードディスコンパイラ機能は，Java のクラスファイルのそれぞれのバイトコードを，いわゆる discompiler 技術により，バイトコードレベルからコントロールフローグラフ，AST(抽象構文木) を含む抽象化レベルのプログラム表現を復元し，以後の OpenJIT の各モジュールの操作の対象とするような処理を行なう．

また，OpenJIT バイトコードディスコンパイラ機能は以下の小機能から構成される．

- バイトコード解析部

  バイトコードを入力とし，その解析をコントロールグラフ出力部，AST 出力部に指示する．

- コントロールグラフ出力部

  バイトコード解析部に入力されたバイトコードに対し，そのコントロールフローグラフを出力する．

- AST 出力部

  バイトコード解析部に入力されたバイトコードに対し，対応する Java の AST を出力する．

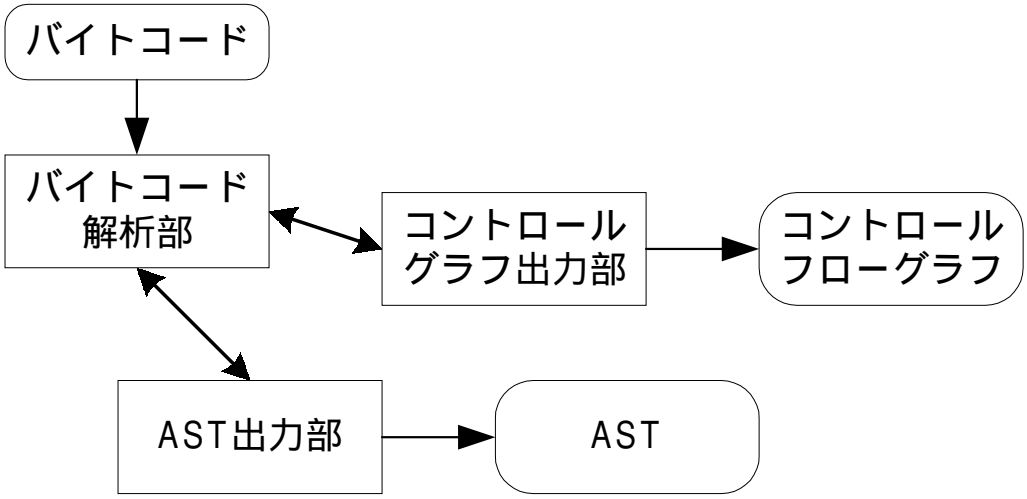

図 1.5: OpenJIT バイトコードディスコンパイラ機能

(3) OpenJIT クラスファイルアノテーション解析機能

OpenJIT クラスファイルアノテーション解析機能 (図 1.6) は，プログラムグラフ (AST) に対して，コンパイル時に適切な拡張された OpenJIT のメタクラスを起動できるようにする．

OpenJIT クラスファイルアノテーション解析機能は，アノテーション情報を解析し，OpenJIT ディスコンパイラ機能が生成したプログラムグラフ (AST) に対して，コンパイル時に適切な拡張された OpenJIT のメタクラスを起動できるようにする．

また，OpenJIT クラスファイルアノテーション機能は以下の小機能から構成される．

- アノテーション解析部

  アノテーションを付記したクラスファイルおよび AST を入力とし，クラスファイルに付加されたアノテーションを解析して，必要に応じてメタクラス制御部，アノテーション登録部を制御する．

- アノテーション登録部

  解析されたアノテーションを AST に対する付加データとして登録して，追加情報を付加した AST を出力する．

- メタクラス制御部

  アノテーションにしたがって，適切なメタクラスが起動されるように制御を行う．

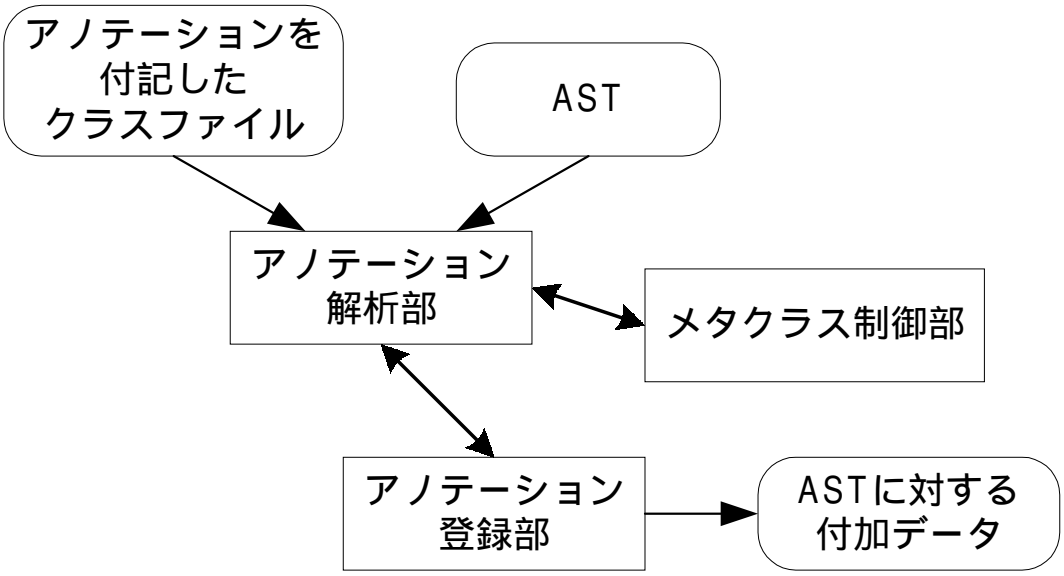

図 1.6: OpenJIT クラスファイルアノテーション機能

(4) OpenJIT 最適化機能

OpenJIT 最適化機能 (図 1.7) は，各種解析モジュールおよびプログラム変換モジュールを用い，プログラム最適化を行なう．

OpenJIT 最適化機能は，OpenJIT フローグラフ構築機能，OpenJIT フローグラフ解析機能，および OpenJIT プログラム変換機能を用い，プログラム最適化を行なう．OpenJIT コンパイラには，標準的なコンパイラの最適化を含む最適化ライブラリ構築のためのサポートが準備される．

また，OpenJIT 最適化機能は以下の小機能から構成される．

- 最適化制御部

  OpenJIT フローグラフ構築機能，OpenJIT フローグラフ解析機能，OpenJIT プログラム変換機能，バイトコード出力部を制御して，入力のバイトコード，AST，コントロールフローグラフから，最適化されたバイトコードの生成を指示する．

- バイトコード出力部

  OpenJIT プログラム変換機能により最適化を行ったバイトコードを出力する．

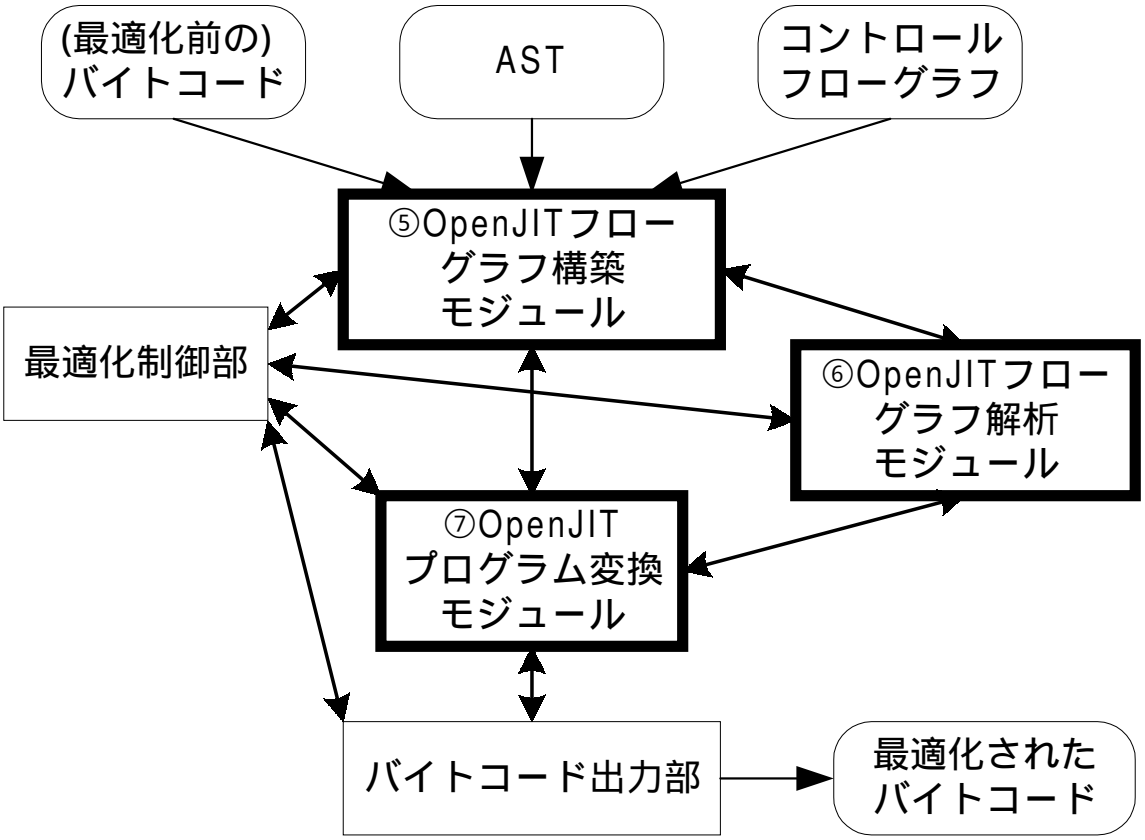

図 1.7: OpenJIT 最適化機能

(5) OpenJIT フローグラフ構築機能

OpenJIT フローグラフ構築機能 (図 1.8) は，AST およびコントロールフローグラフを受け取り，対応するデータ依存グラフ，コントロール依存グラフ，などのフローグラフを出力する．

OpenJIT フローグラフ構築機能は，AST およびコントロールフローグラフを受け取り，対応するデータ依存グラフ，コントロール依存グラフ，を含むフローグラフを出力する．また，クラスファイル間のクラス階層情報情報を得られる場合は，クラスファイルの関係を読み込み，オブジェクト指向解析用のクラス階層グラフも出力する．

また，OpenJIT フローグラフ構築機能は以下の小機能から構成される．

- AST 等入力部

  AST，コントロールフローグラフ，クラスファイル間のクラス階層情報を入力し，中間形式に変換して，その情報をもとにデータフローグラフ構築部，コントロール依存グラフ構築部，クラス階層解析部にそれぞれ処理を指示する．

- データフローグラフ構築部

  AST 等入力部に入力されたプログラム情報からデータフローグラフを構築して出力する．

- コントロール依存グラフ構築部

  AST 等入力部に入力されたプログラム情報からコントロール依存グラフを構築して出力する．

- クラス階層解析部

  AST 等入力部に入力されたプログラム情報からクラス階層解析を行い，その情報を付加したクラス階層グラフを構築して出力する．

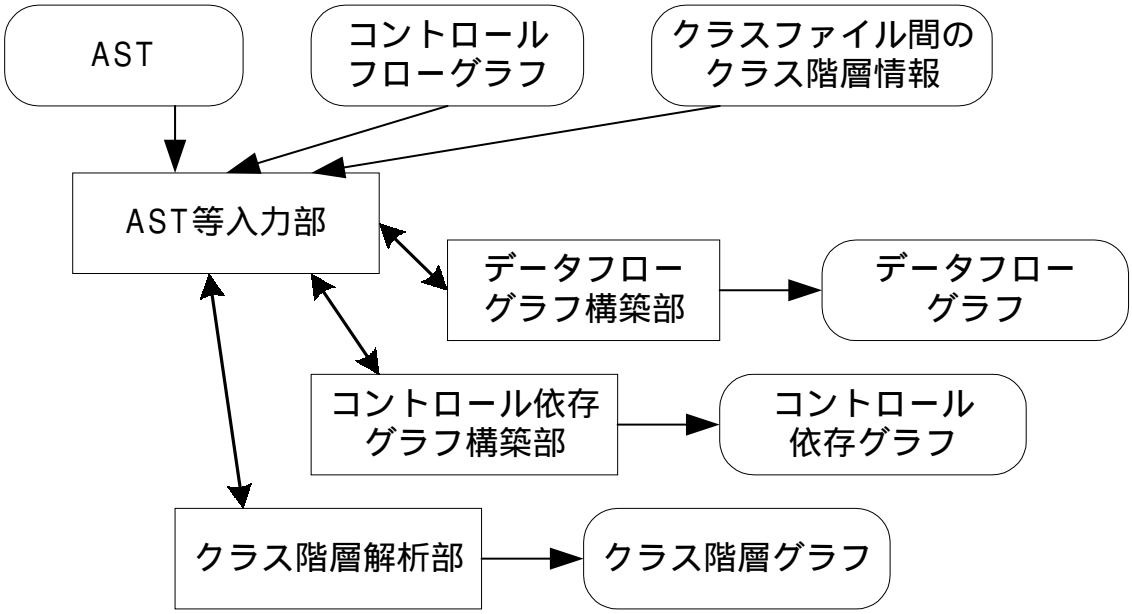

図 1.8: OpenJIT フローグラフ構築機能

(6) OpenJIT フローグラフ解析機能

OpenJIT フローグラフ解析機能 (図 1.9) は，フローグラフ構築モジュールで構築されたプログラム表現のグラフに対し，グラフ上の解析を行なう．

ここでは，OpenJIT フローグラフ構築機能で構築されたプログラム表現のグラフに対し，グラフ上の解析を行なう．基本的には，一般的なグラフのデータフロー問題として定式化され，トップダウンおよびボトムアップの解析のベースとなる汎用的なアルゴリズムをサポートする．具体的には，グラフ上のデータフロー問題，マージ，不動点検出，などの一連のアルゴリズムがメソッド群として用意される．

また，OpenJIT フローグラフ解析機能は以下の小機能から構成される．

- データフロー関数登録部

  データフロー解析に用いるデータフロー関数を登録する．

  データフロー関数群はこのサブモジュールに含まれる．

- フローグラフ解析部

  コントロールフローグラフ，コントロール依存グラフ，データフローグラフ，クラス階層グラフを入力として，登録されたデータフロー関数を用いてデータフロー解析を行う．

- 不動点検出部

  フローグラフ解析部に入力されたデータフローグラフの不動点を検出する．

- クラス階層解析部

  フローグラフ解析部に入力されたクラス階層グラフと，データフローグラフの解析結果から，クラス階層の解析を行い，その結果を出力する．

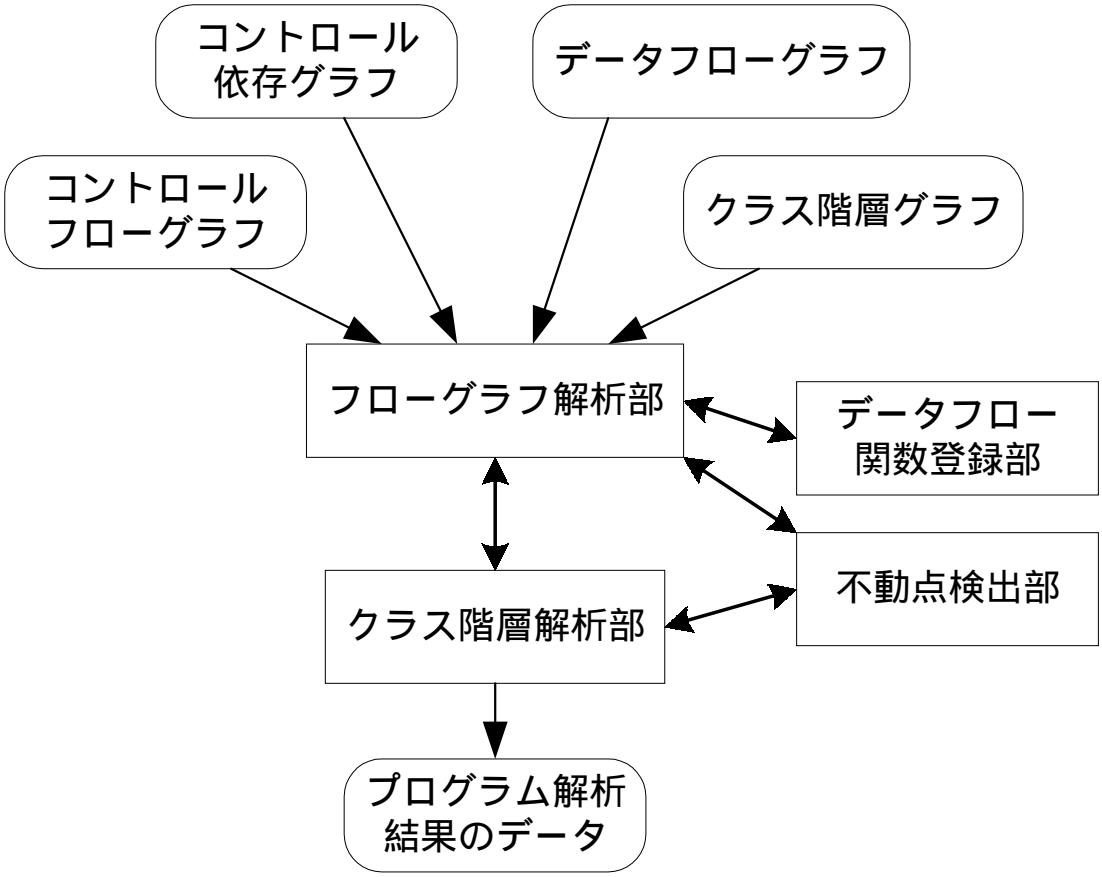

図 1.9: OpenJIT フローグラフ解析機能

(7) OpenJIT プログラム変換機能

OpenJIT プログラム変換機能 (図 1.10) は，OpenJIT フローグラフ解析機能の結果やユーザのコンパイラのカスタマイゼーションに従って，プログラム変換を行なう．

OpenJIT プログラム変換機能では，OpenJIT フローグラフ解析機能の結果やユーザのコンパイラのカスタマイゼーションに従って，プログラム変換を行なう．プログラム変換のためには，AST の書き換え規則がユーザによって定義され，AST 上のパターンマッチが行われ，適用された規則に従ってプログラムの書き換えが行われる．書換え規則自身，全て Java のオブジェクトとして定義され，ユーザはあらかじめ書換え規則を定義して，OpenJIT プログラム変換機能に登録しておく．

また，OpenJIT プログラム変換機能は以下の小機能から構成される．

- AST パターンマッチ部

  AST とプログラム解析結果のデータを入力として，変換ルールのパターンマッチを行う．

- AST 変換部

  パターンマッチした変換規則を用いて，AST の書き換えを行って変換された AST を出力する．

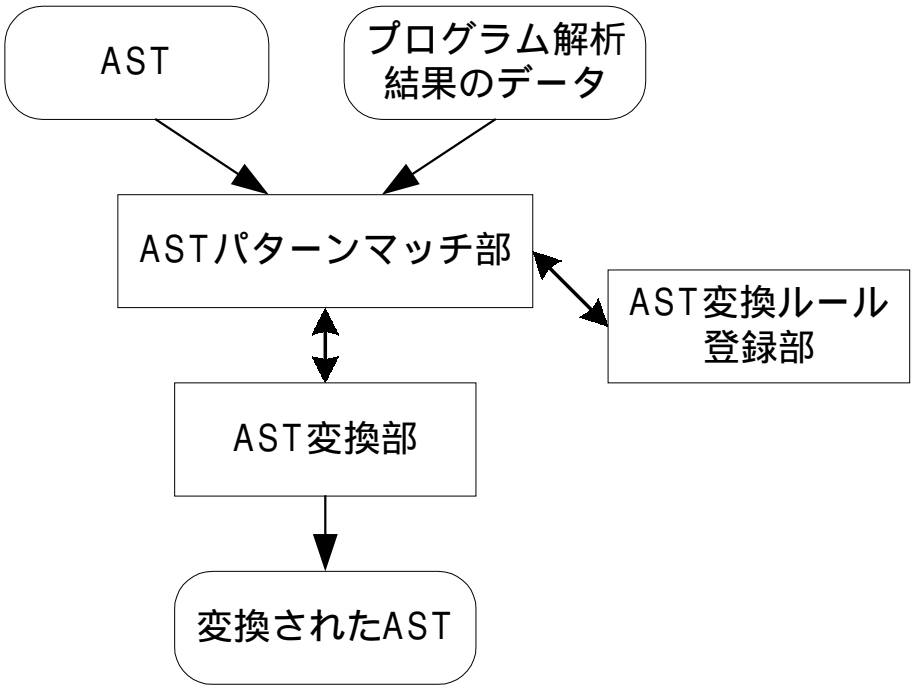

図 1.10: OpenJIT プログラム変換機能

1.2.2 OpenJIT バックエンドシステム

OpenJIT フロントエンドシステムによって最適化されたバイトコード列に対し，OpenJIT バックエンドシステムは以下の技術を用いて，さらなる最適化処理を行い，ネイティブコードを出力する．

OpenJIT ネイティブコード変換機能はバックエンド系処理全体の抽象フレームワークであり，OpenJIT バックエンドシステムの各機能のインタフェースを定義する．このインタフェースに沿って具体的なプロセッサに応じたクラスでモジュールを記述することにより，様々なプロセッサに対応することが可能となる．

OpenJIT 中間コード変換機能によって，バイトコード列からスタックオペランドを使った中間言語へと変換を行なう．バイトコードの命令を解析して分類することにより，単純な命令列に展開を行う．

得られた命令列に対し，OpenJIT RTL 変換機能は，このスタックオペランドを使った中間言語からレジスタを使った中間言語 (RTL) へ変換する．バイトコードの制御の流れを解析し，命令列を基本ブロックに分割する．バイトコードの各命令の実行時のスタックの深さを計算することで，スタックオペランドから無限個数あると仮定した仮想的なレジスタオペランドに変換する．

次に，OpenJIT Peephole 最適化機能によって，RTL の命令列の中から冗長な命令を取り除く最適化を行ない，最適化された RTL が出力され，最後に OpenJIT SPARC プロセッサコード出力モジュールにより，SPARC プロセッサのネイティブコードが出力される．OpenJIT SPARC プロセッサモジュールは，ネイティブコード生成時のレジスタ割り付けのために OpenJIT レジスタ割付機能を利用する．出力されたネイティブコードは，JavaVM によって呼び出され実行されるが，その際に OpenJIT ランタイムモジュールを補助的に呼び出す．

ただし，OpenJIT SPARC プロセッサモジュール，および OpenJIT ランタイムモジュールは，今回の開発とは別途開発が行われるため，試験の対象外である．

(1) OpenJIT ネイティブコード変換機能

OpenJIT ネイティブコード変換機能 (図 1.11) は，OpenJIT バックエンド全体の制御を行う．

Java のバイトコードからネイティブコードを出力するための抽象フレームワークである．実際は，このクラスを具体的なプロセッサに応じたクラスで特化することによって，実際のコード出力機能を定義する．それぞれのバイトコードと，プログラムの各種グラフ，およびフロントエンドシステムのプログラム解析・変換の結果を用いて，ネイティブコードへの変換を行なう．

また，OpenJIT ネイティブコード変換機能は以下の小機能から構成される．

- ネイティブコード変換

  バイトコードを入力とし，SPARC ネイティブコードを出力する
  OpenJIT 中間コード変換機能，OpenJIT RTL 変換機能，OpenJIT Peephole 最適化機能を制御する．

- メソッド情報

  メソッドに関する JDK の内部構造を Java のデータ構造に変換する．

- バイトコードアクセス

  JDK の内部構造であるバイトコードを読み取る．

- 生成コードメモリ管理

  生成するネイティブコードの領域の確保，その領域へのコードの書き込み，開放を司る．

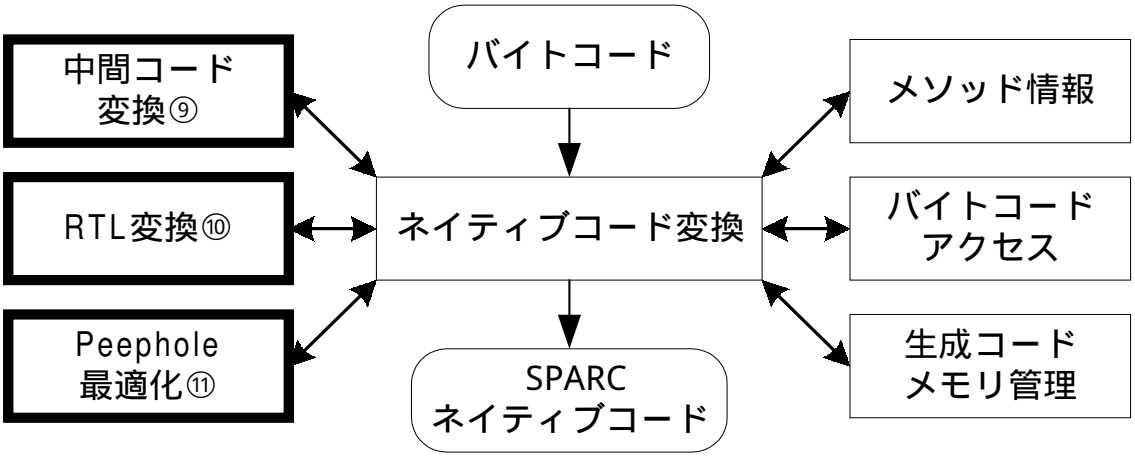

図 1.11: OpenJIT ネイティブコード変換機能

(2) OpenJIT 中間コード変換機能

OpenJIT 中間コード変換機能 (図 1.12) は，バイトコードから中間言語への変換を行う．

フロントエンドシステムの出力であるバイトコードを入力とする．バイトコードの各命令をグループに分別し，より単純な中間言語に変換を行なう．メソッド呼び出しのバイトコード命令について，メソッドの引数の数や型の解析を行い，中間言語列に展開する．この中間言語のオペランドはスタックで与えられる．また，Java 特有な命令列パターンを検出し，単純な中間言語に置き換える最適化を含めて行う．

また，OpenJIT 中間コード変換機能は以下の小機能から構成される．

- 中間言語変換

  バイトコードを入力として，バイトコードの各命令を中間言語に変換して出力する．

- メソッド引数展開

  メソッド呼び出しの引数を中間言語に展開する．

- 命令パターンマッチング

  特定のバイトコードパターンに対し最適化した中間言語に変換する．

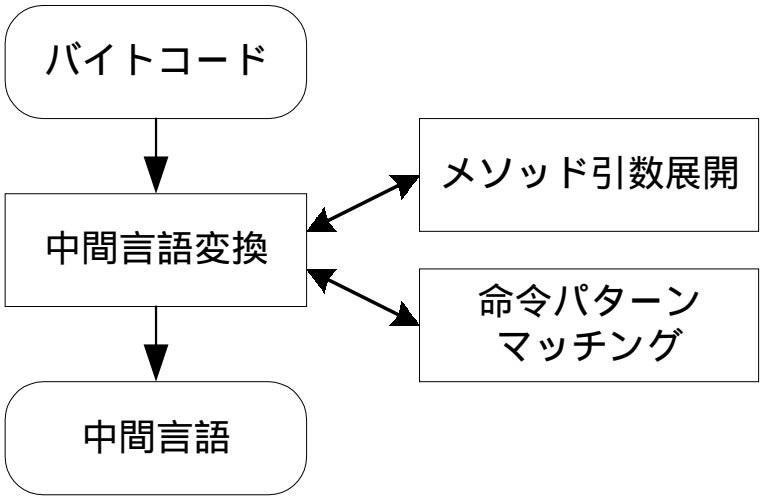

図 1.12: OpenJIT 中間コード変換機能

(3) OpenJIT RTL 変換機能

OpenJIT RTL 変換機能 (図 1.13) は，中間言語から RTL への変換を行う．

OpenJIT 中間コード変換機能の生成結果を入力とし，スタックオペランドを使った中間言語からレジスタを使った中間言語，RTL(Register Transfer Language) に変換を行なう．中間言語列を基本ブロックに分割し，実行の制御の流れを解析することにより，スタックマシンコードからオペランドレジスタコードへの変換を行なう．OpenJIT では，無限資源のレジスタがあるとみなして RTL への変換を行なう．また，オペランドのうち型が未解決のものの型を決定する．

また，OpenJIT RTL 変換機能は以下の小機能から構成される．

- 基本ブロック分割

  中間言語を入力とし，基本ブロックに分割して基本ブロック情報を出力するまた，中間言語列を入力とし，各中間言語を RTL に変換して出力する．

- コントロールフロー解析

  基本ブロック間の処理の流れを解析する．

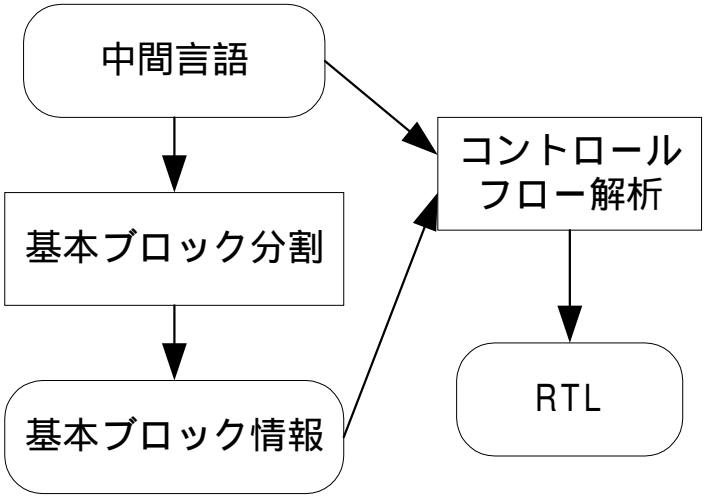

図 1.13: OpenJIT RTL 変換機能

(4) OpenJIT Peephole 最適化機能

OpenJIT Peephole 最適化機能 (図 1.14) は，RTL を最適化し，無駄な命令を取り除いて最適化を行う．

OpenJIT RTL 変換機能の生成した RTL を入力として，RTL に対して Peephole 最適化を施す．Peephole 最適化としては，通常行われる redundant load/store elimination を行う．また，Java 固有の Peephole 最適化も行なわれる．Java に特有な配列のインデックスの境界チェックを取り除く最適化も行なう．このモジュールは冗長な命令を取り除いて最適化された RTL を出力する．

また，OpenJIT Peephole 最適化機能は以下の小機能から構成される．

- データフロー解析

  RTL 列のデータの流れを解析する．

- 各種 Peephole 最適化

  基本ブロック情報と RTL を入力として，データフロー解析結果を元に最適化した RTL を出力する．

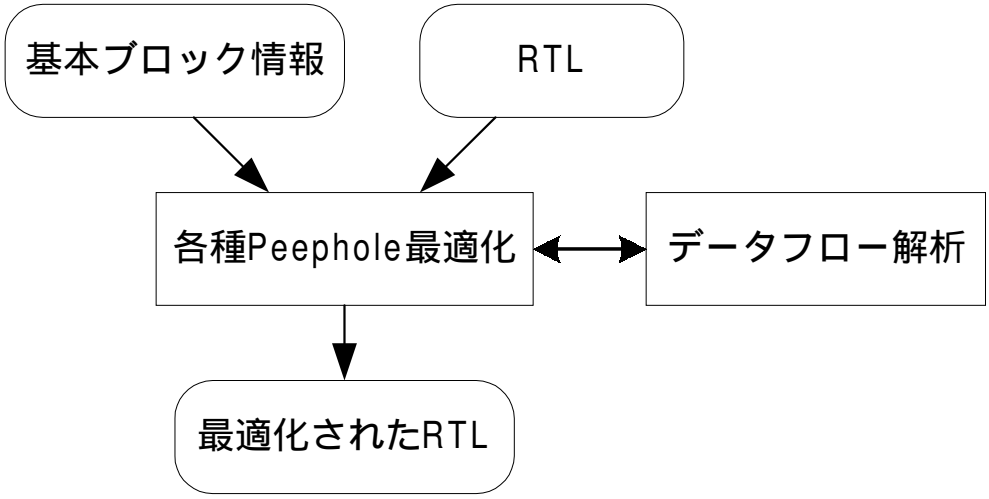

図 1.14: OpenJIT Peephole 最適化機能

(5) OpenJIT レジスタ割付機能

OpenJIT レジスタ割付機能 (図 1.15) は，仮想レジスタから物理レジスタへの割付を行う．

ネイティブコード生成の際，実際のプロセッサレジスタへの割付を行なう．レジスタ割付アルゴリズムを適用し，実際のプロセッサレジスタに対して割付を行なう．物理レジスタの数が足りない場合は，一時レジスタを割り付け，スピル / フィルコードを生成する．

また，OpenJIT レジスタ割付機能は以下の小機能から構成される．

- 仮想レジスタ管理

  中間言語で使用する仮想レジスタの管理をする．

- 物理レジスタ管理

  生成するネイティブコードが使用する物理レジスタを管理する．

- レジスタ割付

  与えられた仮想レジスタ番号に対して，物理レジスタを割り付け，物理レジスタ番号を返す．

  割り付けできないときはスピル / フィルコードを生成する．

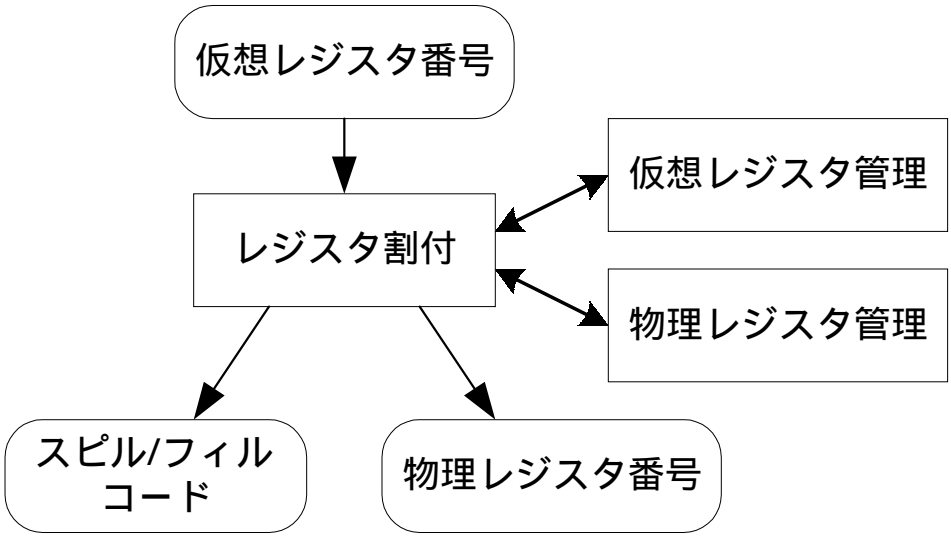

図 1.15: OpenJIT レジスタ割付機能

第 2 章

試験方針

2.1 試験方針

各プログラムが構造仕様書で記述した仕様を満たしていることを確認するため，そのそれぞれのプログラムについて単体試験，および，結合試験を行う．そのため，第 4 章で列挙する試験項目を設定し，試験を行う．

試験項目設定の方針としては，各試験で試験されるプログラムを明らかにするとともに，なるべく各プログラムを独立して調べられるような試験を行った上で，多くのプログラムを結合して試験するものとする．

2.2 用語

試験に関する用語を以下のように定義する．

スタブ

スタブとは切り株のことであるがここでは次の意味で用いる．プログラム A を利用するプログラム B を試験するときに，プログラム A と同一のインタフェースを持つが，他に影響を及ぼさないように，中身の無いサブプログラム A' を用いる場合がある．この A' をスタブと言い，プログラム A の完成度に影響を受けずにプログラム B を試験するのに利用される．

テストドライバ

スタブとは逆に，プログラム A に利用されるプログラム B を試験するときに，プログラム A の代りに，プログラム B を利用して試験を行うプログラム A' を用いる場合がある．この A' をテストドライバと言い，プログラム A の完成度に影響を受けずにプログラム B を試験するために利用される．

プログラムを単体で試験することを単体試験と呼ぶ．また，結合試験には以下のものがある．

ボトムアップ結合試験 テストドライバを準備して，下位プログラムから順にプログラムを結合させたものについて試験することで，プログラム間の結合試験を行う．

トップダウン結合試験 下位プログラムのスタブを準備し，上位プログラムから順にプログラムを結合させたものについて試験することで，プログラム間の結合試験を行う．

ビッグバン結合試験 プログラム間の結合が強い場合など，すべてのプログラムをすべて結合させて試験することで，サブプログラム間の結合試験を行う．

試験方針としては，ボトムアップ結合試験，トップダウン結合試験，ビッグバン結合試験のいずれかを用いて結合試験を行うことで，プログラム間の結合が正しく行われていることを確認する．そのため，第 4 章で列挙する試験項目について試験を行うものとする．

第 3 章

試験環境

3.1 OpenJIT フロントエンドシステム

本システムを構成するサブシステムの試験では，以下のような構成のハードウェア・ソフトウェアを用いた．

() 内は本システムの動作に必要な条件である．

(1) ハードウェア構成

- プロセッサ: Sun Ultra60 <UltraSPARC-II 300MHz 2 基搭載 >
  (SPARC version 8 以降のプロセッサを搭載した Sun Workstation)
- メモリ: 256MB
  (256MB 以上)
- ハードディスク容量: 4GB
  (4GB 以上)

(2) ソフトウェア構成

- オペレーティングシステム: Sun Solaris 2.6 (Sun Solaris 2.5.1 以降)
- Java 実行環境: Sun JDK1.1.6 (Sun JDK 1.1.4 以降)

(3)　他システムとの関連（インタフェース）

Java 仮想マシン (JavaVM) には外部の JIT コンパイラを組み込むインターフェース・ API が準備されている．具体的には JIT は各クラスに対するメソッドディスパッチ部位を書き換え，直接メソッド本体ではなく，JIT コンパイラが起動されるようにしておく．メソッド呼び出しによって起動された JIT コンパイラは，メソッドを構成するバイトコードをその場でコンパイルし，ネイティブコードを得て，ヒープ領域に格納する．メソッドディスパッチ部位をさらに書き換え，以後の起動では直接ネイティブコードが起動されるようにする．

このインターフェース・ API は，Sun Microsystems により定められた Java JIT Interface の仕様に基づいている．

3.2 OpenJIT バックエンドシステム

本システムを構成するサブシステムの試験では，以下のような構成のハードウェア・ソフトウェアを用いた．

() 内は本システムの動作に必要な条件である．

(1) ハードウェア構成

- **プロセッサ**: Sun Ultra60 <UltraSPARC-II 300MHz 2 基搭載 >
  (SPARC version 8 以降のプロセッサを搭載した Sun Workstation)
- **メモリ**: 256MB
  (256MB 以上)
- **ハードディスク容量**: 4GB
  (4GB 以上)

(2) ソフトウェア構成

- **オペレーティングシステム**: Sun Solaris 2.6 (Sun Solaris 2.5.1 以降)
- Java 実行環境: Sun JDK1.1.6 (Sun JDK 1.1.4 以降)

(3) 他システムとの関連 (インタフェース)

Java 仮想マシン (JavaVM) には外部の JIT コンパイラを組み込むインターフェース・ API が準備されている．具体的には JIT は各クラスに対するメソッドディスパッチ部位を書き換え，直接メソッド本体ではなく， JIT コンパイラが起動されるようにしておく．メソッド呼び出しによって起動された JIT コンパイラは，メソッドを構成するバイトコードをその場でコンパイルし，ネイティブコードを得て，ヒープ領域に格納する．メソッドディスパッチ部位をさらに書き換え，以後の起動では直接ネイティブコードが起動されるようにする．

このインターフェース・ API は， Sun Microsystems により定められた Java JIT Interface の仕様に基づいている．

第 4 章

試験項目

4.1 OpenJIT フロントエンドシステム

4.1.1 OpenJIT コンパイラ基盤機能

試験項目	OpenJIT 初期化部を構成するプログラム	OpenJIT コンパイラフロントエンド制御部を構成するプログラム	OpenJIT JNI API 登録部を構成するプログラム	説明
OpenJIT 初期化部動作試験 (1)	○			OpenJIT 初期化部の動作を確認する．
OpenJIT 初期化部動作試験 (2)	○			OpenJIT を使用しない設定の時に，OpenJIT の初期化が行われないことを確認する．
OpenJIT コンパイラフロントエンド制御部動作試験	○	○		OpenJIT コンパイラフロントエンド制御部の動作を確認する．
OpenJIT JNI API 登録部動作試験	○		○	OpenJIT JNI API 登録部の動作を確認する．

(次のページへ続く)

(前のページからの続き)

試験項目	OpenJIT 初期化部を構成するプログラム	OpenJIT コンパイラフロントエンド制御部を構成するプログラム	OpenJIT JNI API 登録部を構成するプログラム	説明
OpenJIT コンパイラ起動試験 (1)	○		○	OpenJIT を有効にした場合に JDK のインタフェースによって OpenJIT が起動できることを確認する．
OpenJIT コンパイラ起動試験 (2)	○		○	OpenJIT を無効にした場合に JDK のインタフェースによって OpenJIT が起動されないことを確認する．
OpenJIT コンパイラ基盤機能動作試験	○	○	○	OpenJIT コンパイラ基盤機能が他の各機能を呼び出せることを確認する．

以下に各小機能を構成するプログラムを挙げる．

- OpenJIT 初期化部を構成するプログラム
  - java_lang_Compiler_start()
- OpenJIT コンパイラフロントエンド制御部を構成するプログラム
  - OpenJIT.Compile.compile()
- OpenJIT JNI API 登録部を構成するプログラム
  - java_lang_Compiler_start()

4.1.2 OpenJIT バイトコードディスコンパイラ機能

試験項目	バイトコード解析部を構成するプログラム	コントロールグラフ出力部を構成するプログラム	AST 出力部を構成するプログラム	説明
バイトコード解析部動作試験	○			バイトコード解析部の動作を確認する．
コントロールグラフ出力部動作試験 1	○	○		コントロールグラフ出力部の動作を確認する．
コントロールグラフ出力部動作試験 2	○	○		ベーシックブロック単位の解析の済んだコントロールグラフを出力する動作を確認する．
コントロールグラフ出力部動作試験 3	○	○		式単位の解析の済んだコントロールグラフを出力する動作を確認する．
AST 出力部動作試験 1	○	○	○	ドミネータツリーを構築する動作を確認する．
AST 出力部動作試験 2	○	○	○	制御構造を回復する動作を確認する．
OpenJIT バイトコードディスコンパイラ機能動作試験	○	○	○	OpenJIT バイトコードディスコンパイラ機能の動作を確認する．

以下に各小機能を構成するプログラムを挙げる．

- バイトコード解析部を構成するプログラム
  - BytecodeParser.BytecodeParser()
- コントロールグラフ出力部を構成するプログラム
  - BytecodeParser.parseFrom()
  - ControlFlowGraph.createCFG()
  - ControlFlowGraph.newBasicBlock()
  - BasicBlockAnalyzer.newBasicBlock()
  - ExpressionAnalyzer.newBasicBlock()
  - ExpressionAnalyzer.ExpressionAnalyzer()
  - DominatorTree.DominatorTree()
- AST 出力部を構成するプログラム
  - Discompiler.discompile()
  - BasicBlockAnalyzer.completeValue()
  - ExpressionAnalyzer.recoverExpressions()
  - ASTFactory.booleanExpression()
  - ASTFactory.byteExpression()
  - ASTFactory.charExpression()
  - ASTFactory.shortExpression()
  - ASTFactory.intExpression()
  - ASTFactory.longExpression()
  - ASTFactory.floatExpression()
  - ASTFactory.doubleExpression()
  - ASTFactory.stringExpression()

- ASTFactory.nullExpression()
- ASTFactory.arrayAccessExpression()
- ASTFactory.bitNotExpression()
- ASTFactory.convertExpression()
- ASTFactory.exprExpression()
- ASTFactory.fieldExpression()
- ASTFactory.lengthExpression()
- ASTFactory.negativeExpression()
- ASTFactory.notExpression()
- ASTFactory.positiveExpression()
- ASTFactory.postDecExpression()
- ASTFactory.postIncExpression()
- ASTFactory.preDecExpression()
- ASTFactory.preIncExpression()
- ASTFactory.castExpression()
- ASTFactory.commaExpression()
- ASTFactory.instanceOfExpression()
- ASTFactory.addExpression()
- ASTFactory.divideExpression()
- ASTFactory.multiplyExpression()
- ASTFactory.subtractExpression()
- ASTFactory.remainderExpression()
- ASTFactory.andExpression()
- ASTFactory.orExpression()
- ASTFactory.bitAndExpression()
- ASTFactory.bitOrExpression()

- ASTFactory.bitXorExpression()
- ASTFactory.equalExpression()
- ASTFactory.notEqualExpression()
- ASTFactory.shiftLeftExpression()
- ASTFactory.shiftRightExpression()
- ASTFactory.unsignedShiftRightExpression()
- ASTFactory.greaterExpression()
- ASTFactory.greaterOrEqualExpression()
- ASTFactory.lessExpression()
- ASTFactory.lessOrEqualExpression()
- ASTFactory.assignExpression()
- ASTFactory.assignAddExpression()
- ASTFactory.assignBitAndExpression()
- ASTFactory.assignBitOrExpression()
- ASTFactory.assignBitXorExpression()
- ASTFactory.assignDivideExpression()
- ASTFactory.assignMultiplyExpression()
- ASTFactory.assignRemainderExpression()
- ASTFactory.assignShiftLeftExpression()
- ASTFactory.assignShiftRightExpression()
- ASTFactory.assignSubtractExpression()
- ASTFactory.assignUnsignedShiftRightExpression()
- ASTFactory.conditionalExpression()
- ASTFactory.arrayExpression()
- ASTFactory.methodExpression()
- ASTFactory.newArrayExpression()

- ASTFactory.newInstanceExpression()
- ASTFactory.identifierExpression()
- ASTFactory.superExpression()
- ASTFactory.thisExpression()
- ASTFactory.typeExpression()
- ASTFactory.breakStatement()
- ASTFactory.caseStatement()
- ASTFactory.catchStatement()
- ASTFactory.compoundStatement()
- ASTFactory.continueStatement()
- ASTFactory.declarationStatement()
- ASTFactory.doStatement()
- ASTFactory.expressionStatement()
- ASTFactory.finallyStatement()
- ASTFactory.forStatement()
- ASTFactory.ifStatement()
- ASTFactory.returnStatement()
- ASTFactory.switchStatement()
- ASTFactory.synchronizedStatement()
- ASTFactory.throwStatement()
- ASTFactory.tryStatement()
- ASTFactory.varDeclarationStatement()
- ASTFactory.whileStatement()

4.1.3 OpenJIT クラスファイルアノテーション解析機能

試験項目	アノテーション解析部を構成するプログラム	アノテーション登録部を構成するプログラム	メタクラス制御部を構成するプログラム	説明
アノテーション解析部動作試験 (1)	○			A1 型のアノテーション情報を解析する動作を確認する．
アノテーション解析部動作試験 (2)	○			A2 型のアノテーション情報を解析する動作を確認する．
アノテーション登録部動作試験 (1)		○		A1 型のアノテーション情報を登録する動作を確認する．
アノテーション登録部動作試験 (2)		○		不正なアノテーション情報が登録されないことを確認する．
メタクラス制御部動作試験			○	A1 型のメタクラスを起動するために必要な情報が与えられることを確認する．
OpenJIT クラスファイルアノテーション解析機能動作試験	○	○	○	OpenJIT クラスファイルアノテーション解析機能の動作を確認する．

以下に各小機能を構成するプログラムを挙げる．

- アノテーション解析部を構成するプログラム
  - AnnotationAnalyzer.readAnnotation()
- アノテーション登録部を構成するプログラム
  - AnnotationAnalyzer.registerAnnotation()
- メタクラス制御部を構成するプログラム
  - AnnotationAnalyzer.addMetaobject()
  - BinaryAssignExpression.check()
  - BreakStatement.check()
  - CaseStatement.check()
  - CatchStatement.check()
  - CommaExpression.check()
  - CompoundStatement.check()
  - ConditionalExpression.check()
  - ContinueStatement.check()
  - DeclarationStatement.check()
  - DoStatement.check()
  - Expression.check()
  - ExpressionStatement.check()
  - FinallyStatement.check()
  - ForStatement.check()
  - IfStatement.check()
  - IncDecExpression.check()
  - MethodExpression.check()

- NewInstanceExpression.check()
- ReturnStatement.check()
- Statement.check()
- SwitchStatement.check()
- SynchronizedStatement.check()
- ThrowStatement.check()
- TryStatement.check()
- WhileStatement.check()
- ArrayAccessExpression.checkValue()
- ArrayExpression.checkValue()
- AssignExpression.checkValue()
- AssignOpExpression.checkValue()
- BinaryExpression.checkValue()
- BinaryLogicalExpression.checkValue()
- CastExpression.checkValue()
- ConditionalExpression.checkValue()
- ConvertExpression.checkValue()
- Expression.checkValue()
- FieldExpression.checkValue()
- IdentifierExpression.checkValue()
- IncDecExpression.checkValue()
- InstanceOfExpression.checkValue()
- LengthExpression.checkValue()
- MethodExpression.checkValue()
- NewArrayExpression.checkValue()
- NewInstanceExpression.checkValue()

- SuperExpression.checkValue()
- ThisExpression.checkValue()
- TypeExpression.checkValue()
- UnaryExpression.checkValue()
- ArrayAccessExpression.checkAssignOp()
- Expression.checkAssignOp()
- FieldExpression.checkAssignOp()
- IdentifierExpression.checkAssignOp()
- ArrayAccessExpression.checkLHS()
- Expression.checkLHS()
- FieldExpression.checkLHS()
- IdentifierExpression.checkLHS()
- ArrayAccessExpression.checkInitializer()
- Expression.checkInitializer()

4.1.4 OpenJIT 最適化機能

試験項目	最適化制御部を構成するプログラム	バイトコード出力部を構成するプログラム出力部	説明
最適化制御部動作試験	○		最適化制御部の動作を確認する．
バイトコード出力部動作試験		○	バイトコード出力部の動作を確認する．
OpenJIT 最適化機能動作試験	○	○	OpenJIT 最適化機能の動作を確認する．

以下に各小機能を構成するプログラムを挙げる．

- 最適化制御部を構成するプログラム
  - Assembler.optimize()
  - Optimizer.optimize()
- バイトコード出力部を構成するプログラム
  - Optimizer.generateBytecode()
  - AddExpression.code()
  - ArrayAccessExpression.codeValue()
  - ArrayExpression.codeValue()
  - AssignExpression.code()
  - AssignExpression.codeValue()
  - AssignOpExpression.code()
  - AssignOpExpression.codeValue()
  - BinaryBitExpression.codeValue()
  - BinaryExpression.codeValue()
  - BitNotExpression.codeValue()
  - BooleanExpression.codeValue()
  - BreakStatement.code()
  - CatchStatement.code()
  - CommaExpression.code()
  - CompoundStatement.code()
  - ConditionalExpression.codeValue()
  - ContinueStatement.code()
  - ConvertExpression.codeValue()
  - DeclarationStatement.code()

- DoStatement.code()
- DoubleExpression.codeValue()
- Expression.code()
- Expression.codeValue()
- ExpressionStatement.code()
- FieldExpression.codeValue()
- FinallyStatement.code()
- FloatExpression.codeValue()
- ForStatement.code()
- IdentifierExpression.codeValue()
- IfStatement.code()
- InlineMethodExpression.code()
- InlineMethodExpression.codeValue()
- InlineNewInstanceExpression.code()
- InlineNewInstanceExpression.codeValue()
- InlineReturnStatement.code()
- InstanceOfExpresion.code()
- InstanceOfExpression.codeValue()
- IntegerExpression.codeValue()
- LengthExpression.codeValue()
- LongExpression.codeValue()
- MethodExpression.codeValue()
- NegativeExpression.codeValue()
- NewArrayExpression.codeValue()
- NewInstanceExpression.code()
- NewInstanceExpression.codeValue()

- NullExpression.codeValue()
- PostDecExpression.code()
- PostDecExpression.codeValue()
- PostIncExpression.code()
- PostIncExpression.codeValue()
- PreDecExpression.code()
- PreDecExpression.codeValue()
- PreIncExpression.code()
- PreIncExpression.codeValue()
- ReturnStatement.code()
- Statement.code()
- StringExpression.codeValue()
- SuperExpression.codeValue()
- SwitchStatement.code()
- SynchronizedStatement.code()
- ThisExpression.codeValue()
- ThrowStatement.code()
- TryStatement.code()
- VarDeclarationStatement()
- WhileStatement.code()

4.1.5 OpenJIT フローグラフ構築機能

試験項目	AST 等入力部を構成するプログラム	データフローグラフ構築部を構成するプログラム	コントロール依存グラフ構築部を構成するプログラム	クラス階層解析部を構成するプログラム	説明
AST 等入力部動作試験 (1)	○				データの入力動作を確認する．
AST 等入力部動作試験 (2)	○				各グラフ構築部が呼び出されることを確認する．
データフローグラフ構築部動作試験 (1)	○	○			データフローグラフ構築が呼び出されることを確認する．
データフローグラフ構築部動作試験 (2)	○	○			データフローグラフ構築の動作を確認する．
コントロール依存グラフ構築部動作試験 (1)	○		○		コントロール依存グラフ構築が呼び出されることを確認する．
コントロール依存グラフ構築部動作試験 (2)	○		○		コントロール依存グラフ構築の動作を確認する．
クラス階層解析部動作試験 (1)	○			○	クラス階層解析のクラス情報構築が呼び出されることを確認する．
クラス階層解析部動作試験 (2)	○			○	クラス階層解析結果のクラス階層グラフ構築の動作を確認する．
OpenJIT フローグラフ構築機能動作試験	○	○	○	○	OpenJIT フローグラフ構築機能の動作を確認する．

以下に各小機能を構成するプログラムを挙げる．

- AST 等入力部を構成するプログラム
  - FlowGraph.setAST()
  - FlowGraph.setCFG()
  - FlowGraph.setCHInfo()
- データフローグラフ構築部を構成するプログラム
  - FlowGraph.constructDFG()
  - DataFlowGraph.constructGraph()
  - DataFlowGraph.seekStatement()
  - DataFlowGraph.seekExpression()
  - DataFlowGraph.initDataFlow()
- コントロール依存グラフ構築部を構成するプログラム
  - FlowGraph.constructCFG()
  - ControlDependencyGraph.constructGraph()
  - ControlDependencyGraph.seekStatement()
- クラス階層解析部を構成するプログラム
  - FlowGraph.analysisCH()
  - ClassHierarchyGraph.analysis()

4.1.6 OpenJIT フローグラフ解析機能

試験項目	データフロー関数登録部を構成するプログラム	フローグラフ解析部を構成するプログラム	浮動点検出部を構成するプログラム	クラス階層解析部を構成するプログラム	説明
データフロー関数登録部動作試験	○				データフロー関数登録の動作を確認する．
フローグラフ解析部動作試験 (1)		○			各フローグラフ解析を呼び出せることを確認する．
フローグラフ解析部動作試験 (2)		○			到達定義の解析の動作を確認する．
フローグラフ解析部動作試験 (3)		○			利用可能な式の解析の動作を確認する．
フローグラフ解析部動作試験 (4)		○			生きている式の解析の動作を確認する．
不動点検出部動作試験			○		浮動点検出の動作を確認する．
クラス階層解析部動作試験				○	クラス階層解析の動作を確認する．
OpenJIT フローグラフ解析機能動作試験	○	○	○	○	OpenJIT フローグラフ解析機能の動作を確認する．

以下に各小機能を構成するプログラムを挙げる．

- データフロー関数登録部を構成するプログラム
  - DFFunctionRegister.register()
- フローグラフ解析部を構成するプログラム
  - FlowGraphAnalysis.analysis()
  - ReachingAnalyzer.analysis()
  - AvailableAnalyzer.analysis()
  - LivenessAnalyzer.analysis()
- 浮動点検出部を構成するプログラム
  - FixedPointDetector.analysis()
- クラス階層解析部を構成するプログラム
  - ClassHierarchyAnalyzer.analysis()

4.1.7 OpenJIT プログラム変換機能

試験項目	AST 変換ルール登録部を構成するプログラム	AST パターンマッチ部を構成するプログラム	AST 変換部を構成するプログラム	説明
AST 変換ルール登録部動作試験	○			AST 変換ルールが登録できることを確認する。
AST パターンマッチ部動作試験 (1)	○	○		変換ルールに登録されたパターンにマッチする場合の動作を確認する．
AST パターンマッチ部動作試験 (2)	○	○		変換ルールに登録されたパターンにマッチしない場合の動作を確認する．
AST 変換部動作試験 (1)	○		○	変換ルールに登録されたパターンの AST が変換されることを確認する．
AST 変換部動作試験 (2)	○		○	変換ルールに登録されていないパターンの AST が変換されないことを確認する．
OpenJIT プログラム変換機能動作試験	○	○	○	OpenJIT プログラム変換機能の動作を確認する．

以下に各小機能を構成するプログラムを挙げる．

- AST 変換ルール登録部を構成するプログラム
  - ASTTransformer.registerRule()
- AST パターンマッチ部を構成するプログラム
  - ASTTransformer.match()
- AST 変換部を構成するプログラム
  - ASTTransformer.transform()

4.2 OpenJIT バックエンドシステム

4.2.1 OpenJIT ネイティブコード変換機能

試験項目	ネイティブコード変換	メソッド情報	バイトコードアクセス	生成コードメモリ管理	説明
ネイティブコード変換試験	○	○	○	○	与えられたバイトコードに対し，ネイティブコードが生成されることを確認する．
ネイティブコード変換機能動作試験 (1)	○				OpenJIT 中間コード変換機能を呼び出せることを確認する．
ネイティブコード変換機能動作試験 (2)	○				OpenJIT RTL 変換機能を呼び出せることを確認
ネイティブコード変換機能動作試験 (3)	○				OpenJIT Peephole 最適化 RTL 変換機能を呼び出せることを確認
メソッド情報取得試験 (1)		○			メソッドのクラス名，メソッド名，シグナチャが Java から読めることを確認
メソッド情報取得試験 (2)		○			メソッドのアクセス情報が Java から読めることを確認
メソッド情報取得試験 (3)		○			メソッドの nlocals, maxstack, args_size 情報が Java から読めることを確認
バイトコードアクセス試験 (1)			○		メソッドのバイトコードのサイズが Java から読めることを確認

(次のページへ続く)

(前のページからの続き)

試験項目	ネイティブコード変換				
		メソッド情報			
			バイトコードアクセス		
				生成コードメモリ管理	
					説明
バイトコードアクセス試験 (2)			○		メソッドのバイトコードのサイズが Java から読めることを確認
生成コードメモリ管理試験 (1)				○	メモリ領域が確保できることを確認
生成コードメモリ管理試験 (2)				○	確保したメモリ領域に命令が書き込めることを確認

以下に各小機能を構成するプログラムを挙げる．

- ネイティブコード変換を構成するプログラム
  - OpenJIT.Compile.compile()
- メソッド情報を構成するプログラム
  - OpenJIT_compile()
- バイトコードアクセスを構成するプログラム
  - OpenJIT.Compile.pc2uchar()
  - OpenJIT.Compile.pc2signedchar()
  - OpenJIT.Compile.pc2signedshort()
  - OpenJIT.Compile.pc2signedlong()
  - OpenJIT.Compile.pc2ushort()
- 生成コードメモリ管理を構成するプログラム
  - OpenJIT.Compile.NativeCodeAlloc()
  - OpenJIT.Compile.NativeCodeReAlloc()
  - OpenJIT.Compile.setNativeCode()
  - OpenJIT.Compile.getNativeCode()

4.2.2 OpenJIT 中間コード変換機能

試験項目	中間言語変換	メソッド引数展開	命令パターンマッチング	説明
中間言語変換試験 (1)	○			バイトコードがバックエンド中間コードに変換できることをを確認
中間言語変換試験 (2)	○			中間言語変換機能がメソッド引数展開を呼び出せることを確認
中間言語変換試験 (3)	○			中間言語変換機能が命令パターンマッチング (optim_neg) を呼び出せることを確認
中間言語変換試験 (4)	○			中間言語変換機能が命令パターンマッチング (optim_lcmp) を呼び出せることを確認
中間言語変換試験 (4)	○			中間言語変換機能が命令パターンマッチング (optim_fcmp) を呼び出せることを確認
メソッド引数展開試験 (1)		○		メソッド呼び出しの引数がバックエンド中間コードに展開できることを確認 (引数の個数)
メソッド引数展開試験 (2)		○		メソッド呼び出しの引数がバックエンド中間コードに展開できることを確認 (引数の型)
メソッド引数展開試験 (3)		○		メソッド呼び出しがバックエンド中間コードに展開できることを確認 (返り値)
命令パターンマッチング試験 (1)			○	論理値の否定を行うパターンが，最適化されたバックエンド中間コードに変換されることを確認
命令パターンマッチング試験 (2)			○	long の比較・分岐パターンが，最適化されたバックエンド中間コードに変換されることを確認

(次のページへ続く)

(前のページからの続き)

試験項目	中間言語変換			
		メソッド引数展開		
			命令パターンマッチング	
				説明
命令パターンマッチング試験 (2)			○	float の比較・分岐パターンが，最適化されたバックエンド中間コードに変換されることを確認
命令パターンマッチング試験 (2)			○	double の比較・分岐パターンが，最適化されたバックエンド中間コードに変換されることを確認

以下に各小機能を構成するプログラムを挙げる．

- 中間言語変換を構成するプログラム
  - OpenJIT.ParseBytecode.parseBytecode()
- メソッド引数展開を構成するプログラム
  - OpenJIT.ParseBytecode.callMethod()
- 命令パターンマッチングを構成するプログラム
  - OpenJIT.ParseBytecode.optim_neg()
  - OpenJIT.ParseBytecode.optim_lcmp()
  - OpenJIT.ParseBytecode.optime_fcmp()

4.2.3 OpenJIT RTL 変換機能

試験項目	基本ブロック分割		
		コントロールフロー解析	
			説明
基本ブロック分割機能試験 (1)	○	○	バイトコードから基本ブロック情報が得られることを確認 (条件分岐)
基本ブロック分割機能試験 (2)	○		バイトコードから基本ブロック情報が得られることを確認 (tableswitch)
基本ブロック分割機能試験 (3)	○		バイトコードから基本ブロック情報が得られることを確認 (lookupswitch)
基本ブロック分割機能試験 (4)	○		バイトコードから基本ブロック情報が得られることを確認 (例外処理)
基本ブロック分割機能試験 (5)	○		バイトコードから基本ブロック情報が得られることを確認 (jsr,ret)
RTL 変換機能試験		○	バックエンド中間コードから RTL への変換確認
コントロールフロー解析試験		○	コントロールフロー解析ができていることを確認
オペランドの型の決定試験		○	バックエンド中間コードのオペランドの型が未定義なものから，RTL に変換できることを確認

以下に各小機能を構成するプログラムを挙げる．

- 基本ブロック分割を構成するプログラム
  - OpenJIT.ConvertRTL.extractBB()
- コントロールフロー解析を構成するプログラム
  - OpenJIT.ConvertRTL.convertRTL()

4.2.4 OpenJIT Peephole 最適化機能

試験項目	データフロー解析	各種 Peephole 最適化	説明
データフロー解析試験 (1)	○	○	データ (定数) の流れが正しく解析できていることを確認
データフロー解析試験 (2)	○	○	データ (ローカル変数) の流れが正しく解析できていることを確認
Peephole 最適化試験 (1)	○	○	結果をローカル変数へ代入する処理の最適化確認
Peephole 最適化試験 (2)	○	○	結果をパラメタ変数へ代入する処理の最適化確認
Constant Folding 最適化試験 (1)	○	○	右論理シフトの最適化ができていることを確認
Constant Folding 最適化試験 (2)	○	○	0 との論理積の最適化ができていることを確認
Constant Folding 最適化試験 (3)	○	○	-1(all 1) との論理積の最適化ができていることを確認
ハンドル参照除去最適化試験		○	ハンドル間接参照の load 命令の削除の最適化確認

以下に各小機能を構成するプログラムを挙げる．

- データフロー解析を構成するプログラム
  - OpenJIT.OptimizeRTL.optimizeRTL()
- 各種 Peephole 最適化を構成するプログラム
  - OpenJIT.OptimizeRTL.eliminateIL()
  - OpenJIT.OptimizeRTL.moveUpSV()
  - OpenJIT.OptimizeRTL.moveUpSP()

4.2.5 OpenJIT レジスタ割付機能

試験項目	仮想レジスタ管理	物理レジスタ管理	レジスタ割り付け	説明
整数レジスタ割付試験	○	○	○	整数レジスタ割り付け機能を確認
浮動小数レジスタ割付試験	○	○	○	浮動小数レジスタ割り付け機能を確認
整数物理レジスタ割付試験	○	○	○	整数物理レジスタの割付確認
浮動小数物理レジスタ割付試験	○	○	○	浮動小数物理レジスタの割付確認
スピルコード生成試験	○	○	○	スピルコードが生成されることを確認
フィルコード生成試験	○	○	○	フィルコードが生成されることを確認

以下に各小機能を構成するプログラムを挙げる．

- 仮想レジスタ管理を構成するプログラム
  - OpenJIT.RegAlloc$VirReg.assign()
  - OpenJIT.RegAlloc$VirReg.fill()
  - OpenJIT.RegAlloc$VirReg.spill()
  - OpenJIT.RegAlloc$VirReg.nouse()
  - OpenJIT.RegAlloc$VirReg.invalidate()
- 物理レジスタ管理を構成するプログラム
  - OpenJIT.RegAlloc$PhyRegs.fixReg()
  - OpenJIT.RegAlloc$PhyRegs.getFixReg()
  - OpenJIT.RegAlloc$PhyRegs.unlock()
  - OpenJIT.RegAlloc$PhyRegs.save()
  - OpenJIT.RegAlloc$PhyRegs.invalidateI()
  - OpenJIT.RegAlloc$PhyRegs.invalidateF()
  - OpenJIT.RegAlloc$PhyRegs.read()
  - OpenJIT.RegAlloc$PhyRegs.write()
  - OpenJIT.RegAlloc$PhyRegs.searchVector()
  - OpenJIT.RegAlloc$PhyRegs.getTmpReg()
  - OpenJIT.RegAlloc$PhyRegs.freeTmpReg()
  - OpenJIT.RegAlloc$PhyRegs.isTmpReg()
- レジスタ割り付けを構成するプログラム
  - OpenJIT.RegAlloc.regAlloc()

第 5 章

試験方法

5.1 OpenJIT フロントエンドシステム

5.1.1 OpenJIT コンパイラ基盤機能

試験項目: OpenJIT 初期化部動作試験 (1)	(7) 合否判定

(1) 試験目的，試験内容

OpenJIT 初期化部の動作を確認する．

(2) 試験データの内容

Test クラスとして，以下に定義するものを用いる．

```
public class Test {
    public static void main(String args[]) {
        System.out.println("OpenJIT ready!");
    }
}
```

(3) 予想結果及び確認方法

予想結果 OpenJIT の初期化が行われる．

確認方法 デバッガでブレークポイントを設定し，Test クラスを実行する．ブレークポイントでプログラムの実行が停止されることにより OpenJIT の初期化部の動作が完了したと確認できる．

(4) 試験条件

特になし．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. 環境変数 CLASSPATH，JAVA_COMPILER を設定する．
2. デバッガ (gdb) を起動する．
3. ファイル api.c の OpenJIT_compile 関数の末尾の行にブレークポイントを設定する．
4. `run -Dcompile.enable=Test Test` を実行する．
5. 設定したブレークポイントで実行が停止したことを確認する．

(6) 試験結果

試験項目: OpenJIT 初期化部動作試験 (2)	(7) 合否判定

(1) 試験目的，試験内容

OpenJIT を使用しない設定の時に，OpenJIT の初期化が行われないことを確認する．

(2) 試験データの内容

Test クラスとして，以下に定義するものを用いる．

```
public class Test {
    public static void main(String args[]) {
        System.out.println("OpenJIT not ready!");
    }
}
```

(3) 予想結果及び確認方法

予想結果 OpenJIT の初期化が行われない．

確認方法 OpenJIT を使用しない設定の場合，デバッガで，OpenJIT 初期化部動作試験 (1) と同じ方法でブレークポイントを設定しようとしても，OpenJIT モジュールが組み込まれていないので目的のブレークポイントが設定できないことを確認する．

(4) 試験条件

特になし．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. 環境変数 CLASSPATH を設定し，JAVA_COMPILER を消去する．
2. デバッガ (gdb) を起動する．
3. ファイル api.c の OpenJIT_compile 関数の末尾の行にブレークポイントを設定することを試みる．

(6) 試験結果

試験項目: OpenJIT コンパイラフロントエンド制御部動作試験	(7) 合否判定

(1) 試験目的，試験内容

OpenJIT コンパイラフロントエンド制御部の動作を確認する．

(2) 試験データの内容

Test クラスとして，以下に定義するものを用いる．

```
public class Test {
    public static void main(String args[]) {
        System.out.println("OpenJIT ready!");
    }
}
```

(3) 予想結果及び確認方法

予想結果 OpenJIT コンパイラフロントエンド部の実行が行われる．

確認方法 Test クラスを実行し，標準出力の内容を確認する．

(4) 試験条件

検査に必要な出力を行うように改造された OpenJIT フロントエンド制御部を用いる．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. Test.java をコンパイルする (`javac Test.java`) .
2. Test クラスを実行する (`java Test`) .

(6) 試験結果

試験項目: OpenJIT JNI API 登録部動作試験	(7) 合否判定

(1) 試験目的，試験内容

OpenJIT JNI API 登録部の動作を確認する．

(2) 試験データの内容

Test クラスとして，以下に定義するものを用いる．

```
public class Test {
    public static void main(String args[]) {
        System.out.println("OpenJIT ready!");
    }
}
```

(3) 予想結果及び確認方法

予想結果 OpenJIT JNI API が登録される．

確認方法 デバッガでブレークポイントを設定し，Test クラスを実行する．プログラムの実行が停止した時点でデバッガのコマンドを使い，OpenJIT JNI API が登録されていることを確認する．

(4) 試験条件

特になし．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. 環境変数 CLASSPATH，JAVA_COMPILER を設定する．
2. デバッガ (gdb) を起動する．
3. ファイル api.c の OpenJIT_compile 関数の末尾の行にブレークポイントを設定する．
4. `run -Dcompile.enable=Test Test` を実行する．
5. `*linkVector[Vec_CompiledCodeAttribute]` の内容を表示する．

(6) 試験結果

試験項目: OpenJIT コンパイラ起動試験 (1)	(7) 合否判定

(1) 試験目的，試験内容

OpenJIT を有効にした場合に JDK のインタフェースによって OpenJIT が起動できることを確認する．

(2) 試験データの内容

Test クラスとして，以下に定義するものを用いる．

```
public class Test {
    public static void main(String args[]) {
        System.out.println("Hello, World!");
    }
}
```

(3) 予想結果及び確認方法

予想結果 Test クラスが OpenJIT を使って実行される．

確認方法 検査に必要な出力を行うように改造された OpenJIT を使用して Test クラスを実行し，標準出力の内容を 確認する．

(4) 試験条件

検査に必要な出力を行うように改造された OpenJIT を使用する．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. Test.java をコンパイルする (`javac Test.java`) .
2. 環境変数 JAVA_COMPILER を設定する .
3. Test クラスを実行する (`java Test`) .

(6) 試験結果

試験項目: OpenJIT コンパイラ起動試験 (2)	(7) 合否判定

(1) 試験目的，試験内容

OpenJIT を無効にした場合に JDK のインタフェースによって OpenJIT が起動されないことを確認する．

(2) 試験データの内容

Test クラスとして，以下に定義するものを用いる．

```
public class Test {
    public static void main(String args[]) {
        System.out.println("Hello, World!");
    }
}
```

(3) 予想結果及び確認方法

予想結果 Test クラスが OpenJIT を使わずに実行される．

確認方法 OpenJIT コンパイラ起動試験 (1) と同じように，検査に必要な出力を行うように改造された OpenJIT を使用して Test クラスを実行し，標準出力の内容を 確認する．

(4) 試験条件

OpenJIT コンパイラ起動試験 (1) と同じように，検査に必要な出力を行うように改造された OpenJIT を使用する．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. Test.java をコンパイルする (`javac Test.java`) .
2. 環境変数 JAVA_COMPILER を消去する .
3. Test クラスを実行する (`java Test`) .

(6) 試験結果

試験項目: OpenJIT コンパイラ基盤機能動作試験	(7) 合否判定

(1) 試験目的，試験内容

OpenJIT コンパイラ基盤機能が他の各機能を呼び出せることを確認する．

(2) 試験データの内容

Test クラスとして，以下に定義するものを用いる．

```
public class Test {
    public static void main(String args[]) {
        System.out.println("Hello, World!");
    }
}
```

(3) 予想結果及び確認方法

予想結果 OpenJIT コンパイラのフロントエンド及びバックエンドの実行が行われる．

確認方法 検査に必要な出力を行うように改造された OpenJIT コンパイラフロントエンド及びバックエンドを用いて Test クラスを実行し，標準出力の内容を確認する．

(4) 試験条件

検査に必要な出力を行うように改造された OpenJIT コンパイラフロントエンド及びバックエンドを用いる．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. Test.java をコンパイルする (`javac Test.java`) .
2. 環境変数 JAVA_COMPILER を消去する .
3. Test クラスを実行する (`java Test`) .

(6) 試験結果

5.1.2 OpenJIT バイトコードディスコンパイラ機能

試験項目: バイトコード解析部動作試験	(7) 合否判定

(1) 試験目的，試験内容

バイトコード解析部の動作を確認する．

(2) 試験データの内容

Exam クラスとして，以下に定義するものを用いる．

```
import java.util.Enumeration;

public class Exam {
    public Exam(Enumeration enum) {
        System.out.println((enum != null ? "items of Enumeration"
                                : "enum is null"));
        if (enum == null)
            return;
        while (enum.hasMoreElements()) {
            System.out.println(enum.nextElement());
        }
        System.out.println("end of Enumeration");
    }
}
```

(3) 予想結果及び確認方法

予想結果 バイトコードをディスアセンブルした結果が得られると予想される．

確認方法 確認に用いるクラス Exam を定義した上でテストドライバを起動し，標準出力結果を確認する．

(4) 試験条件

テストドライバの内容は，付録 A.1.1を参照のこと．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. Exam.java をコンパイルする (`javac Exam.java`) .
2. テストドライバを起動する (`java Test 0 Exam.class`) .

(6) 試験結果

試験項目: コントロールグラフ出力部動作試験 (1)	(7) 合否判定

(1) 試験目的，試験内容

コントロールグラフ出力部の動作を確認する．

(2) 試験データの内容

Exam クラスとして，以下に定義するものを用いる．

```
import java.util.Enumeration;

public class Exam {
    public Exam(Enumeration enum) {
        System.out.println((enum != null ? "items of Enumeration"
                            : "enum is null"));
        if (enum == null)
            return;
        while (enum.hasMoreElements()) {
            System.out.println(enum.nextElement());
        }
        System.out.println("end of Enumeration");
    }
}
```

(3) 予想結果及び確認方法

予想結果 バイトコードから作られたコントロールグラフが得られる．

確認方法 確認に用いるクラス Exam を定義した上でテストドライバを起動し，標準出力結果を確認する．

(4) 試験条件

テストドライバの内容は，付録 A.1.1を参照のこと．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. Exam.java をコンパイルする (`javac Exam.java`) .
2. テストドライバを起動する (`java Test 1 Exam.class`) .

(6) 試験結果

試験項目: コントロールグラフ出力部動作試験 (2)	(7) 合否判定

(1) 試験目的，試験内容

ベーシックブロック単位の解析の済んだコントロールグラフを出力する動作を確認する．

(2) 試験データの内容

Exam クラスとして，以下に定義するものを用いる．

```
import java.util.Enumeration;

public class Exam {
    public Exam(Enumeration enum) {
        System.out.println((enum != null ? "items of Enumeration"
                                : "enum is null"));
        if (enum == null)
            return;
        while (enum.hasMoreElements()) {
            System.out.println(enum.nextElement());
        }
        System.out.println("end of Enumeration");
    }
}
```

(3) 予想結果及び確認方法

予想結果 ベーシックブロック単位の解析の済んだコントロールグラフが得られる．

確認方法 確認に用いるクラス Exam を定義した上でテストドライバを起動し，標準出力結果を確認する．

(4) 試験条件

テストドライバの内容は，付録 A.1.1を参照のこと．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. Exam.java をコンパイルする (`javac Exam.java`) .
2. テストドライバを起動する (`java Test 2 Exam.class`) .

(6) 試験結果

試験項目: コントロールグラフ出力部動作試験 (3)	(7) 合否判定

(1) 試験目的，試験内容

式単位の解析の済んだコントロールグラフを出力する動作を確認する．

(2) 試験データの内容

Exam クラスとして，以下に定義するものを用いる．

```
import java.util.Enumeration;

public class Exam {
    public Exam(Enumeration enum) {
        System.out.println((enum != null ? "items of Enumeration"
                            : "enum is null"));
        if (enum == null)
            return;
        while (enum.hasMoreElements()) {
            System.out.println(enum.nextElement());
        }
        System.out.println("end of Enumeration");
    }
}
```

(3) 予想結果及び確認方法

予想結果 式単位の解析の済んだコントロールグラフが得られる．

確認方法 確認に用いるクラス Exam を定義した上でテストドライバを起動し，標準出力結果を確認する．

(4) 試験条件

テストドライバの内容は，付録 A.1.1を参照のこと．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. Exam.java をコンパイルする (`javac Exam.java`) .
2. テストドライバを起動する (`java Test 3 Exam.class`) .

(6) 試験結果

試験項目: AST 出力部動作試験 (1)	(7) 合否判定

(1) 試験目的，試験内容

ドミネータツリーを構築する動作を確認する．

(2) 試験データの内容

Exam クラスとして，以下に定義するものを用いる．

```
import java.util.Enumeration;

public class Exam {
    public Exam(Enumeration enum) {
        System.out.println((enum != null ? "items of Enumeration"
                            : "enum is null"));
        if (enum == null)
            return;
        while (enum.hasMoreElements()) {
            System.out.println(enum.nextElement());
        }
        System.out.println("end of Enumeration");
    }
}
```

(3) 予想結果及び確認方法

予想結果 バイトコードから作られたコントロールフローグラフに対応する，ドミネータツリーが得られる．

確認方法 確認に用いるクラス Exam を定義した上でテストドライバを起動し，標準出力結果を確認する．

(4) 試験条件

テストドライバの内容は，付録 A.1.1を参照のこと．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. Exam.java をコンパイルする (`javac Exam.java`) .
2. テストドライバを起動する (`java Test 4 Exam.class`) .

(6) 試験結果

試験項目: AST 出力部動作試験 (2)	(7) 合否判定

(1) 試験目的，試験内容

制御構造を回復する動作を確認する．

(2) 試験データの内容

Exam クラスとして，以下に定義するものを用いる．

```
import java.util.Enumeration;

public class Exam {
    public Exam(Enumeration enum) {
        System.out.println((enum != null ? "items of Enumeration"
                            : "enum is null"));
        if (enum == null)
            return;
        while (enum.hasMoreElements()) {
            System.out.println(enum.nextElement());
        }
        System.out.println("end of Enumeration");
    }
}
```

(3) 予想結果及び確認方法

予想結果 バイトコードから作られたコントロールグラフを解析して得られる構造化された (structured な) コントロールグラフが得られる．

確認方法 確認に用いるクラス Exam を定義した上でテストドライバを起動し，標準出力結果を確認する．

(4) 試験条件

テストドライバの内容は，付録 A.1.1を参照のこと．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. Exam.java をコンパイルする (`javac Exam.java`) .
2. テストドライバを起動する (`java Test 5 Exam.class`) .

(6) 試験結果

試験項目: OpenJIT バイトコードディスコンパイラ機能動作試験	(7) 合否判定

(1) 試験目的，試験内容

OpenJIT バイトコードディスコンパイラ機能の動作を確認する．

(2) 試験データの内容

Exam クラスとして，以下に定義するものを用いる．

```
import java.util.Enumeration;

public class Exam {
    public Exam(Enumeration enum) {
        System.out.println((enum != null ? "items of Enumeration"
                            : "enum is null"));
        if (enum == null)
            return;
        while (enum.hasMoreElements()) {
            System.out.println(enum.nextElement());
        }
        System.out.println("end of Enumeration");
    }
}
```

(3) 予想結果及び確認方法

予想結果 バイトコードをディスコンパイルした結果としての AST が得られる．

確認方法 確認に用いるクラス Exam を定義した上でテストドライバを起動し，標準出力結果を確認する．

(4) 試験条件

テストドライバの内容は，付録 A.1.1を参照のこと．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. Exam.java をコンパイルする (`javac Exam.java`) .
2. テストドライバを起動する (`java Test 6 Exam.class`) .

(6) 試験結果

5.1.3 OpenJIT クラスファイルアノテーション解析機能

試験項目: アノテーション解析部動作試験 (1)	(7) 合否判定
(1) 試験目的，試験内容 A1 型のアノテーション情報を解析する動作を確認する．	
(2) 試験データの内容 A1 型のアノテーション情報．	
(3) 予想結果及び確認方法 予想結果 A1 型のアノテーション情報を表す Annotation オブジェクトが得られる． 確認方法 アノテーション解析部用テストドライバを用いて A1 型のアノテーション情報を解析し，標準出力の内容を調べることにより，適切な Annotation オブジェクトが得られることを確認する．	
(4) 試験条件 アノテーション解析部用テストドライバの内容に関しては，A.1.2.1参照．	

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. テストドライバを実行する (`java TestAnnotationAnalysis A1`) .

(6) 試験結果

試験項目: アノテーション解析部動作試験 (2)	(7) 合否判定
(1) 試験目的，試験内容 A2 型のアノテーション情報を解析する動作を確認する．	
(2) 試験データの内容 A2 型のアノテーション情報．	
(3) 予想結果及び確認方法 予想結果 A2 型のアノテーション情報を表す Annotation オブジェクトが得られる． 確認方法 アノテーション解析部用テストドライバを用いて A2 型のアノテーション情報を解析し，標準出力の内容を調べることにより，適切な Annotation オブジェクトが得られることを確認する．	
(4) 試験条件 アノテーション解析部用テストドライバの内容に関しては，A.1.2.1参照．	

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. テストドライバを実行する (`java TestAnnotationAnalysis A2`) .

(6) 試験結果

試験項目: アノテーション登録部動作試験 (1)	(7) 合否判定

(1) 試験目的，試験内容

A1 型のアノテーション情報を登録する動作を確認する．

(2) 試験データの内容

登録するアノテーション情報は次の定義で与えられる A1 クラスのオブジェクトとする．

```
import OpenJIT.frontend.discompiler.Metaclass;
public class A1 extends Metaclass {
    public void metaInvoke() {
        System.out.println("A1: OpenJIT ready!");
    }
    public String toString() {
        return "metaclass A1";
    }
}
```

(3) 予想結果及び確認方法

予想結果 A1 型のアノテーション情報に対し適切なメタクラスが登録される．

確認方法 アノテーション登録部動作試験用テストドライバを実行し，標準出力の内容を確認する．

(4) 試験条件

アノテーション登録部用テストドライバの内容に関しては，A.1.2.2参照．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. A1 クラスをコンパイルする (`javac A1.java`) .
2. テストドライバを実行する (`java TestAnnotationRegister A1`) .

(6) 試験結果

試験項目: アノテーション登録部動作試験 (2)	(7) 合否判定
(1) 試験目的，試験内容 不正なアノテーション情報が登録されないことを確認する．	
(2) 試験データの内容 なし．	
(3) 予想結果及び確認方法 予想結果 例外が放出される． 確認方法 アノテーション登録部動作試験用テストドライバを実行し，標準出力の内容を確認する．	
(4) 試験条件 アノテーション登録部用テストドライバの内容に関しては，A.1.2.2参照．	

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. テストドライバを実行する (`java TestAnnotationRegister Invalid`) .

(6) 試験結果

試験項目: メタクラス制御部動作試験	(7) 合否判定

(1) 試験目的，試験内容

A1 型のメタクラスを起動するために必要な情報が与えられることを確認する．

(2) 試験データの内容

A1 型のアノテーション情報に対するメタクラスとして次の定義で与えられる A1 クラスを用いる．

```
import OpenJIT.frontend.discompiler.Metaclass;
public class A1 extends Metaclass {
    public void metaInvoke() {
        System.out.println("metaclass A1: invoked.");
    }
    public String toString() {
        return "metaclass A1";
    }
}
```

(3) 予想結果及び確認方法

予想結果 A1 型のメタクラスが起動される．

確認方法 メタクラス制御部用テストドライバを実行し，標準出力の内容を確認する．

(4) 試験条件

メタクラス制御部用テストドライバの内容は，付録 A.1.2.3参照．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. A1 クラスをコンパイルする (`javac A1`) .
2. テストドライバを実行する (`java TestMetaclass A1`) .

(6) 試験結果

試験項目: OpenJIT クラスファイルアノテーション解析機能動作試験	(7) 合否判定

(1) 試験目的，試験内容

OpenJIT クラスファイルアノテーション解析機能の動作を確認する．

(2) 試験データの内容

A1 型のアノテーション情報に対するメタクラスとして次の定義で与えられる A1 クラスを用いる．

```
import OpenJIT.frontend.discompiler.Metaclass;
public class A1 extends Metaclass {
    public void metaInvoke() {
        System.out.println("metaclass A1: invoked.");
    }
    public String toString() {
        return "metaclass A1";
    }
}
```

(3) 予想結果及び確認方法

予想結果 A1 型のアノテーション情報に対して適切なメタクラスが起動される．

確認方法 アノテーション解析部全体試験用テストドライバを起動し，標準出力結果を確認する．

(4) 試験条件

アノテーション解析部全体試験用テストドライバの内容については，付録 A.1.2.4参照．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. A1 クラスをコンパイルする (`javac A1.java`) .
2. テストドライバを起動する (`java TestAll A1`) .

(6) 試験結果

5.1.4 OpenJIT 最適化機能

試験項目: 最適化制御部動作試験	(7) 合否判定
(1) 試験目的，試験内容 最適化制御部の動作を確認する．	
(2) 試験データの内容 特になし．	
(3) 予想結果及び確認方法 予想結果 最適化制御部が起動される． 確認方法 最適化制御部動作試験用のテストドライバである Test クラスを実行する．	
(4) 試験条件 テストドライバの内容に関しては，付録 A.1.3参照．	

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. Test クラスを実行する (`java Test optimize`) .

(6) 試験結果

試験項目: バイトコード出力部動作試験	(7) 合否判定
(1) 試験目的，試験内容 バイトコード出力部の動作を確認する．	
(2) 試験データの内容 特になし．	
(3) 予想結果及び確認方法 予想結果 バイトコード出力部が起動される． 確認方法 最適化制御部動作試験用のテストドライバである Test クラスを実行する．	
(4) 試験条件 テストドライバの内容に関しては，付録 A.1.3参照．	

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. Test クラスを実行する (`java Test gen`) .

(6) 試験結果

試験項目: OpenJIT 最適化機能動作試験	(7) 合否判定
(1) 試験目的，試験内容 最適化機能の動作を確認する．	
(2) 試験データの内容 特になし．	
(3) 予想結果及び確認方法 予想結果 最適化制御部とバイトコード出力部が起動される． 確認方法 最適化制御部動作試験用のテストドライバである Test クラスを実行する．	
(4) 試験条件 テストドライバの内容に関しては，付録 A.1.3参照．	

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. Test クラスを実行する (`java Test all`) .

(6) 試験結果

5.1.5 OpenJIT フローグラフ構築機能

試験項目: AST 等入力部動作試験	(7) 合否判定

(1) 試験目的，試験内容

データの入力動作を確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 AST 等入力部へ入力したものがフィールドに格納されていることが予想される．

確認方法 Test クラスを定義し起動して，入力に対する標準出力結果を確認する．

(4) 試験条件

Test クラスとして，以下の定義を用いる．

```
package OpenJIT.frontend.flowgraph;
import OpenJIT.frontend.discompiler.ControlFlowGraph;
import OpenJIT.frontend.tree.*;
public class Test{
  public static void main(String a[]){
    Statement s = new CompoundStatement(0, null);
    ControlFlowGraph cfg = new ControlFlowGraph(null, null);
    FlowGraph fg = new FlowGraph();
    fg.setAST(s);
    fg.setCFG(cfg);
    if (fg.ast.equals(s) && fg.cfg.equals(cfg))
      System.out.println("FlowGraph input: OK");
    else System.out.println("FlowGraph input: Error");  } }
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. OpenJIT/frontend/flowgraph/Test.java をコンパイルする．
2. java OpenJIT.frontend.flowgraph.Test を実行する．

(6) 試験結果

試験項目: AST 等入力部動作試験 (2)	(7) 合否判定

(1) 試験目的，試験内容

各グラフ構築部が呼び出されることを確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 データフローグラフ構築，コントロール依存グラフ，クラス階層解析が呼び出されることが予想される．

確認方法 OpenJIT.frontend.flowgraph.FlowGraph のサブクラス Test を定義し，実行して標準出力結果を確認する．

(4) 試験条件

Test クラスとして以下に定義するものを用いる．

```
package OpenJIT.frontend.flowgraph;
public class Test extends FlowGraph{
  public static void main(String a[]){
    Test test = new Test();
    test.constructGraph();  }
  public void constructDFG(){
    System.out.println("DataFlowGraph called: OK"); }
  public void constructCDG(){
    System.out.println("ControlDependencyGraph called: OK"); }
  public void constructCH(){
    System.out.println("ClassHierarchyGraph called: OK"); }  }
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. OpenJIT/frontend/flowgraph/Test.java をコンパイルする．
2. java OpenJIT.frontend.flowgraph.Test を実行する．

(6) 試験結果

試験項目: データフローグラフ構築部動作試験 (1)	(7) 合否判定

(1) 試験目的，試験内容

データフローグラフ構築が呼び出されることを確認する．

(2) 試験データの内容

Test クラスとして，以下に定義するものを用いる．

```
public class Test {
    public static void main(String args[]) {
        System.out.println("Hello, World!");
    }
}
```

(3) 予想結果及び確認方法

予想結果 データフローグラフを構築するメソッドが呼ばれることが予想される．

確認方法 DataFlowGraph クラスを検査に必要な出力を行うように改造した OpenJIT コンパイラフロントエンド及びバックエンドを用いて Test クラスを実行し，標準出力の内容を確認する．

(4) 試験条件

検査に必要な出力を行うように改造された DataFlowGraph クラスを用いた OpenJIT コンパイラを用いる．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. Test.java をコンパイルする．
2. java Test を実行する．

(6) 試験結果

試験項目: データフローグラフ構築部動作試験 (2)	(7) 合否判定

(1) 試験目的，試験内容

データフローグラフ構築の動作を確認する．

(2) 試験データの内容

Test クラスとして，以下に定義するものを用いる．

```
public class Test {
    public static void main(String args[]) {
        System.out.println("Hello, World!");
    }
}
```

(3) 予想結果及び確認方法

予想結果 データフローグラフを構築するメソッドが呼ばれデータフローグラフが構築されることが予想される．

確認方法 DataFlowGraph クラスを検査に必要な出力を行うように改造した OpenJIT コンパイラフロントエンド及びバックエンドを用いて Test クラスを実行し，標準出力の内容を確認する．

(4) 試験条件

検査に必要な出力を行うように改造された DataFlowGraph クラスを用いた OpenJIT コンパイラを用いる．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. Test.java をコンパイルする．
2. java Test を実行する．

(6) 試験結果

試験項目: コントロール依存グラフ構築部動作試験 (1)	(7) 合否判定

(1) 試験目的，試験内容

コントロール依存グラフ構築が呼び出されることを確認する．

(2) 試験データの内容

Test クラスとして，以下に定義するものを用いる．

```
public class Test {
    public static void main(String args[]) {
        System.out.println("Hello, World!");
    }
}
```

(3) 予想結果及び確認方法

予想結果 コントロール依存グラフを構築するメソッドが呼ばれることが予想される．

確認方法 ControlDependencyGraph クラスを検査に必要な出力を行うように改造した OpenJIT コンパイラフロントエンド及びバックエンドを用いて Test クラスを実行し，標準出力の内容を確認する．

(4) 試験条件

検査に必要な出力を行うように改造された ControlDependencyGraph クラスを用いた OpenJIT コンパイラを用いる．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. Test.java をコンパイルする．
2. java Test を実行する．

(6) 試験結果

試験項目: コントロール依存グラフ構築部動作試験 (2)	(7) 合否判定

(1) 試験目的，試験内容

コントロール依存グラフ構築の動作を確認する．

(2) 試験データの内容

Test クラスとして，以下に定義するものを用いる．

```
public class Test {
    public static void main(String args[]) {
        System.out.println("Hello, World!");
    }
}
```

(3) 予想結果及び確認方法

予想結果 コントロール依存グラフを構築するメソッドが呼ばれコントロール依存グラフが構築されることが予想される．

確認方法 ControlDependencyGraph クラスを検査に必要な出力を行うように改造した OpenJIT コンパイラフロントエンド及びバックエンドを用いて Test クラスを実行し，標準出力の内容を確認する．

(4) 試験条件

検査に必要な出力を行うように改造された ControlDependencyGraph クラスを用いた OpenJIT コンパイラを用いる．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. Test.java をコンパイルする．
2. java Test を実行する．

(6) 試験結果

試験項目: クラス階層解析部動作試験 (1)	(7) 合否判定

(1) 試験目的，試験内容

クラス階層解析が呼び出されることを確認する．

(2) 試験データの内容

Test クラスとして，以下に定義するものを用いる．

```
public class Test {
    public static void main(String args[]) {
        System.out.println("Hello, World!");
    }
}
```

(3) 予想結果及び確認方法

予想結果 クラス階層解析を行うメソッドが呼ばれることが予想される．

確認方法 ClassHierarchyGraph クラスを検査に必要な出力を行うように改造した OpenJIT コンパイラフロントエンド及びバックエンドを用いて Test クラスを実行し，標準出力の内容を確認する．

(4) 試験条件

検査に必要な出力を行うように改造された ClassHierarchyGraph クラスを用いた OpenJIT コンパイラを用いる．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. Test.java をコンパイルする．
2. java Test を実行する．

(6) 試験結果

試験項目: クラス階層解析部動作試験 (2)	(7) 合否判定

(1) 試験目的，試験内容

クラス階層解析の動作を確認する．

(2) 試験データの内容

Test クラスとして，以下に定義するものを用いる．

```
public class Test {
    public static void main(String args[]) {
        System.out.println("Hello, World!");
    }
}
```

(3) 予想結果及び確認方法

予想結果 クラス階層解析を行うメソッドが呼ばれクラス階層グラフが構築されることが予想される．

確認方法 ClassHierarchyGraph クラスを検査に必要な出力を行うように改造した OpenJIT コンパイラフロントエンド及びバックエンドを用いて Test クラスを実行し，標準出力の内容を確認する．

(4) 試験条件

検査に必要な出力を行うように改造された ClassHierarchyGraph クラスを用いた OpenJIT コンパイラを用いる．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. Test.java をコンパイルする．
2. java Test を実行する．

(6) 試験結果

試験項目: OpenJIT フローグラフ構築機能動作試験	(7) 合否判定

(1) 試験目的，試験内容

OpenJIT フローグラフ構築機能の動作を確認する．

(2) 試験データの内容

Test クラスとして，以下に定義するものを用いる．

```
public class Test {
    public static void main(String args[]) {
        System.out.println("Hello, World!");
    }
}
```

(3) 予想結果及び確認方法

予想結果 入力を受け取り，データフローグラフの構築，コントロール依存グラフの構築，クラス階層解析を行うメソッド順に呼ばれることが予想される．

確認方法 DataFlowGraph クラス，ControlDependencyGraph クラス，ClassHierarchyGraph クラスを検査に必要な出力を行うように改造した OpenJIT コンパイラフロントエンド及びバックエンドを用いて Test クラスを実行し，標準出力の内容を確認する．

(4) 試験条件

検査に必要な出力を行うように改造された DataFlowGraph クラス，ControlDependencyGraph クラス，ClassHierarchyGraph クラスを用いた OpenJIT コンパイラを用いる．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. Test.java をコンパイルする．
2. java Test を実行する．

(6) 試験結果

5.1.6 OpenJIT フローグラフ解析機能

試験項目: データフロー関数登録部動作試験	(7) 合否判定

(1) 試験目的，試験内容

データフロー関数登録の動作を確認する．

(2) 試験データの内容

Test クラスとして，以下に定義するものを用いる．

```
public class Test {
    public static void main(String args[]) {
        System.out.println("Hello, World!");
    }
}
```

(3) 予想結果及び確認方法

予想結果 到達定義の解析，利用可能な式の解析，行きている式の解析を行うデータフロー関数が登録されることが予想される．

確認方法 DFFunctionRegister クラスを検査に必要な出力を行うように改造した OpenJIT コンパイラフロントエンド及びバックエンドを用いて Test クラスを実行し，標準出力の内容を確認する．

(4) 試験条件

検査に必要な出力を行うように改造された DFFunctionRegister クラスを用いた OpenJIT コンパイラを用いる．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. Test.java をコンパイルする．
2. java Test を実行する．

(6) 試験結果

試験項目: フローグラフ解析部動作試験 (1)	(7) 合否判定

(1) 試験目的，試験内容

各データフロー解析を呼び出せることを確認する．

(2) 試験データの内容

Test クラスとして，以下に定義するものを用いる．

```
public class Test{
  public Test(){
    int a = 1;
    int b = 0;
    int c = 3;
    a = b + 1;
    b = c + a;
    a = b + c;
  }
}
```

(3) 予想結果及び確認方法

予想結果 到達定義の解析，利用可能な式の解析，行きている式の解析が呼び出されることが予想される．

確認方法 各データフロー解析クラスを検査に必要な出力を行うように改造した OpenJIT コンパイラフロントエンド及びバックエンドを用いて Test クラスを実行し，標準出力の内容を確認する．

(4) 試験条件

検査に必要な出力を行うように改造された各データフロー解析クラスを用いた OpenJIT コンパイラを用いる．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. Test.java をコンパイルする．
2. java Test を実行する．

(6) 試験結果

試験項目: フローグラフ解析部動作試験 (2)	(7) 合否判定

(1) 試験目的，試験内容

到達定義の解析の動作を確認する．

(2) 試験データの内容

Test クラスとして，以下に定義するものを用いる．

```
public class Test{
  public Test(){
    int a = 1;
    int b = 0;
    int c = 3;
    a = b + 1;
    b = c + a;
    a = b + c;
  }
}
```

(3) 予想結果及び確認方法

予想結果 到達定義の解析が行われることが予想される．

確認方法 ReachingAnaluyzer クラスを検査に必要な出力を行うように改造した OpenJIT コンパイラフロントエンド及びバックエンドを用いて Test クラスを実行し，標準出力の内容を確認する．

(4) 試験条件

検査に必要な出力を行うように改造された ReachingAnalyzer クラスを用いた OpenJIT コンパイラを用いる．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. Test.java をコンパイルする．
2. java Test を実行する．

(6) 試験結果

試験項目: フローグラフ解析部動作試験 (3)	(7) 合否判定

(1) 試験目的，試験内容

利用可能な式の解析の動作を確認する．

(2) 試験データの内容

Test クラスとして，以下に定義するものを用いる．

```
public class Test{
  public Test(){
    int a = 1;
    int b = 0;
    int c = 3;
    a = b + 1;
    b = c + a;
    a = b + c;
  }
}
```

(3) 予想結果及び確認方法

予想結果 利用可能な式の解析が行われることが予想される．

確認方法 AvailableAnaluyzer クラスを検査に必要な出力を行うように改造した OpenJIT コンパイラフロントエンド及びバックエンドを用いて Test クラスを実行し，標準出力の内容を確認する．

(4) 試験条件

検査に必要な出力を行うように改造された AvailableAnalyzer クラスを用いた OpenJIT コンパイラを用いる．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. Test.java をコンパイルする．
2. java Test を実行する．

(6) 試験結果

試験項目: フローグラフ解析部動作試験 (4)	(7) 合否判定

(1) 試験目的，試験内容

生きている式の解析の動作を確認する．

(2) 試験データの内容

Test クラスとして，以下に定義するものを用いる．

```
public class Test{
  public Test(){
    int a = 1;
    int b = 0;
    int c = 3;
    a = b + 1;
    b = c + a;
    a = b + c;
  }
}
```

(3) 予想結果及び確認方法

予想結果 到達定義の解析が行われることが予想される．

確認方法 LivenessAnaluyzer クラスを検査に必要な出力を行うように改造した OpenJIT コンパイラフロントエンド及びバックエンドを用いて Test クラスを実行し，標準出力の内容を確認する．

(4) 試験条件

検査に必要な出力を行うように改造された LivenessAnalyzer クラスを用いた OpenJIT コンパイラを用いる．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. Test.java をコンパイルする．
2. java Test を実行する．

(6) 試験結果

試験項目: 不動点検出部動作試験	(7) 合否判定

(1) 試験目的，試験内容

不動点検出の動作を確認する．

(2) 試験データの内容

Test クラスとして，以下に定義するものを用いる．

```
public class Test{
  public Test(){
    int a = 1;
    int b = 0;
    int c = 3;
    a = b + 1;
    b = c + a;
    a = b + c;
  }
}
```

(3) 予想結果及び確認方法

予想結果 不動点検の検出が行われることが予想される．

確認方法 FixedPointDetector クラスを検査に必要な出力を行うように改造した OpenJIT コンパイラフロントエンド及びバックエンドを用いて Test クラスを実行し，標準出力の内容を確認する．

(4) 試験条件

検査に必要な出力を行うように改造された FixedPointDetector クラスを用いた OpenJIT コンパイラを用いる．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. Test.java をコンパイルする．
2. java Test を実行する．

(6) 試験結果

試験項目: クラス階層解析部動作試験	(7) 合否判定

(1) 試験目的，試験内容

クラス階層解析の動作を確認する．

(2) 試験データの内容

Test クラスとして，以下に定義するものを用いる．

```
public class Test{
  public Test(){
    int a = 1;
    int b = 0;
    int c = 3;
    a = b + 1;
    b = c + a;
    a = b + c;
  }
}
```

(3) 予想結果及び確認方法

予想結果 クラス階層解析が行われることが予想される．

確認方法 ClassHierarchyAnaluyzer クラスを検査に必要な出力を行うように改造した OpenJIT コンパイラフロントエンド及びバックエンドを用いて Test クラスを実行し，標準出力の内容を確認する．

(4) 試験条件

検査に必要な出力を行うように改造された ClassHierarchyAnalyzer クラスを用いた OpenJIT コンパイラを用いる．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. Test.java をコンパイルする．
2. java Test を実行する．

(6) 試験結果

試験項目: OpenJIT フローグラフ解析機能動作試験	(7) 合否判定

(1) 試験目的，試験内容

OpenJIT フローグラフ解析機能の動作を確認する．

(2) 試験データの内容

Test クラスとして，以下に定義するものを用いる．

```
public class Test{
  public Test(){
    int a = 1;
    int b = 0;
    int c = 3;
    a = b + 1;
    b = c + a;
    a = b + c;  } }
```

(3) 予想結果及び確認方法

予想結果 各種フローグラフ解析機能が呼び出され，解析が行われることが予想される．

確認方法 OpenJIT フローグラフ解析機能のクラスを検査に必要な出力を行うように改造した OpenJIT コンパイラフロントエンド及びバックエンドを用いて Test クラスを実行し，標準出力の内容を確認する．

(4) 試験条件

検査に必要な出力を行うように改造された FlowGraphAnalysis クラス，DFFunctionRegister クラス，ReachingAnalyzer クラス，AvailableAnalyzer クラス，LivenessAnalyzer クラス，FixedPointDetector クラス，ClassHierarchyAnalysis クラスを用いた OpenJIT コンパイラを用いる．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. Test.java をコンパイルする．
2. java Test を実行する．

(6) 試験結果

5.1.7 OpenJIT プログラム変換機能

試験項目: AST 変換ルール登録部動作試験	(7) 合否判定

(1) 試験目的，試験内容

AST 変換ルールが登録できることを確認する．

(2) 試験データの内容

整数定数 3 を表す AST を整数定数 7 を表す AST に変換するルールを考える．

(3) 予想結果及び確認方法

予想結果 AST 変換ルールを保持するテーブルに AST 変換ルールが登録される．

確認方法 AST 変換ルールを登録した後のテーブルの内容を確認するテストドライバを実行し，標準出力の内容を確認する．

(4) 試験条件

使用するテストドライバについては，付録 A.1.4.1参照．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. テストドライバを実行する (java Test) .

(6) 試験結果

試験項目: AST パターンマッチ部動作試験 (1)	(7) 合否判定

(1) 試験目的，試験内容

変換ルールに登録されたパターンにマッチする場合の動作を確認する．

(2) 試験データの内容

整数定数 3 を表す AST を整数定数 7 を表す AST に変換するルールが登録されている状態で，整数定数 3 を表す AST をマッチする．

(3) 予想結果及び確認方法

予想結果 マッチする AST が得られる．

確認方法 マッチする AST を表示するテストドライバを実行し，標準出力の内容を確認する．

(4) 試験条件

使用するテストドライバについては，付録 A.1.4.2参照．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. テストドライバを実行する (java Test) .

(6) 試験結果

試験項目: AST パターンマッチ部動作試験 (2)	(7) 合否判定

(1) 試験目的，試験内容

変換ルールに登録されたパターンにマッチしない場合の動作を確認する．

(2) 試験データの内容

整数定数 3 を表す AST を整数定数 7 を表す AST に変換するルールが登録されている状態で，整数定数 7 を表す AST をマッチする．

(3) 予想結果及び確認方法

予想結果 null が得られる．

確認方法 マッチする AST を表示するテストドライバを実行し，標準出力の内容を確認する．

(4) 試験条件

使用するテストドライバについては，付録 A.1.4.3参照．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. テストドライバを実行する (java Test) .

(6) 試験結果

試験項目: AST 変換部動作試験 (1)	(7) 合否判定

(1) 試験目的，試験内容

変換ルールに登録されたパターンの AST が変換されることを確認する．

(2) 試験データの内容

整数定数 3 を表す AST を整数定数 7 を表す AST に変換するルールが登録されている状態で，整数定数 3 を表す AST を変換する．

(3) 予想結果及び確認方法

予想結果 変換された AST が得られる．

確認方法 変換結果の AST を表示するテストドライバを実行し，標準出力の内容を確認する．

(4) 試験条件

使用するテストドライバについては，付録 A.1.4.4参照．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. テストドライバを実行する (java Test) .

(6) 試験結果

試験項目: AST 変換部動作試験 (2)	(7) 合否判定

(1) 試験目的，試験内容

変換ルールに登録されていないパターンの AST が変換されないことを確認する．

(2) 試験データの内容

整数定数 3 を表す AST を整数定数 7 を表す AST に変換するルールが登録されている状態で，整数定数 7 を表す AST を変換する．

(3) 予想結果及び確認方法

予想結果 変換されなかった AST が得られる．

確認方法 変換結果の AST を表示するテストドライバを実行し，標準出力の内容を確認する．

(4) 試験条件

使用するテストドライバについては，付録 A.1.4.5参照．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. テストドライバを実行する (java Test) .

(6) 試験結果

試験項目: OpenJIT プログラム変換機能動作試験	(7) 合否判定

(1) 試験目的，試験内容

OpenJIT プログラム変換機能の動作を確認する．

(2) 試験データの内容

変換ルールとして整数定数 3 を表す AST を整数定数 7 を表す AST に変換するルールを登録し，整数定数 3 を表す AST をマッチし，変換する．

(3) 予想結果及び確認方法

予想結果 プログラム変換が行われる．

確認方法 プログラム変換機能の動作を試験するテストドライバを実行し，標準出力の内容を確認する．

(4) 試験条件

使用するテストドライバについては，付録 A.1.4.6参照．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

1. テストドライバを実行する (java Test) .

(6) 試験結果

5.2 OpenJIT バックエンドシステム

5.2.1 OpenJIT ネイティブコード変換機能

試験項目: ネイティブコード変換試験	(7) 合否判定

(1) 試験目的，試験内容

与えられたバイトコードに対し，ネイティブコードが生成されることを確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 ネイティブコードがメモリ上に生成されていることが予想される．

確認方法 デバッガ (gdb) でブレークポイントを設定し，nativeTest クラスを実行する．プログラムがブレークポインタで停止した時点で，デバッガのコマンドを使ってネイティブコードが生成されていることを確認する．

(4) 試験条件

nativeTest クラスとして，以下に定義するものを用いる．

```
class nativeTest {
    static int args_size;
    public static void main(String argv[]) {
        args_size = argv.length;
    }
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．
2. デバッガ (gdb) を起動する．
3. ファイル api.c の OpenJIT_compile 関数の do_execute_java_method_vararg を呼び出した後の行にブレークポイントを設定する．
4. `run -Dcompile.enable=nativeTest nativeTest` を実行する．
5. `mb->CompiledCode` から `mb->CompiledCodeInfo` のサイズの逆アセンブルを行う．

(6) 試験結果

試験項目: ネイティブコード変換機能動作試験 (1)	(7) 合否判定

(1) 試験目的，試験内容

OpenJIT ネイティブコード変換機能が，OpenJIT 中間コード変換機能を呼び出せることを確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 OpenJIT ネイティブコード変換機能が，OpenJIT 中間コード変換機能を呼び出せることが予想される．

確認方法 OpenJIT.Compile のサブクラスを Test を定義し，OpenJIT システムに組み込む．一つのメソッドだけをコンパイルするよう指示し，java を起動し標準出力結果を確認する．

(4) 試験条件

Test クラスとして，以下に定義するものを用いる．

```
package OpenJIT;
class Test extends Compile {
    void parseBytecode() {
        System.out.println("parseBytecode ok");
        System.exit(0);
    }
    void convertRTL() { }
    void optimizeRTL() { }
    void regAlloc() { }
    void genNativeCode() { }
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. OpenJIT/Test.java をコンパイルする．
2. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．
3. 環境変数 OpenJIT_COMPILER を OpenJIT/Test に設定する．
4. `java -Dcompile.enable=OpenJIT/Compile.compile empty` を実行する．

(6) 試験結果

試験項目: ネイティブコード変換機能動作試験 (2)	(7) 合否判定

(1) 試験目的，試験内容

OpenJIT ネイティブコード変換機能が，OpenJIT RTL 変換機能を呼び出せることを確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 OpenJIT ネイティブコード変換機能が，OpenJIT RTL 変換機能を呼び出せることが予想される．

確認方法 OpenJIT.Compile のサブクラスを Test を定義し，OpenJIT システムに組み込む．一つのメソッドだけをコンパイルするよう指示し，java を起動し標準出力結果を確認する．

(4) 試験条件

Test クラスとして，以下に定義するものを用いる．

```
package OpenJIT;
class Test extends Compile {
    void parseBytecode() { }
    void convertRTL() {
        System.out.println("convertRTL ok");
        System.exit(0);
    }
    void optimizeRTL() { }
    void regAlloc() { }
    void genNativeCode() { }
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. OpenJIT/Test.java をコンパイルする．
2. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．
3. 環境変数 OpenJIT_COMPILER を OpenJIT/Test に設定する．
4. `java -Dcompile.enable=OpenJIT/Compile.compile empty` を実行する．

(6) 試験結果

試験項目: ネイティブコード変換機能動作試験 (3)	(7) 合否判定

(1) 試験目的，試験内容

OpenJIT ネイティブコード変換機能が，OpenJIT Peephole 最適化 RTL 変換機能を呼び出せることを確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 OpenJIT ネイティブコード変換機能が，OpenJIT Peephole 最適化 RTL 変換機能を呼び出せることが予想される．

確認方法 OpenJIT.Compile のサブクラスを Test を定義し，OpenJIT システムに組み込む．一つのメソッドだけをコンパイルするよう指示し，java を起動し標準出力結果を確認する．

(4) 試験条件

Test クラスとして，以下に定義するものを用いる．

```
package OpenJIT;
class Test extends Compile {
    void parseBytecode() { }
    void convertRTL() { }
    void optimizeRTL() {
        System.out.println("optimizeRTL ok");
        System.exit(0);
    }
    void regAlloc() { }
    void genNativeCode() { }
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. OpenJIT/Test.java をコンパイルする．
2. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．
3. 環境変数 OpenJIT_COMPILER を OpenJIT/Test に設定する．
4. `java -Dcompile.enable=OpenJIT/Compile.compile empty` を実行する．

(6) 試験結果

試験項目: メソッド情報取得試験 (1)	(7) 合否判定

(1) 試験目的，試験内容

メソッドに関する JDK の内部構造 (クラス名，メソッド名，シグナチャ) が Java のデータ構造に変換されていることを確認する．

(2) 試験データの内容

以下のコンパイルメソッド指定オプション (-Dcompile) について試験する．

- -Dcompile.enable=OpenJIT/Compile.compile
- -Dcompile.enable=java/lang/Thread

(3) 予想結果及び確認方法

予想結果 メソッドに関する JDK の内部構造 (クラス名，メソッド名，シグナチャ) が Java のデータ構造に変換されていることが予想される．

確認方法 OpenJIT.Compile のサブクラスを Test を定義し，OpenJIT システムに組み込む．一つのメソッドだけをコンパイルするよう指示し，java を起動し標準出力結果を確認する．

(4) 試験条件

Test クラスとして，付録 A.2.1に定義するものを用いる．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. OpenJIT/Test.java をコンパイルする．
2. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．
3. 環境変数 OpenJIT_COMPILER を OpenJIT/Test に設定する．
4. `java <オプション> empty` を実行する．

(6) 試験結果

試験項目: メソッド情報取得試験 (2)	(7) 合否判定

(1) 試験目的，試験内容

メソッドに関する JDK の内部構造 (アクセス情報) が Java のデータ構造に変換されていることを確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 メソッドに関する JDK の内部構造 (アクセス情報) が OpenJIT .Compile クラスのオブジェクトの access フィールドに設定されていることが予想される．

確認方法 OpenJIT.Compile のサブクラスを Test を定義し，OpenJIT システムに組み込む．java を起動し標準出力結果を確認する．

(4) 試験条件

Test クラスとして，以下に定義するものを用いる．

```
package OpenJIT;
class Test extends Compile {
    void parseBytecode() { }
    void convertRTL() {}
    void optimizeRTL() {}
    void genNativeCode() {}
    void regAlloc() {}
    public boolean compile() {
      if (clazz.getName().regionMatches(0, "java.lang.Thread", 0, 16)) {
          System.out.println(this + " access:" + access);
      }
      return false;
    }
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. OpenJIT/Test.java をコンパイルする．
2. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．
3. 環境変数 OpenJIT_COMPILER を OpenJIT/Test に設定する．
4. java を起動する．

(6) 試験結果

試験項目: メソッド情報取得試験 (3)	(7) 合否判定

(1) 試験目的，試験内容

メソッドに関する JDK の内部構造 (nlocals, maxstack, args_size 情報) が Java のデータ構造に変換されていることを確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 メソッドに関する JDK の内部構造 (nlocals, maxstack, args_size 情報) が Java のデータ構造に変換されていることが予想される．

確認方法 OpenJIT.Compile のサブクラスを Test を定義し，OpenJIT システムに組み込む．java を起動し標準出力結果を確認する．

(4) 試験条件

Test クラスとして，以下に定義するものを用いる．

```
package OpenJIT;
class Test extends Compile {
    void parseBytecode() { }
    void convertRTL() {}
    void optimizeRTL() {}
    void genNativeCode() {}
    void regAlloc() {}
    public boolean compile() {
      if (clazz.getName().regionMatches(0, "java.lang.Thread", 0, 16)) {
          System.out.println(this);
          System.out.println("nlocals:\t" + nlocals +
                             "\tmaxstack:\t" + maxstack +
                             "\targs_size:\t" + args_size);
      }
      return false;
    }
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. OpenJIT/Test.java をコンパイルする．
2. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．
3. 環境変数 OpenJIT_COMPILER を OpenJIT/Test に設定する．
4. java を起動する．

(6) 試験結果

試験項目: バイトコードアクセス試験 (1)	(7) 合否判定

(1) 試験目的，試験内容

JDK の内部構造であるバイトコードのサイズが Java から読めることを確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 JDK の内部構造であるバイトコードのサイズが Java から読めることが予想される．

確認方法 OpenJIT.Compile のサブクラスを Test を定義し，OpenJIT システムに組み込む．java を起動し標準出力結果を確認する．

(4) 試験条件

Test クラスとして，付録 A.2.2に定義するものを用いる．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. OpenJIT/Test.java をコンパイルする．
2. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．
3. 環境変数 OpenJIT_COMPILER を OpenJIT/Test に設定する．
4. java を起動する．

(6) 試験結果

試験項目: バイトコードアクセス試験 (2)	(7) 合否判定

(1) 試験目的，試験内容

JDK の内部構造であるバイトコードが Java から読めることを確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 JDK の内部構造であるバイトコードが Java から読めることが予想される．

確認方法 OpenJIT.Compile のサブクラスを Test を定義し，OpenJIT システムに組み込む．java を起動し標準出力結果を確認する．

(4) 試験条件

Test クラスとして，以下に定義するものを用いる．

```
package OpenJIT;
class Test extends Compile {
    void parseBytecode() { }
    void convertRTL() {}
    void optimizeRTL() {}
    void genNativeCode() {}
    void regAlloc() {}
    public boolean compile() {
      if (clazz.getName().regionMatches(0, "java.lang.Thread", 0, 16)) {
          System.out.println(this);
          for(int i = 0; i < bytecode.length; i++) {
              if (bytecode[i]<16) System.out.print("0");
              System.out.print(Integer.toHexString(bytecode[i]&0xff));
          }
          System.out.println();
      }
      return false;
    }
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. OpenJIT/Test.java をコンパイルする．
2. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．
3. 環境変数 OpenJIT_COMPILER を OpenJIT/Test に設定する．
4. java を起動する．

(6) 試験結果

試験項目: 生成コードメモリ管理試験 (1)	(7) 合否判定

(1) 試験目的，試験内容

生成するネイティブコードのためのメモリ領域が確保できることを確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 生成するネイティブコードのためのメモリ領域が確保できることが予想される．

確認方法 OpenJIT.Compile のサブクラスを Test を定義し，OpenJIT システムに組み込む．java を起動し標準出力結果を確認する．

(4) 試験条件

Test クラスとして，以下に定義するものを用いる．

```
package OpenJIT;
class Test extends Compile {
    void parseBytecode() { }
    void convertRTL() {}
    void optimizeRTL() {}
    void genNativeCode() {}
    void regAlloc() {}
    public boolean compile() {
        if (clazz != this.getClass()) return false;
        for (int size = 1; size <= 1 << 20; size <<= 1) {
            NativeCodeAlloc(size);
            System.out.println("addr:" +
                               Integer.toHexString(code_area)
                               + "\tsize:" + size);
        }
        return false;
    }
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. OpenJIT/Test.java をコンパイルする．
2. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．
3. 環境変数 OpenJIT_COMPILER を OpenJIT/Test に設定する．
4. java を起動する．

(6) 試験結果

試験項目: 生成コードメモリ管理試験 (2)	(7) 合否判定

(1) 試験目的，試験内容

確保したメモリ領域に命令が書き込めることを確認する．

確保したメモリ領域の先頭から，0 から 65536 までの 32bit 値を順に書き込んだ後，先頭から順に値を読み出して，書き込んだデータと同じかチェックする．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 確保したメモリ領域に命令が書き込めることが予想される．

確認方法 OpenJIT.Compile のサブクラスを Test を定義し，OpenJIT システムに組み込む．java を起動し標準出力結果を確認する．

(4) 試験条件

Test クラスとして，付録 A.2.3に定義するものを用いる．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. OpenJIT/Test.java をコンパイルする．
2. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．
3. 環境変数 OpenJIT_COMPILER を OpenJIT/Test に設定する．
4. java を起動する．

(6) 試験結果

5.2.2 OpenJIT 中間コード変換機能

試験項目: 中間言語変換試験 (1)	(7) 合否判定

(1) 試験目的，試験内容

バイトコードがバックエンド中間コードに変換できることを確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 バイトコードがバックエンド中間コードに変換できることをが予想される．

確認方法 OpenJIT.Compile のサブクラスを Test を定義し，OpenJIT システムに組み込む．java を起動し標準出力結果を確認する．

(4) 試験条件

Test クラスとして，付録 A.2.4に定義するものを用いる．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. OpenJIT/Test.java をコンパイルする．
2. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．
3. 環境変数 OpenJIT_COMPILER を OpenJIT/Test に設定する．
4. java を起動する．

(6) 試験結果

試験項目: 中間言語変換試験 (2)	(7) 合否判定

(1) 試験目的，試験内容

中間言語変換機能がメソッド引数展開を呼び出せることを確認する．

(2) 試験データの内容

OpenJIT のコンパイルメソッド選択オプションを使って，コンパイルするメソッドを指定する．

- test.test1(メソッド呼び出しを持つメソッド)
- test.test2(メソッド呼び出しを持たないメソッド)

(3) 予想結果及び確認方法

予想結果 中間言語変換機能がメソッド引数展開を呼び出せることが予想される．

確認方法 OpenJIT/ParseBytecode.callmethod と (2) で指定したメソッドだけをコンパイルするように指示して，java_g を用いて test クラスを実行する．test.test1 を指定したときには OpenJIT /ParseBytecode .callmethod をコンパイルしたメッセージが表示され，test.test2 を指定したときには表示されない．

(4) 試験条件

test クラスとして，以下に定義するものを用いる．

```
class test {
    public static void main(String argv[]) {
        test1(0);
        test2(0);
    }
    static int test1(int a) { method(); return a; }
    static int test2(int a) { return a; }
    static void method() { }
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1．環境変数 CLASSPATH，JAVA_COMPILER を設定する．

2．java_g -Dcompile.enable=test.test1:OpenJIT/ParseBytecode.ca llMethod test を実行する．

3．java_g -Dcompile.enable=test.test2:OpenJIT/ParseBytecode.ca llMethod test を実行する．

(6) 試験結果

試験項目: 中間言語変換試験 (3)	(7) 合否判定

(1) 試験目的，試験内容

中間言語変換機能が命令パターンマッチング (optim_neg) を呼び出せることを確認する．

(2) 試験データの内容

OpenJIT のコンパイルメソッド選択オプションを使って，コンパイルするメソッドを指定する．

- test.test1(マッチするメソッド)
- test.test2(マッチしないメソッド)

(3) 予想結果及び確認方法

予想結果 中間言語変換機能が命令パターンマッチングを呼び出せることが予想される．

確認方法 OpenJIT/ParseBytecode.optim_neg と (2) で指定したメソッドだけをコンパイルするように指示して，java_g を用いて test クラスを実行する．test.test1 を指定したときには OpenJIT /ParseBytecode .optim_neg をコンパイルしたメッセージが表示され，test.test2 を指定したときには表示されない．

(4) 試験条件

test クラスとして，以下に定義するものを用いる．

```
class test {
    static boolean b;
    public static void main(String argv[]) {
        test1();    test2();
    }
    static void test1() { b = !b; }
    static void test2() { b = b; }
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1．環境変数 CLASSPATH，JAVA_COMPILER を設定する．

2．java_g -Dcompile.enable=test.test1:OpenJIT/ParseBytecode.op tim_neg test を実行する．

3．java_g -Dcompile.enable=test.test2:OpenJIT/ParseBytecode.op tim_neg test を実行する．

(6) 試験結果

試験項目: 中間言語変換試験 (4)	(7) 合否判定

(1) 試験目的，試験内容

中間言語変換機能が命令パターンマッチング (optim_lcmp) を呼び出せることを確認する．

(2) 試験データの内容

OpenJIT のコンパイルメソッド選択オプションを使って，コンパイルするメソッドを指定する．

- test.test1(マッチするメソッド)
- test.test2(マッチしないメソッド)

(3) 予想結果及び確認方法

予想結果 中間言語変換機能が命令パターンマッチングを呼び出せることが予想される．

確認方法 OpenJIT/ParseBytecode.optim_lcmp と (2) で指定したメソッドだけをコンパイルするように指示して，java_g を用いて test クラスを実行する．test.test1 を指定したときには OpenJIT /ParseBytecode .optim_lcmp をコンパイルしたメッセージが表示され，test.test2 を指定したときには表示されない．

(4) 試験条件

test クラスとして，以下に定義するものを用いる．

```
class test {
    public static void main(String argv[]) {
        test1(0L,1L);        test2(0L,1L);
    }
    static boolean test1(long x, long y) { return x == y; }
    static boolean test2(long x, long y) { return true; }
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1．環境変数 CLASSPATH，JAVA_COMPILER を設定する．

2．java_g -Dcompile.enable=test.test1:OpenJIT/ParseBytecode.op tim_lcmp test を実行する．

3．java_g -Dcompile.enable=test.test2:OpenJIT/ParseBytecode.op tim_lcmp test を実行する．

(6) 試験結果

試験項目: 中間言語変換試験 (5)	(7) 合否判定

(1) 試験目的，試験内容

中間言語変換機能が命令パターンマッチング (optim_fcmp) を呼び出せることを確認する．

(2) 試験データの内容

OpenJIT のコンパイルメソッド選択オプションを使って，コンパイルするメソッドを指定する．

- test.test1(マッチするメソッド)
- test.test2(マッチしないメソッド)

(3) 予想結果及び確認方法

予想結果 中間言語変換機能が命令パターンマッチングを呼び出せることが予想される．

確認方法 OpenJIT/ParseBytecode.optim_fcmp と (2) で指定したメソッドだけをコンパイルするように指示して，java_g を用いて test クラスを実行する．test.test1 を指定したときには OpenJIT /ParseBytecode .optim_fcmp をコンパイルしたメッセージが表示され，test.test2 を指定したときには表示されない．

(4) 試験条件

test クラスとして，以下に定義するものを用いる．

```
class test {
    public static void main(String argv[]) {
        test1(0.0,1.0);        test2(0.0,1.0);
    }
    static boolean test1(double x, double y) { return x == y; }
    static boolean test2(double x, double y) { return true; }
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1．環境変数 CLASSPATH，JAVA_COMPILER を設定する．

2．java_g -Dcompile.enable=test.test1:OpenJIT/ParseBytecode.op tim_fcmp test を実行する．

3．java_g -Dcompile.enable=test.test2:OpenJIT/ParseBytecode.op tim_fcmp test を実行する．

(6) 試験結果

試験項目: メソッド引数展開試験 (1)	(7) 合否判定

(1) 試験目的，試験内容

メソッド呼び出しの引数がバックエンド中間コードに展開できることを確認する．引数の個数だけ，中間コードが生成されることを確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 メソッド呼び出しの引数がバックエンド中間コードに展開できることが予想される．引数の個数だけ，中間コードが生成されることが予想される．

確認方法 OpenJIT.Compile のサブクラスを Test を定義し，OpenJIT システムに組み込む．test クラスを実行し，標準出力結果を確認する．

(4) 試験条件

- Test クラスとして，付録 A.2.5に定義するものを用いる．
- test クラスとして，以下に定義するものを用いる．

```
class test {
    public static void main(String argv[]) { test(); }
    static void test() {
        m();
        m(1);
        m(1,2);
        m(1,2,3);
    }
    static void m() {}
    static void m(int a1) {}
    static void m(int a1,int a2) {}
    static void m(int a1,int a2,int a3) {}
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. OpenJIT/Test.java をコンパイルする．

2. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．

3. 環境変数 OpenJIT_COMPILER を OpenJIT/Test に設定する．

4. java test を実行する．

(6) 試験結果

試験項目: メソッド引数展開試験 (2)	(7) 合否判定

(1) 試験目的，試験内容

メソッド呼び出しの引数がバックエンド中間コードに展開できることを確認する．引数の型によって，生成される中間コードのオペランドが異なることを確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 メソッド呼び出しの引数がバックエンド中間コードに展開できることが予想される．引数の型によって，生成される中間コードのオペランドが異なることを確認する．

確認方法 OpenJIT.Compile のサブクラスを Test を定義し，OpenJIT システムに組み込む．test クラスを実行し，標準出力結果を確認する．

(4) 試験条件

- Test クラスとして，付録 A.2.5に定義するものを用いる．
- test クラスとして，以下に定義するものを用いる．

```
class test {
    public static void main(String argv[]) { test(); }
    static void test() {
        m(0);         m(0L);
        m(0F);        m(0D);
    }
    static void m(int x) {}
    static void m(long x) {}
    static void m(float x) {}
    static void m(double x) {}
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. OpenJIT/Test.java をコンパイルする．

2. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．

3. 環境変数 OpenJIT_COMPILER を OpenJIT/Test に設定する．

4. java test を実行する．

(6) 試験結果

試験項目: メソッド引数展開試験 (3)	(7) 合否判定

(1) 試験目的，試験内容

メソッド呼び出しがバックエンド中間コードに展開できることを確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 帰り値を持つメソッド呼び出しは，返り値を得るバックエンド中間コードが生成されていることが予想される．

確認方法 OpenJIT.Compile のサブクラスを Test を定義し，OpenJIT システムに組み込む．test クラスを実行し，標準出力結果を確認する．

(4) 試験条件

- Test クラスとして，付録 A.2.5に定義するものを用いる．
- test クラスとして，以下に定義するものを用いる．

```
class test {
    public static void main(String argv[]) { test(); }
    static void test() {
        mI();
        mJ();
        mF();
        mD();
    }
    static int mI() { return 0; }
    static long mJ() { return 0L; }
    static float mF() { return 0F; }
    static double mD() { return 0D; }
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. OpenJIT/Test.java をコンパイルする．

2. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．

3. 環境変数 OpenJIT_COMPILER を OpenJIT/Test に設定する．

4. java test を実行する．

(6) 試験結果

試験項目: 命令パターンマッチング試験 (1)	(7) 合否判定

(1) 試験目的，試験内容

論理値の否定を行うパターンが，最適化されたバックエンド中間コードに変換されることを確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 論理値の否定を行うパターンが，subcc と subx を使った最適化されたバックエンド中間コードに変換されることが予想される．

確認方法 OpenJIT.Compile のサブクラスを Test を定義し，OpenJIT システムに組み込む．test クラスを実行し，標準出力結果を確認する．

(4) 試験条件

- Test クラスとして，付録 A.2.5に定義するものを用いる．
- test クラスとして，以下に定義するものを用いる．

```
class test {
    static boolean b;
    public static void main(String argv[]) { test(); }
    static void test() {
        b = !b;
    }
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. OpenJIT/Test.java をコンパイルする．

2. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．

3. 環境変数 OpenJIT_COMPILER を OpenJIT/Test に設定する．

4. java test を実行する．

(6) 試験結果

試験項目: 命令パターンマッチング試験 (2)	(7) 合否判定

(1) 試験目的，試験内容

long data の比較・分岐パターンが，最適化されたバックエンド中間コードに変換されることを確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 long data の比較・分岐パターンが，最適化されたバックエンド中間コードに変換されることが予想される．バイトコードの lcmp 命令が複数の条件分岐命令に展開されると予想される．

確認方法 OpenJIT.Compile のサブクラスを Test を定義し，OpenJIT システムに組み込む．test クラスを実行し，標準出力結果を確認する．

(4) 試験条件

- Test クラスとして，付録 A.2.5に定義するものを用いる．
- test クラスとして，以下に定義するものを用いる．

```
class test {
    public static void main(String argv[]) { test(0L, 1L); }
    static boolean test(long x, long y) {
        return (x > y);
    }
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. OpenJIT/Test.java をコンパイルする．

2. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．

3. 環境変数 OpenJIT_COMPILER を OpenJIT/Test に設定する．

4. java test を実行する．

(6) 試験結果

試験項目: 命令パターンマッチング試験 (2)	(7) 合否判定

(1) 試験目的，試験内容

float data の比較・分岐パターンが，最適化されたバックエンド中間コードに変換されることを確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 float data の比較・分岐パターンが，最適化されたバックエンド中間コードに変換されることが予想される．バイトコード fcmpl 命令が fcmp と条件分岐命令に展開されると予想される．

確認方法 OpenJIT.Compile のサブクラスを Test を定義し，OpenJIT システムに組み込む．test クラスを実行し，標準出力結果を確認する．

(4) 試験条件

- Test クラスとして，付録 A.2.5に定義するものを用いる．
- test クラスとして，以下に定義するものを用いる．

```
class test {
    public static void main(String argv[]) { test(0F, 1F); }
    static boolean test(float x, float y) {
        return (x > y);
    }
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. OpenJIT/Test.java をコンパイルする．
2. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．
3. 環境変数 OpenJIT_COMPILER を OpenJIT/Test に設定する．
4. java test を実行する．

(6) 試験結果

試験項目: 命令パターンマッチング試験 (2)	(7) 合否判定

(1) 試験目的，試験内容

double data の比較・分岐パターンが，最適化されたバックエンド中間コードに変換されることを確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 double data の比較・分岐パターンが，最適化されたバックエンド中間コードに変換されることが予想される．バイトコード dcmpl 命令が fcmp と条件分岐命令に展開されると予想される．

確認方法 OpenJIT.Compile のサブクラスを Test を定義し，OpenJIT システムに組み込む．test クラスを実行し，標準出力結果を確認する．

(4) 試験条件

- Test クラスとして，付録 A.2.5に定義するものを用いる．
- test クラスとして，以下に定義するものを用いる．

```
class test {
    public static void main(String argv[]) { test(0D, 1D); }
    static boolean test(double x, double y) {
        return (x > y);
    }
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. OpenJIT/Test.java をコンパイルする．

2. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．

3. 環境変数 OpenJIT_COMPILER を OpenJIT/Test に設定する．

4. java test を実行する．

(6) 試験結果

5.2.3 OpenJIT RTL 変換機能

試験項目: 基本ブロック分割機能試験 (1)	(7) 合否判定

(1) 試験目的，試験内容

条件分岐命令を持つバイトコードから基本ブロック情報が得られることを確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 バイトコードから基本ブロック情報が得られることが予想される．

確認方法 OpenJIT のデバッグオプションとコンパイルメソッド選択オプションを用いて test クラスを実行し，その標準エラー出力を確認する．出力の "RTL bb beg: 基本ブロック開始番地 end: 基本ブロックの終了番地 " の表示が基本ブロック情報を示し，それが正しいことを確認する．

(4) 試験条件

test クラスとして，以下に定義するものを用いる．

```
class test {
    public static void main(String argv[]) { test(true); }
    static int test(boolean cond) {
        int i = 0;
        if (cond) { i = 1; };
        return i;
    }
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．

2. java -Dcompile.debug=1 -Dcompile.enable=.test test を実行する．

(6) 試験結果

試験項目: 基本ブロック分割機能試験 (2)	(7) 合否判定

(1) 試験目的，試験内容

tableswitch 命令 (多方向分岐) を持つバイトコードから基本ブロック情報が得られることを確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 バイトコードから基本ブロック情報が得られることが予想される．

確認方法 OpenJIT のデバッグオプションとコンパイルメソッド選択オプションを用いて test クラスを実行し，その標準エラー出力を確認する．出力の "RTL bb beg: 基本ブロック開始番地 end: 基本ブロックの終了番地 " の表示が基本ブロック情報を示し，それが正しいことを確認する．

(4) 試験条件

test クラスとして，以下に定義するものを用いる．

```
class test {
    public static void main(String argv[]) { test(10); }
    static int test(int cond) {
        int i = 0;
        switch(cond) {
        case 11: i++;
        case 12: i+=2;
        case 13: i+=3;
            break;
        default:
            --i;
        }
        return i;
    }
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．

2. java -Dcompile.debug=1 -Dcompile.enable=.test test を実行する．

(6) 試験結果

試験項目: 基本ブロック分割機能試験 (3)	(7) 合否判定

(1) 試験目的，試験内容

lookupswitch 命令 (多方向分岐) を持つバイトコードから基本ブロック情報が得られることを確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 バイトコードから基本ブロック情報が得られることが予想される．

確認方法 OpenJIT のデバッグオプションとコンパイルメソッド選択オプションを用いて test クラスを実行し，その標準エラー出力を確認する．出力の "RTL bb beg: 基本ブロック開始番地 end: 基本ブロックの終了番地 " の表示が基本ブロック情報を示し，それが正しいことを確認する．

(4) 試験条件

test クラスとして，以下に定義するものを用いる．

```
class test {
    public static void main(String argv[]) { test(10); }
    static int test(int cond) {
        int i = 0;
        switch(cond) {
        case 1:   i++;
        case 10:  i+=2;
        case 100: i+=3;
            break;
        default:
            --i;
        }
        return i;
    }
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．

2. `java -Dcompile.debug=1 -Dcompile.enable=.test test` を実行する．

(6) 試験結果

試験項目: 基本ブロック分割機能試験 (4)	(7) 合否判定

(1) 試験目的，試験内容

例外処理を持つバイトコードから基本ブロック情報が得られることを確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 バイトコードから基本ブロック情報が得られることが予想される．

確認方法 OpenJIT のデバッグオプションとコンパイルメソッド選択オプションを用いて test クラスを実行し，その標準エラー出力を確認する．出力の "RTL bb beg: 基本ブロック開始番地 end: 基本ブロックの終了番地 " の表示が基本ブロック情報を示し，それが正しいことを確認する．

(4) 試験条件

test クラスとして，以下に定義するものを用いる．

```
class test {
    public static void main(String argv[]) { test(); }
    static int test() {
        int ret = 0;
        try{
            ++ret;
        } catch(Error e) { ret = 1; }
        return ret;
    }
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．

2. java -Dcompile.debug=1 -Dcompile.enable=.test test を実行する．

(6) 試験結果

試験項目: 基本ブロック分割機能試験 (5)	(7) 合否判定

(1) 試験目的，試験内容

jsr,ret 命令を持つバイトコードから基本ブロック情報が得られることを確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 バイトコードから基本ブロック情報が得られることが予想される．

確認方法 OpenJIT のデバッグオプションとコンパイルメソッド選択オプションを用いて test クラスを実行し，その標準エラー出力を確認する．出力の "RTL bb beg: 基本ブロック開始番地 end: 基本ブロックの終了番地 " の表示が基本ブロック情報を示し，それが正しいことを確認する．

(4) 試験条件

test クラスとして，以下に定義するものを用いる．

```
class test {
    public static void main(String argv[]) { test(); }
    static int test() {
        int ret = 0;
        try{
            ++ret;
        } finally {
            ret += 2;
        }
        return ret;
    }
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．

2. java -Dcompile.debug=1 -Dcompile.enable=.test test を実行する．

(6) 試験結果

試験項目: RTL 変換機能試験	(7) 合否判定

(1) 試験目的，試験内容

OpenJIT RTL 変換機能がバックエンド中間コードから RTL に変換できることを確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 OpenJIT RTL 変換機能がバックエンド中間コードから RTL に変換できることが予想される．

確認方法 OpenJIT.Compile のサブクラスを Test を定義し，OpenJIT システムに組み込む．test クラスを実行し，標準出力にバックエンド中間コードの情報が出力される．OpenJIT のデバッグオプションとコンパイルメソッド選択オプションを用いて test クラスを実行し，その標準エラーに変換された RTL の情報が出力される．この 2 つの結果を比較し，正しく変換されたことを確認する．%s で始まるオペランドの数字が変更されることが予想される．

(4) 試験条件

- Test クラスとして，付録 A.2.5に定義するものを用いる．
- test クラスとして，以下に定義するものを用いる．

```
class test {
    public static void main(String argv[]) { test(); }
    static int m(int a, int b, int c, int d) { return a+b+c+d; }
    static int test() {
        return m(1,2,3,m(4,5,6,7));
    }
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. OpenJIT/Test.java をコンパイルする．
2. 環境変数 CLASSPATH，JAVA_COMPILER を設定する．
3. 環境変数 OpenJIT_COMPILER を OpenJIT/Test に設定する．
4. java -Dcompile.debug=1 -Dcompile.enable=.test test を実行する．
5. 環境変数 OpenJIT_COMPILER の設定を解除する．
6. java -Dcompile.debug=1 -Dcompile.enable=.test test を実行する．

(6) 試験結果

試験項目: コントロールフロー解析試験	(7) 合否判定

(1) 試験目的，試験内容

コントロールフロー解析ができていることを確認する．すなわち，各基本ブロックにおいて，処理が開始するときの Java のスタックの深さが正しく与えられており，OpenJIT RTL 変換機能がバックエンド中間コードから RTL に変換できることを確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 OpenJIT RTL 変換機能がバックエンド中間コードから RTL に変換できることが予想される．

確認方法 OpenJIT.Compile のサブクラスを Test を定義し，OpenJIT システムに組み込む．test クラスを実行し，標準出力にバックエンド中間コードの情報が出力される．OpenJIT のデバッグオプションとコンパイルメソッド選択オプションを用いて test クラスを実行し，その標準エラーに変換された RTL の情報が出力される．この 2 つの結果を比較し，正しく変換されたことを確認する．%s で始まるオペランドの数字が正しく変更されることが予想される．

(4) 試験条件

- Test クラスとして，付録 A.2.5に定義するものを用いる．
- test クラスとして，付録 A.2.6に定義するものを用いる．

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. OpenJIT/Test.java をコンパイルする．
2. 環境変数 CLASSPATH，JAVA_COMPILER を設定する．
3. 環境変数 OpenJIT_COMPILER を OpenJIT/Test に設定する．
4. java -Dcompile.debug=1 -Dcompile.enable=.test test を実行する．
5. 環境変数 OpenJIT_COMPILER の設定を解除する．
6. java -Dcompile.debug=1 -Dcompile.enable=.test test を実行する．

(6) 試験結果

試験項目: オペランドの型の決定試験	(7) 合否判定

(1) 試験目的，試験内容

バックエンド中間コードのオペランドの型が未定義なものから，RTL に変換できることを確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 バイトコード命令の dup 命令を RTL に変換した命令のオペランドの型が正しく変換できていることが予想される．

確認方法 OpenJIT のデバッグオプションとコンパイルメソッド選択オプションを用いて test クラスを実行し，その標準エラー出力を確認する．バイトコード命令の dup 命令が変換された RTL の型が正しいことを確認する．

(4) 試験条件

test クラスとして，以下に定義するものを用いる．

```
class test {
    public static void main(String argv[]) { test(); }
    public test() { }
    public void m() { }
    static void test() {
        (new test()).m();
    }
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．

2. java -Dcompile.debug=1 -Dcompile.enable=.test test を実行する．

(6) 試験結果

5.2.4 OpenJIT Peephole 最適化機能

試験項目: データフロー解析試験 (1)	(7) 合否判定

(1) 試験目的，試験内容

データ (定数) の流れが正しく解析できていることを確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 データの流れが正しく解析できていることが予想される．バイトコードの invokestatic 命令の引数が正しく最適化されて変換されていることが予想される．

確認方法 OpenJIT のデバッグオプションとコンパイルメソッド選択オプションを用いて test クラスを実行し，その標準エラー出力を確認する．

(4) 試験条件

test クラスとして，以下に定義するものを用いる．

```
class test {
    public static void main(String argv[]) { test(); }
    static void m(int a, int b, int c, int d) {}
    static void test() {
        m(1,2,3,4);
        m(4,3,2,1);
    }
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．

2. `java -Dcompile.debug=2 -Dcompile.enable=.test test` を実行する．

(6) 試験結果

試験項目: データフロー解析試験 (2)	(7) 合否判定

(1) 試験目的，試験内容

データ (ローカル変数) の流れが正しく解析できていることを確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 データの流れが正しく解析できていることが予想される．バイトコードの invokestatic 命令の引数が正しく最適化されて変換されていることが予想される．

確認方法 OpenJIT のデバッグオプションとコンパイルメソッド選択オプションを用いて test クラスを実行し，その標準エラー出力を確認する．

(4) 試験条件

test クラスとして，以下に定義するものを用いる．

```
class test {
    public static void main(String argv[]) { test(1,2,3,4); }
    static void m(int a, int b, int c, int d) {}
    static void test(int a, int b, int c, int d) {
        m(a,b,c,d);
        m(d,c,b,a);
    }
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．

2. java -Dcompile.debug=2 -Dcompile.enable=.test test を実行する．

(6) 試験結果

試験項目: Peephole 最適化試験 (1)	(7) 合否判定

(1) 試験目的，試験内容

演算を行った結果をローカル変数へ代入する処理が最適化できていることを確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 iadd 命令が変換された RTL のデスティネーション・オペランドがローカル変数 (%v) に置き換わっていて，その後の istore 命令が消去されていることが予想される．

確認方法 OpenJIT のデバッグオプションとコンパイルメソッド選択オプションを用いて test クラスを実行し，その標準エラー出力を確認する．

(4) 試験条件

test クラスとして，以下に定義するものを用いる．

```
class test {
    public static void main(String argv[]) { test(1,2); }
    static int test(int a, int b) {
        int x = a+b;
        return x;
    }
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．

2. `java -Dcompile.debug=2 -Dcompile.enable=.test test` を実行する．

(6) 試験結果

試験項目: Peephole 最適化試験 (2)	(7) 合否判定

(1) 試験目的，試験内容

演算を行った結果をパラメタ変数へ代入する処理が最適化できていることを確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 演算命令 (iadd,isub,imul,idiv) が変換された RTL のデスティネーション・オペランドがパラメタ変数 (%r) に置き換わっていて，その後の invokestatic 命令のパラメタを設定する move 命令が消去されていることが予想される．

確認方法 OpenJIT のデバッグオプションとコンパイルメソッド選択オプションを用いて test クラスを実行し，その標準エラー出力を確認する．

(4) 試験条件

test クラスとして，以下に定義するものを用いる．

```
class test {
    public static void main(String argv[]) { test(1,2); }
    static void m(int a, int b, int c, int d) { }
    static void test(int a, int b) {
        m(a+b, a-b, a*b, a/b);
    }
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．

2. `java -Dcompile.debug=2 -Dcompile.enable=.test test` を実行する．

(6) 試験結果

試験項目: Constant Folding 最適化試験 (1)	(7) 合否判定

(1) 試験目的，試験内容

Constant Folding 最適化により右論理シフトの最適化ができていることを確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 sll 命令が削除されていることが予想される．

確認方法 OpenJIT のデバッグオプションとコンパイルメソッド選択オプションを用いて test クラスを実行し，その標準エラー出力を確認する．バイトコードの castore 命令が RTL に変換された RTL 命令列から sll 命令が消去 (xxx) されていることを確認する．

(4) 試験条件

test クラスとして，以下に定義するものを用いる．

```
class test {
    public static void main(String argv[]) { test(); }
    static char[] test() {
        char x[] = {'a'};
        return x;
    }
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．

2. java -Dcompile.debug=2 -Dcompile.enable=.test test を実行する．

(6) 試験結果

試験項目: Constant Folding 最適化試験 (2)	(7) 合否判定

(1) 試験目的，試験内容

Constant Folding 最適化により 0 との論理積の最適化ができていることを確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 and 命令が削除されていることが予想される．

確認方法 OpenJIT のデバッグオプションとコンパイルメソッド選択オプションを用いて test クラスを実行し，その標準エラー出力を確認する．バイトコードの land 命令が RTL に変換された RTL 命令列から and 命令が消去 (xxx) されていることを確認する．

(4) 試験条件

test クラスとして，以下に定義するものを用いる．

```
class test {
    public static void main(String argv[]) { test(1L<<32); }
    static int test(long x) {
        return (int)(x & 0xffff);
    }
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．

2. java -Dcompile.debug=2 -Dcompile.enable=.test test を実行する．

(6) 試験結果

試験項目: Constant Folding 最適化試験 (3)	(7) 合否判定

(1) 試験目的，試験内容

Constant Folding 最適化により -1(all 1) との論理積の最適化ができていることを確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 and 命令が -1 の転送命令に変更され，さらに消去されていることが予想される．

確認方法 OpenJIT のデバッグオプションとコンパイルメソッド選択オプションを用いて test クラスを実行し，その標準エラー出力を確認する．バイトコードの iand 命令が RTL に変換された RTL 命令列から and 命令が消去 (xxx) されていることを確認する．

(4) 試験条件

test クラスとして，以下に定義するものを用いる．

```
class test {
    public static void main(String argv[]) { test(1); }
    static int test(int x) {
        return (x & -1);
    }
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．

2. java -Dcompile.debug=2 -Dcompile.enable=.test test を実行する．

(6) 試験結果

試験項目: ハンドル参照除去最適化試験	(7) 合否判定

(1) 試験目的，試験内容

ハンドルを介したオブジェクト間接参照の load 命令の削除の最適化ができていることを確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 生成される RTL の先頭にハンドル参照を行う ld 命令が付加され，バイトコードの getfield 命令の ld 命令が削除されていることが予想される．

確認方法 OpenJIT のデバッグオプションとコンパイルメソッド選択オプションを用いて test クラスを実行し，その標準エラー出力を確認する．

(4) 試験条件

test クラスとして，以下に定義するものを用いる．

```
class test {
    int x,y;
    public static void main(String argv[]) { (new test()).test(); }
    int test() {
        return (this.x + this.y);
    }
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．

2. java -Dcompile.debug=2 -Dcompile.enable=.test test を実行する．

(6) 試験結果

5.2.5 OpenJIT レジスタ割付機能

試験項目: 整数レジスタ割付試験	(7) 合否判定

(1) 試験目的，試験内容

整数レジスタ割り付け機能を試験する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 整数レジスタの割り付けがされ，割り付けらなかったレジスタに一時レジスタが割り当てられることが予想される．

確認方法 OpenJIT のデバッグオプションとコンパイルメソッド選択オプションを用いて test クラスを実行し，その標準エラー出力を確認する．仮想レジスタ番号 13 までが物理レジスタに割り付けられ，仮想レジスタ番号 14 以上のものは，一時レジスタを割り付けたメッセージ "assign:　仮想レジスタ番号 = 物理レジスタ番号" と表示される．

(4) 試験条件

test クラスとして，以下に定義するものを用いる．

```
class test {
    public static void main(String argv[]) { test(); }
    public static void test() {
        int i1,i2,i3,i4,i5,i6,i7,i8,i9,i10;
        int i11,i12,i13,i14,i15,i16;

        i1=1; i2=2; i3=3; i4=4; i5=5; i6=6; i7=7; i8=8;
        i9=9; i10=10; i11=11; i12=12; i13=13; i14=14; i15=15; i16=16;
    }
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．

2. `java -Dcompile.debug=2 -Dcompile.enable=.test test` を実行する．

(6) 試験結果

試験項目: 浮動小数レジスタ割付試験	(7) 合否判定

(1) 試験目的，試験内容

浮動小数レジスタ割り付け機能を試験する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 浮動小数レジスタは全てが一時レジスタとして割り当てられ，物理レジスタとして 15 個しかないので，16 個以上の仮想レジスタを割り当てるとスピルコードが生成されることが予想される．

確認方法 OpenJIT のデバッグオプションとコンパイルメソッド選択オプションを用いて test クラスを実行し，その標準エラー出力を確認する．一時レジスタの割り当ては "assign:　仮想レジスタ番号 = 物理レジスタ番号" と表示される．また，スピルした場合 "spill:　物理レジスタ番号" と表示される．

(4) 試験条件

test クラスとして，以下に定義するものを用いる．

```
class test {
    public static void main(String argv[]) { test(); }
    public static void test() {
        float i1,i2,i3,i4,i5,i6,i7,i8,i9,i10;
        float i11,i12,i13,i14,i15,i16;

        i1=1F;i2=2F;i3=3F;i4=4F;i5=5F;i6=6F;i7=7F;i8=8F;
        i9=9F;i10=10F;i11=11F;i12=12F;i13=13F;i14=14F;i15=15F;i16=16F;
    }
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．

2. `java -Dcompile.debug=2 -Dcompile.enable=.test test` を実行する．

(6) 試験結果

試験項目: 整数物理レジスタ割付試験	(7) 合否判定

(1) 試験目的，試験内容

整数物理レジスタの割り付けを確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 整数物理レジスタの割り付けが，Sparc レジスタの %l0 ~ %l7,%i0 ~ %i5 の順で割り付けられていることが予想される．

確認方法 デバッガ (gdb) のもとで，OpenJIT のデバッグオプションとコンパイルメソッド選択オプションを用いて test クラスを実行する．そして，ブレークポイントを設定して再実行させ，コンパイルされたネイティブコードの逆アセンブラリストを確認する．

(4) 試験条件

test クラスとして，以下に定義するものを用いる．

```
class test {
    public static void main(String argv[]) { test(); }
    public static void test() {
        int i1,i2,i3,i4,i5,i6,i7,i8,i9,i10;
        int i11,i12,i13,i14;

        i1=1; i2=2; i3=3; i4=4; i5=5; i6=6; i7=7; i8=8;
        i9=9; i10=10; i11=11; i12=12; i13=13; i14=14;
    }
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．
2. gdb を起動する．
3. run -Dcompile.debug=2 -Dcompile.enable=.test test を実行する．
4. ファイル api.c の OpenJIT_compile 関数の do_execute_java_method_vararg を呼び出した後の行にブレークポイントを設定する．
5. run を実行してブレークポイントで停止するのを待つ．
6. disassemble mb->CompiledCode
   mb->CompiledCode+(int)mb->CompiledCodeI nfo を実行する．

(6) 試験結果

試験項目: 浮動小数物理レジスタ割付試験	(7) 合否判定

(1) 試験目的，試験内容

浮動小数物理レジスタの割り付けを確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 浮動小数物理レジスタの割り付けが，Sparc レジスタの %f2 ~ %f30 の順で割り付けられていることが予想される．

確認方法 デバッガ (gdb) のもとで，OpenJIT のデバッグオプションとコンパイルメソッド選択オプションを用いて test クラスを実行する．そして，ブレークポイントを設定して再実行させ，コンパイルされたネイティブコードの逆アセンブラリストを確認する．

(4) 試験条件

test クラスとして，以下に定義するものを用いる．

```
class test {
    public static void main(String argv[]) { test(); }
    public static void test() {
        float i1,i2,i3,i4,i5,i6,i7,i8,i9,i10;
        float i11,i12,i13,i14,i15;

        i1=0F; i2=1F; i3=2F; i4=0F; i5=1F; i6=2F; i7=0F; i8=1F;
        i9=2F; i10=0F; i11=1F; i12=2F; i13=0F; i14=1F; i15=2F;
    }
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．
2. gdb を起動する．
3. run -Dcompile.debug=2 -Dcompile.enable=.test test を実行する．
4. ファイル api.c の OpenJIT_compile 関数の do_execute_java_method_vararg を呼び出した後の行にブレークポイントを設定する．
5. run を実行してブレークポイントで停止するのを待つ．
6. disassemble mb->CompiledCode mb->CompiledCode + (int) mb->CompiledCo deInfo を実行する．

(6) 試験結果

試験項目: スピルコード生成試験	(7) 合否判定

(1) 試験目的，試験内容

スピルコードが生成されることを確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 19 個の仮想レジスタの値を変更すれば，19 番目の仮想レジスタの値の変更を行う命令に対してスピルコード (%sp をベースとした st 命令) が生成されることが予想される．

確認方法 デバッガ (gdb) のもとで，OpenJIT のデバッグオプションとコンパイルメソッド選択オプションを用いて test クラスを実行する．そして，ブレークポイントを設定して再実行させ，コンパイルされたネイティブコードの逆アセンブラリストを確認する．

(4) 試験条件

test クラスとして，以下に定義するものを用いる．

```
class test {
    public static void main(String argv[]) { test(); }
    public static void test() {
        int i1,i2,i3,i4,i5,i6,i7,i8,i9,i10;
        int i11,i12,i13,i14,i15,i16,i17,i18;
        int i19;

        i1=1; i2=2; i3=3; i4=4; i5=5; i6=6; i7=7; i8=8;
        i9=9; i10=10; i11=11; i12=12; i13=13; i14=14; i15=15; i16=16;
        i17=17; i18=18; i19 = 19;
    }
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．
2. gdb を起動する．
3. run -Dcompile.debug=2 -Dcompile.enable=.test test を実行する．
4. ファイル api.c の OpenJIT_compile 関数の do_execute_java_method_vararg を呼び出した後の行にブレークポイントを設定する．
5. run を実行してブレークポイントで停止するのを待つ．
6. disassemble mb->CompiledCode mb->CompiledCode + (int) mb->CompiledCo deInfo を実行する．

(6) 試験結果

試験項目: フィルコード生成試験	(7) 合否判定

(1) 試験目的，試験内容

フィルコードが生成されることを確認する．

(2) 試験データの内容

特になし．

(3) 予想結果及び確認方法

予想結果 スピルした仮想レジスタの値を読めば，フィルコード (%sp をベースとした ld 命令) が生成されることが予想される．

確認方法 デバッガ (gdb) のもとで，OpenJIT のデバッグオプションとコンパイルメソッド選択オプションを用いて test クラスを実行する．そして，ブレークポイントを設定して再実行させ，コンパイルされたネイティブコードの逆アセンブラリストを確認する．

(4) 試験条件

test クラスとして，以下に定義するものを用いる．

```
class test {
    public static void main(String argv[]) { test(); }
    public static void test() {
        int i1,i2,i3,i4,i5,i6,i7,i8,i9,i10;
        int i11,i12,i13,i14,i15,i16,i17,i18;
        int i19;

        i1=1; i2=2; i3=3; i4=4; i5=5; i6=6; i7=7; i8=8;
        i9=9; i10=10; i11=11; i12=12; i13=13; i14=14; i15=15; i16=16;
        i17=17; i18=18; i19 = 19;
        i1=i15;
    }
}
```

(次ページへ続く)

(前ページからの続き)

(5) 試験手順

試験手順は以下の通りである．

1. 環境変数 CLASSPATH, JAVA_COMPILER を設定する．
2. gdb を起動する．
3. run -Dcompile.debug=2 -Dcompile.enable=.test test を実行する．
4. ファイル api.c の OpenJIT_compile 関数の do_execute_java_method_vararg を呼び出した後の行にブレークポイントを設定する．
5. run を実行してブレークポイントで停止するのを待つ．
6. disassemble mb->CompiledCode mb->CompiledCode + (int) mb->CompiledCo deInfo を実行する．

(6) 試験結果

付録 A

試験方法補足

A.1 OpenJIT フロントエンドシステム

A.1.1 ディスコンパイル機能用テストドライバ

ディスコンパイル機能の試験で用いられるテストドライバは，次の A.1.1.1，A.1.1.2，A.1.1.3，A.1.1.4，A.1.1.5で定義されるクラス群で構成されている．

A.1.1.1 OpenJIT.frontend.discompiler.driver.Constants クラス

```
package OpenJIT.frontend.discompiler.driver;

public interface Constants {
    public static final int BYTECODE_PARSER = 0;
    public static final int CONTROL_FLOW_GRAPH = 1;
    public static final int BASICBLOCK_ANALYZER = 2;
    public static final int EXPRESSION_ANALYZER = 3;
    public static final int DOMINATOR_TREE = 4;
    public static final int STRUCTURED_CFG = 5;
    public static final int DISCOMPILED_AST = 6;
};
```

A.1.1.2 OpenJIT.frontend.discompiler.driver.StandaloneDiscompiler クラス

```
package OpenJIT.frontend.discompiler.driver;
```

```
import OpenJIT.frontend.classfile.*;
import OpenJIT.frontend.discompiler.*;
import OpenJIT.frontend.util.IndentedPrintStream;
import java.io.FileInputStream;
import java.io.InputStream;
import java.io.IOException;
import java.io.PrintStream;

public class StandaloneDiscompiler extends ClassFile implements Constants {
    public StandaloneDiscompiler(InputStream is) throws IOException {
        super(is);
    }

    private String canonClassName(String name) {
        return name.replace('/', '.');
    }

    public void print(PrintStream out, int level) {
        IndentedPrintStream iout = new IndentedPrintStream(out, 4);
        iout.println(AccessFlag.toString(accessFlags));
        StringBuffer buf = new StringBuffer();
        String thisClassName = constantPool.resolveClassName(thisClass);
        buf.append("class ").append(canonClassName(thisClassName))
            .append(" extends ")
            .append(canonClassName(constantPool.resolveClassName(superClass)));
        if (interfaces.length > 0) {
            buf.append(" implements ")
                .append(canonClassName(constantPool
                    .resolveClassName(interfaces[0])));
            for (int i = 1, count = interfaces.length; i < count; i++)
                buf.append(", ")
                    .append(canonClassName(constantPool
                        .resolveClassName(interfaces[i])));
        }
        buf.append(" {");
        iout.println(buf.toString());
```

```
iout.inc();
for (int i = 0, count = fields.length; i < count; i++) {
    FieldInfo field = fields[i];
    StringBuffer sbuf = new StringBuffer();
    sbuf.append(AccessFlag.toString(field.accessFlags()))
        .append(NameAndType.toString(((ConstantUTF8)constantPool
            .itemAt(field.descriptorIndex())).bytes()))
        .append(" ")
        .append(constantPool.itemAt(field.nameIndex()).toString());
    ConstantValueAttribute attr
        = (ConstantValueAttribute)field.attributes()
        .lookup(Attributes.CONSTANTVALUE);
    if (attr != null) {
        sbuf.append(" = ");
        int index = attr.constantValueIndex();
        ConstantPoolItem constant = constantPool.itemAt(index);
        if (constant.isString()) {
            sbuf.append("\"")
                .append(constantPool.resolveString(index))
                .append("\"");
        } else
            sbuf.append(constant.toString());
    }
    sbuf.append(";");
    iout.println(sbuf.toString());
}
TestDiscompiler discompiler[] = new TestDiscompiler[methods.length];
for (int i = 0, count = methods.length; i < count; i++) {
    MethodInfo method = methods[i];
    StringBuffer sbuf = new StringBuffer();
    sbuf.append(AccessFlag.toString(method.accessFlags()));
    String name = constantPool.itemAt(method.nameIndex()).toString();
    ConstantUTF8 descriptor
        = (ConstantUTF8)constantPool.itemAt(method.descriptorIndex());
    try {
        NameAndType sig
            = new NameAndType(null, name, descriptor.bytes());
```

```
        sbuf.append(sig.toString());
    } catch (ArrayIndexOutOfBoundsException e) {
        System.err.println("cannot parse: " + descriptor);
        throw e;
    }
    if (method.attributes().lookup(Attributes.CODE) != null) {
        sbuf.append(" {");
        iout.println(sbuf.toString());
        iout.inc();
        StandaloneMethodInformation methodInfo
            = new StandaloneMethodInformation(method);
        discompiler[i] = new TestDiscompiler(methodInfo);
        switch (level) {
        case BYTECODE_PARSER:
            discompiler[i].printBytecode(iout);
            break;
        case CONTROL_FLOW_GRAPH:
            discompiler[i].printCFG(iout);
            break;
        case BASICBLOCK_ANALYZER:
            discompiler[i].printBBACFG(iout);
            break;
        case EXPRESSION_ANALYZER:
            discompiler[i].printEACFG(iout);
            break;
        case DOMINATOR_TREE:
            discompiler[i].printDT(iout);
            break;
        case STRUCTURED_CFG:
            discompiler[i].printSCFG(iout);
            break;
        case DISCOMPILED_AST:
            discompiler[i].printAST(iout);
            break;
    }
        iout.dec();
        iout.println("}");
```

```
            } else {
                sbuf.append(";");
                iout.println(sbuf.toString());
            }
        }
        iout.dec();
        iout.println("}");
    }
}
```

A.1.1.3 OpenJIT.frontend.discompiler.driver.StandaloneMethodInformation クラス

```
package OpenJIT.frontend.discompiler.driver;

import OpenJIT.Constants;
import OpenJIT.ExceptionHandler;
import OpenJIT.frontend.classfile.*;
import OpenJIT.frontend.discompiler.*;
import OpenJIT.frontend.util.IntKeyHashtable;

public class StandaloneMethodInformation implements MethodInformation, Constants {
    private MethodInfo method;
    private CodeAttribute code;
    private ConstantPool constantPool;
    private byte[] bytecode;

    public StandaloneMethodInformation(MethodInfo method) {
        this.method = method;
        this.constantPool = method.classFile().constantPool();
        code = (CodeAttribute)method.attributes().lookup(Attributes.CODE);
        bytecode = code.code();
    }

    /**
     * Returns the number of local variables.
     */
```

```
public int nlocals() {
    return code.maxLocals();
}

/**
 * Returns whether the method is static one.
 */
public boolean isStatic() {
    return (method.accessFlags() & ACC_STATIC) != 0;
}

private String thisClassName;
/**
 * Returns a String of the class name of the discompiled method belongs to.
 */
public synchronized String thisClassName() {
    if (thisClassName != null)
        return thisClassName;
    thisClassName = method.classFile().thisClassName();
    return thisClassName;
}

/**
 * Returns the name of the class indeced by index into ConstantPool.
 */
public String className(int index) {
    return constantPool.resolveString(index);
}

/**
 * Returns the width of the field/method indeced by given index into
 * ConstantPool.
 */
public int fieldWidth(int index) {
    NameAndType sig = nameAndType(index);
    switch (sig.type()[0]) {
    case SIGC_LONG:
```

```
    case SIGC_DOUBLE:
        return 2;
    default:
        return 1;
    }
}

/**
 * Returns the name and type information of method/field reference
 * indeced by index into ConstantPool.
 */
private IntKeyHashtable nameAndTypes = new IntKeyHashtable();
public NameAndType nameAndType(int index) {
    NameAndType result = (NameAndType)nameAndTypes.get(index);
    if (result != null)
        return result;
    String className = constantPool.resolveClassName(index);
    String name = constantPool.resolveMemberName(index);
    byte descriptor[] = constantPool.resolveMemberDescriptor(index);
    result = new NameAndType(className, name, descriptor);
    nameAndTypes.put(index, result);
    return result;
}

/**
 * Returns the kind of ConstantPool entry indeced by given index.
 */
public int kindOfConstant(int index) {
    return constantPool.itemAt(index).tag();
}

/**
 * Returns an int value of ConstantPool indeced by given index.
 */
public int constantInt(int index) {
    return constantPool.resolveInt(index);
}
```

```
/**
 * Returns an float value of ConstantPool indeced by given index.
 */
public float constantFloat(int index) {
    return constantPool.resolveFloat(index);
}

/**
 * Returns an String value of ConstantPool indeced by given index.
 */
public String constantString(int index) {
    return constantPool.resolveString(index);
}

/**
 * Returns an double value of ConstantPool indeced by given index.
 */
public double constantDouble(int index) {
    return constantPool.resolveDouble(index);
}

/**
 * Returns an long value of ConstantPool indeced by given index.
 */
public long constantLong(int index) {
    return constantPool.resolveLong(index);
}

/**
 * Returns the array of ExceptionHandler.
 */
public ExceptionHandler[] exceptionHandler() {
    return code.exceptionTable();
}

/**
```

```
 * Returns the length of bytecode[].
 */
public int bytecodeLength() {
    return bytecode.length;
}

/**
 * Returns (signed)byte-value at pc of bytecode[].
 */
public int byteAt(int pc) {
    return bytecode[pc];
}

/**
 * Returns (unsigned)byte-value at pc of bytecode[].
 */
public int unsignedByteAt(int pc) {
    return bytecode[pc] & 0xff;
}

/**
 * Returns (signed)short-value at pc of bytecode[].
 */
public int shortAt(int pc) {
    return (short)((bytecode[pc] << 8) + unsignedByteAt(pc + 1));
}

/**
 * Returns (unsigned)short-value at pc of bytecode[].
 */
public int unsignedShortAt(int pc) {
    return ((unsignedByteAt(pc) << 8) + unsignedByteAt(pc + 1));
}

/**
 * Returns int-value at pc of bytecode[].
 */
```

```
    public int intAt(int pc) {
        return ((shortAt(pc) << 16) + unsignedShortAt(pc + 2));
    }
}
```

A.1.1.4 OpenJIT.frontend.discompiler.driver.Test クラス

```
import OpenJIT.frontend.discompiler.driver.StandaloneDiscompiler;
import java.io.FileInputStream;
import java.io.IOException;

public class Test {
    public static void main(String args[]) {
        if (args.length > 1) {
            int level = Integer.parseInt(args[0]);
            try {
                FileInputStream fin = new FileInputStream(args[1]);
                StandaloneDiscompiler discompiler
                    = new StandaloneDiscompiler(fin);
                discompiler.print(System.out, level);
                fin.close();
            } catch (IOException e) {
                System.err.println("No such file: " + args[1]);
                System.exit(1);
            }
        } else
            usage();
    }

    static void usage() {
        System.err.println("usage: java Test level filename");
    }
}
```

A.1.1.5 OpenJIT.frontend.discompiler.driver.TestDiscompiler クラス

```
package OpenJIT.frontend.discompiler.driver;
```

```
import OpenJIT.frontend.discompiler.*;
import OpenJIT.frontend.tree.Node;
import OpenJIT.frontend.util.IndentedPrintStream;

public class TestDiscompiler extends Discompiler {
    public TestDiscompiler(MethodInformation method) {
        super(method);
    }

    public void printBytecode(IndentedPrintStream out) {
        bytecodeInfo.print(out);
    }

    public void printCFG(IndentedPrintStream out) {
        ControlFlowGraph cfg = new ControlFlowGraph(method, bytecodeInfo);
        cfg.createCFG();
        cfg.print(out);
    }

    public void printBBACFG(IndentedPrintStream out) {
        BasicBlockAnalyzer cfg = new BasicBlockAnalyzer(method, bytecodeInfo,
                                                        astFactory);
        cfg.createCFG();
        cfg.print(out);
    }

    public void printEACFG(IndentedPrintStream out) {
        structurelessCFG.print(out);
    }

    public void printDT(IndentedPrintStream out) {
        structurelessCFG.printTree(out);
    }

    public void printSCFG(IndentedPrintStream out) {
        ControlFlowAnalyzer scfg = new ControlFlowAnalyzer(structurelessCFG);
```

```
        scfg.print(out);
    }

    public void printAST(IndentedPrintStream out) {
        Node head = discompile();
        head.print(out.out);
    }
}
```

A.1.2 OpenJIT クラスファイルアノテーション解析機能

A.1.2.1 アノテーション解析部用テストドライバ

```
import OpenJIT.frontend.discompiler.*;
import java.io.*;

public class TestAnnotationAnalysis extends AnnotationAnalyzer {
    public static void main(String args[]) throws Exception {
if (args.length == 0)
    return;

Annotation annotation = new Annotation(args[0]);
ByteArrayOutputStream baos = new ByteArrayOutputStream();
ObjectOutputStream oos = new ObjectOutputStream(baos);
oos.writeObject(annotation);
oos.flush();
byte[] attribute = baos.toByteArray();

System.out.println(readAnnotation(attribute));
    }
}
```

A.1.2.2 アノテーション登録部用テストドライバ

```
import OpenJIT.frontend.discompiler.*;
import java.io.*;

public class TestAnnotationRegister extends AnnotationAnalyzer {
    public static void main(String args[]) throws Exception {
if (args.length == 0)
    return;
```

```
Annotation annotation = new Annotation(args[0]);

try {
    registerAnnotation(annotation);
} catch (DiscompilerError e) {
    System.out.println("registerAnnotation: fail");
    throw e;
}
System.out.println("registerAnnotation: success");
    }
}
```

A.1.2.3 メタクラス制御部用テストドライバ

```
import OpenJIT.frontend.discompiler.*;
import java.io.*;

public class TestMetaclass extends AnnotationAnalyzer {
    public static void main(String args[]) throws Exception {
if (args.length == 0)
    return;

Annotation annotation = new Annotation(args[0]);

System.out.println(addMetaobject(annotation));
    }
}
```

A.1.2.4 アノテーション解析部全体試験用テストドライバ

```
import OpenJIT.frontend.discompiler.*;
```

```
import java.io.*;

public class TestAll extends AnnotationAnalyzer {
    public static void main(String args[]) throws Exception {
if (args.length == 0)
    return;

Annotation annotation = new Annotation(args[0]);
ByteArrayOutputStream baos = new ByteArrayOutputStream();
ObjectOutputStream oos = new ObjectOutputStream(baos);
oos.writeObject(annotation);
oos.flush();
byte[] attribute = baos.toByteArray();

Annotation test = readAnnotation(attribute);
System.out.print("after analysis: ");
System.out.println(test);

try {
    registerAnnotation(test);
} catch (DiscompilerError e) {
    System.out.println("registerAnnotation: fail");
    throw e;
}
System.out.println("registerAnnotation: success");

System.out.print("after metaobject added: ");
System.out.println(addMetaobject(annotation));

annotation.metaobject.metaInvoke();
    }
```

```
}
```

A.1.3 最適化機能用テストドライバ

最適化制御部動作試験に用いられるテストドライバは，次のように定義される．

```
import OpenJIT.frontend.discompiler.ControlFlowGraph;
import OpenJIT.frontend.flowgraph.Optimizer;
import OpenJIT.frontend.tree.Node;

public class Test extends Optimizer {
    int debuglevel;

    public Test() {
        super(null);
        debuglevel = 0;
    }

    public byte[] optimize(byte bytecode[], Node ast, ControlFlowGraph cfg) {
        if (debuglevel != 2)
            System.out.println("optimize: ok");
        if (debuglevel != 1)
            generateBytecode();
        return null;
    }

    public byte[] generateBytecode() {
        System.out.println("generateBytecode: ok");
        return null;
    }

    public static void main(String args[]) {
        Test test = new Test();
        if (args.length != 1)
```

```
            return;
        if (args[0].equals("optimize"))
            test.debuglevel = 1;
        else if (args[0].equals("gen"))
            test.debuglevel = 2;
        else if (args[0].equals("all"))
            test.debuglevel = 3;
        test.optimize(null, null, null);
    }
}
```

A.1.4 プログラム変換機能用テストドライバ

A.1.4.1 AST 変換ルール登録部動作試験

AST 変換ルール登録部の試験で用いられるテストドライバは，次のように定義される．

```
import OpenJIT.frontend.tree.*;
import OpenJIT.frontend.flowgraph.*;
import java.util.Vector;
import java.util.Hashtable;
import java.util.Enumeration;

public class Test extends ASTTransformer {
    public Test() {
        super(null);
    }

    public static void main(String args[]) {
        Test test = new Test();
        test.test();
    }

    public void test() {
        Expression from = new IntExpression(0, 3);
        Expression to = new IntExpression(0, 7);
        registerRule(from, to);
        Enumeration keys = dst.keys();
        Enumeration elements = dst.elements();
        while (keys.hasMoreElements()) {
            System.out.print(keys.nextElement());
            System.out.print(" -> ");
            System.out.println(elements.nextElement());
```

```
        }
    }
}
```

A.1.4.2 AST パターンマッチ部動作試験 (1)

AST パターンマッチ部動作試験 (1) で用いられるテストドライバは，次のように定義される．

```
import OpenJIT.frontend.tree.*;
import OpenJIT.frontend.flowgraph.*;
import java.util.Vector;
import java.util.Hashtable;
import java.util.Enumeration;

public class Test extends ASTTransformer {
    public Test() {
        super(null);
    }

    public static void main(String args[]) {
        Test test = new Test();
        test.test();
    }

    public void test() {
        Expression from = new IntExpression(0, 3);
        Expression to = new IntExpression(0, 7);
        registerRule(from, to);
        Node result = match(from);
        if (result != null) {
            System.out.print(from);
```

```
            System.out.print(" matches with ");
            System.out.println(result);
        }
    }
}
```

A.1.4.3 AST パターンマッチ部動作試験 (2)

AST パターンマッチ部動作試験 (2) で用いられるテストドライバは，次のように定義される．

```
import OpenJIT.frontend.tree.*;
import OpenJIT.frontend.flowgraph.*;
import java.util.Vector;
import java.util.Hashtable;
import java.util.Enumeration;

public class Test extends ASTTransformer {
    public Test() {
        super(null);
    }

    public static void main(String args[]) {
        Test test = new Test();
        test.test();
    }

    public void test() {
        Expression from = new IntExpression(0, 3);
        Expression to = new IntExpression(0, 7);
        registerRule(from, to);
        Node result = match(to);
```

```
        if (result == null) {
            System.out.print(to);
            System.out.println(" doesn't match with any rules.");
        }
    }
}
```

A.1.4.4 AST 変換部動作試験 (1)

AST 変換部動作試験 (1) で用いられるテストドライバは，次のように定義される．

```
import OpenJIT.frontend.tree.*;
import OpenJIT.frontend.flowgraph.*;
import java.util.Vector;
import java.util.Hashtable;
import java.util.Enumeration;

public class Test extends ASTTransformer {
    public Test() {
        super(null);
    }

    public static void main(String args[]) {
        Test test = new Test();
        test.test();
    }

    public void test() {
        Expression from = new IntExpression(0, 3);
        Expression to = new IntExpression(0, 7);
        registerRule(from, to);
        Node result = transform(from);
```

```
        System.out.print(from);
        System.out.print(" is transformed to ");
        System.out.println(result);
    }
}
```

A.1.4.5 AST 変換部動作試験 (2)

AST 変換部動作試験 (2) で用いられるテストドライバは，次のように定義される．

```
import OpenJIT.frontend.tree.*;
import OpenJIT.frontend.flowgraph.*;
import java.util.Vector;
import java.util.Hashtable;
import java.util.Enumeration;

public class Test extends ASTTransformer {
    public Test() {
        super(null);
    }

    public static void main(String args[]) {
        Test test = new Test();
        test.test();
    }

    public void test() {
        Expression from = new IntExpression(0, 3);
        Expression to = new IntExpression(0, 7);
        registerRule(from, to);
        Node result = transform(to);
```

```
        System.out.print(to);
        System.out.print(" is transformed to ");
        System.out.println(result);
    }
}
```

A.1.4.6 OpenJIT プログラム変換機能動作試験

OpenJIT プログラム変換機能動作試験で用いられるテストドライバは，次のように定義される．

```
import OpenJIT.frontend.tree.*;
import OpenJIT.frontend.flowgraph.*;
import java.util.Vector;
import java.util.Hashtable;
import java.util.Enumeration;

public class Test extends ASTTransformer {
    public Test() {
        super(null);
    }

    public static void main(String args[]) {
        Test test = new Test();
        test.test();
    }

    public void test() {
        Expression from = new IntExpression(0, 3);
        Expression to = new IntExpression(0, 7);
        registerRule(from, to);
```

```
        System.out.print("registering: ");
        System.out.print(from);
        System.out.print(" -> ");
        System.out.println(to);

        Enumeration keys = dst.keys();
        Enumeration elements = dst.elements();
        while (keys.hasMoreElements()) {
            System.out.print("registered rule: ");
            System.out.print(keys.nextElement());
            System.out.print(" -> ");
            System.out.println(elements.nextElement());
        }

        Node result = match(from);
        if (result != null) {
            System.out.print(from);
            System.out.print(" matches with ");
            System.out.println(result);
            result = transform(from);
            System.out.print(from);
            System.out.print(" is transformed to ");
            System.out.println(result);
        }
    }
}
```

A.1.5 OpenJIT フロントエンド用テストドライバで使用されているその他のクラス

OpenJIT フロントエンド用テストドライバでは，クラスファイルを読み込むために次のように定義される OpenJIT.frontend.classfile パッケージを使用している．

A.1.5.1 public class AccessFlag implements Constants

```
/*
 * $Id: AccessFlag.java,v 1.1 1998/12/20 18:45:21 maruyama Exp $
 */

package OpenJIT.frontend.classfile;

import OpenJIT.Constants;

public class AccessFlag implements Constants {
    public static final boolean isPublic(int flags) {
        return (flags & ACC_PUBLIC) != 0;
    }
    public static final boolean isPrivate(int flags) {
        return (flags & ACC_PRIVATE) != 0;
    }
    public static final boolean isProtected(int flags) {
        return (flags & ACC_PROTECTED) != 0;
    }
    public static final boolean isStatic(int flags) {
        return (flags & ACC_STATIC) != 0;
    }
    public static final boolean isFinal(int flags) {
        return (flags & ACC_FINAL) != 0;
    }
```

```
public static final boolean isSuper(int flags) {
    return (flags & ACC_SUPER) != 0;
}
public static final boolean isSynchronized(int flags) {
    return (flags & ACC_SYNCHRONIZED) != 0;
}
public static final boolean isVolatile(int flags) {
    return (flags & ACC_VOLATILE) != 0;
}
public static final boolean isTransient(int flags) {
    return (flags & ACC_TRANSIENT) != 0;
}
public static final boolean isNative(int flags) {
    return (flags & ACC_NATIVE) != 0;
}
public static final boolean isInterface(int flags) {
    return (flags & ACC_INTERFACE) != 0;
}
public static final boolean isAbstract(int flags) {
    return (flags & ACC_ABSTRACT) != 0;
}

public static String toString(int flags) {
    StringBuffer buf = new StringBuffer();
    if (isPublic(flags))
        buf.append("public ");
    if (isPrivate(flags))
        buf.append("private ");
    if (isProtected(flags))
        buf.append("protected ");
    if (isStatic(flags))
```

```
            buf.append("static ");
        if (isFinal(flags))
            buf.append("final ");
        if (isSynchronized(flags))
            buf.append("synchronized ");
        if (isVolatile(flags))
            buf.append("volatile ");
        if (isTransient(flags))
            buf.append("transient ");
        if (isNative(flags))
            buf.append("native ");
        if (isAbstract(flags))
            buf.append("abstract ");
        return buf.toString();
    }
}
```

A.1.5.2 public abstract class AttributeInfo

```
/*
 * $Id: AttributeInfo.java,v 1.6 1998/12/15 14:40:34 maruyama Exp $
 */

package OpenJIT.frontend.classfile;

import OpenJIT.frontend.util.IndentedPrintStream;
import java.io.IOException;

public abstract class AttributeInfo {
    protected ClassFile classFile;
    protected int attributeNameIndex;
    protected int attributeLength;
```

```
protected String attributeName;

public AttributeInfo(int nameIndex) {
    attributeNameIndex = nameIndex;
}

public AttributeInfo(int nameIndex, ClassFileInputStream stream,
                     ClassFile cF) throws IOException {
    classFile = cF;
    attributeNameIndex = nameIndex;
    attributeLength = stream.readU4();
    attributeName = classFile.constantPool.resolveString(nameIndex);
}

public void write(ClassFileOutputStream stream) throws IOException {
    stream.writeU2(attributeNameIndex);
    stream.writeU4(attributeLength);
}

public String toString() {
    return "attribute_name_index = "
        + Integer.toString(attributeNameIndex)
        + ", attribute_length = " + Integer.toString(attributeLength);
}

public String attributeName() {
    return attributeName;
}

abstract public void print(IndentedPrintStream out);
```

```
}
```

A.1.5.3 public class Attributes

```
/*
 * $Id: Attributes.java,v 1.8 1998/12/20 18:45:22 maruyama Exp $
 */

package OpenJIT.frontend.classfile;

import OpenJIT.frontend.util.IndentedPrintStream;
import java.io.IOException;

/**
 * Each instances of the class </code>Attributes</code> represent a
 * set of </code>AttributeInfo</code> objects.  It appears in
 * </code>ClassFile</code> for ‘SourceFile’,
 * in </code>FieldInfo</code> for ‘ConstantValue’, in </code>MethodInfo</code>
 * for ‘Code’, ‘Exceptions’ and in </code>CodeAttribute</code> for
 * debug informations.
 * <p>
 * Each of thier have very similar style and complex, and users may
 * plan to design thier subclasses, for this reason, it uses Factory
 * Method pattern.
 */
public class Attributes {
    public static final String SOURCEFILE = "SourceFile";
    public static final String CONSTANTVALUE = "ConstantValue";
    public static final String CODE = "Code";
    public static final String EXCEPTIONS = "Exceptions";
    public static final String LINENUMBERTABLE = "LineNumberTable";
    public static final String LOCALVARIABLETABLE = "LocalVariableTable";
```

```
/**
 * It holds a set of </code>AttributeInfo</code>.
 */
protected AttributeInfo attributes[];

/**
 * Constructor.  It reads bytestream from </code>stream</code> and
 * initialize itself.  To determine what kind of attributes to be
 * read, it references a ClassFile.constantPool.
 */
public Attributes(ClassFileInputStream stream, ClassFile cF)
    throws IOException {
    int count = stream.readU2();
    attributes = new AttributeInfo[count];
    for (int i = 0; i < count; i++) {
        int nameIndex = stream.readU2();
        String attributeName = cF.constantPool.resolveString(nameIndex);
        if (attributeName.equals(SOURCEFILE))
            attributes[i] = sourceFile(nameIndex, stream, cF);
        else if (attributeName.equals(CONSTANTVALUE))
            attributes[i] = constantValue(nameIndex, stream, cF);
        else if (attributeName.equals(CODE))
            attributes[i] = code(nameIndex, stream, cF);
        else if (attributeName.equals(EXCEPTIONS))
            attributes[i] = exceptions(nameIndex, stream, cF);
        else if (attributeName.equals(LINENUMBERTABLE))
            attributes[i] = lineNumberTable(nameIndex, stream, cF);
        else if (attributeName.equals(LOCALVARIABLETABLE))
            attributes[i] = localVariableTable(nameIndex, stream, cF);
        else
```

```
            attributes[i] = unknown(attributeName, nameIndex, stream, cF);
    }
}

/**
 * These are factory method to construct various attribute object
 * called by constructor.  Subclasses can override these to
 * change the implementation of each attribute object.
 */
public AttributeInfo sourceFile(int nameIndex, ClassFileInputStream stream,
                                ClassFile cF) throws IOException {
    return new SourceFileAttribute(nameIndex, stream, cF);
}

public AttributeInfo constantValue(int nameIndex,
                                   ClassFileInputStream stream,
                                   ClassFile cF) throws IOException {
    return new ConstantValueAttribute(nameIndex, stream, cF);
}

public AttributeInfo code(int nameIndex, ClassFileInputStream stream,
                          ClassFile cF) throws IOException {
    return new CodeAttribute(nameIndex, stream, cF);
}

public AttributeInfo exceptions(int nameIndex, ClassFileInputStream stream,
                                ClassFile cF) throws IOException {
    return new ExceptionsAttribute(nameIndex, stream, cF);
}

public AttributeInfo lineNumberTable(int nameIndex,
```

```
                                        ClassFileInputStream stream,
                                        ClassFile cF)
    throws IOException {
    return new LineNumberTableAttribute(nameIndex, stream, cF);
}

public AttributeInfo localVariableTable(int nameIndex,
                                            ClassFileInputStream stream,
                                            ClassFile cF)
    throws IOException {
    return new LocalVariableTableAttribute(nameIndex, stream, cF);
}

/**
 * This is treated specially because subclass may override it to
 * handle some AttributeInfo about what I don't know but she know.
 */
public AttributeInfo unknown(String attributeName, int nameIndex,
                             ClassFileInputStream stream,
                             ClassFile cF) throws IOException {
    return new GenericAttribute(nameIndex, stream, cF);
}

/**
 * Write this into </code>stream</code> with the style of Java classfile.
 */
public void write(ClassFileOutputStream stream) throws IOException {
    int count = attributes.length;
    stream.writeU2(count);
    for (int i = 0; i < count; i++)
        attributes[i].write(stream);
```

```
    }

    /**
     * Pretty printer of ClassFile object.
     */
    public void print(IndentedPrintStream out) {
        int count = attributes.length;
        out.println("u2 attributes_count = " + Integer.toString(count) + ";");
        out.println("attribute_info attributes[] = {");
        out.inc();
        for (int i = 0; i < count; i++)
            attributes[i].print(out);
        out.dec();
        out.println("}");
    }

    /**
     * Returns an AttributeInfo named </code>attributeName</code>.
     */
    public AttributeInfo lookup(String attributeName) {
        for (int i = 0, count = attributes.length; i < count; i++) {
            AttributeInfo attribute = attributes[i];
            if (attribute.attributeName.equals(attributeName))
                return attribute;
        }
        return null;
    }
}
```

A.1.5.4 public class ClassFile implements RuntimeConstants

```
/*
```

```
 * $Id: ClassFile.java,v 1.11 1998/12/28 15:30:01 maruyama Exp $
 */

package OpenJIT.frontend.classfile;

import OpenJIT.frontend.java.RuntimeConstants;
import OpenJIT.frontend.util.IndentedPrintStream;
import java.io.InputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.io.PrintStream;

/**
 * Each instances of the class </code>ClassFile</code> represent a
 * classfile which represent a Java class definition.
 * <p>
 * Its constructor invokes makeInner() method to construct inner
 * structures of ClassFile object for customization about the representation
 * of its inner structure.  If the user decide to modify some of inner
 * structures, the only work should do is to make its subclasses and modify
 * the </code>new*</code> -- factory methods -- to construct appropriate
 * objects.
 * <p>
 * It can be used to not only represent newer created (by a program)
 * ClassFile, but also read from </code>*.class</code> file.
 */
public class ClassFile implements RuntimeConstants {
    /**
     * Each of these are just represent the value in </code>*.class</code>.
     */
    protected int magic;
```

```
protected int minorVersion;
protected int majorVersion;
protected int accessFlags;
protected int thisClass;
protected int superClass;
protected int interfaces[];

/**
 * </code>constantPool</code> represent a ConstantPool structure,
 * it is just an array in classfile but ConstantPool class gives
 * more abstract operations like new value appendings.
 */
protected ConstantPool constantPool;

/**
 * Each elements of </code>fields[]</code> represent a field
 * contained in this java class.
 */
protected FieldInfo fields[];

/**
 * Each elements of </code>methods[]</code> represent a method
 * contained in this java class.
 */
protected MethodInfo methods[];

/**
 * </code>attributes</code> represent some additional informations
 * like file name of the source code.
 */
protected Attributes attributes;
```

```
/**
 * Factory methods for inner complex structures.
 */
protected ConstantPool newConstantPool(ClassFileInputStream stream)
    throws IOException {
    return new ConstantPool(stream);
}
protected Attributes newAttributes(ClassFileInputStream stream,
                                   ClassFile cF) throws IOException {
    return new Attributes(stream, cF);
}

/**
 * Factory methods of FieldInfo/MethodInfo initialized from classfile.
 */
protected FieldInfo newFieldInfo(ClassFileInputStream stream)
    throws IOException {
    return new FieldInfo(stream, this);
}
protected MethodInfo newMethodInfo(ClassFileInputStream stream)
    throws IOException {

    return new MethodInfo(stream, this);
}

/**
 * Constructor.  To keep some chances of modification of its
 * inner structure, it uses factory methods for more complex
 * structures instantiation.
```

```
 */
public ClassFile(InputStream is) throws IOException {
    ClassFileInputStream stream = new ClassFileInputStream(is);
    magic = stream.readU4();
    if (magic != JAVA_MAGIC)
        throw new UnknownFileException("magic not match");
    minorVersion = stream.readU2();
    if (minorVersion != JAVA_MINOR_VERSION)
        throw new UnknownFileException("minor version not match");
    majorVersion = stream.readU2();
    if (majorVersion != JAVA_VERSION)
        throw new UnknownFileException("major version not match");
    constantPool = newConstantPool(stream);
    accessFlags = stream.readU2();
    thisClass = stream.readU2();
    superClass = stream.readU2();
    int count = stream.readU2();
    interfaces = new int[count];
    for (int i = 0; i < count; i++)
        interfaces[i] = (short)stream.readU2();
    count = stream.readU2();
    fields = new FieldInfo[count];
    for (int i = 0; i < count; i++)
        fields[i] = newFieldInfo(stream);
    count = stream.readU2();
    methods = new MethodInfo[count];
    for (int i = 0; i < count; i++)
        methods[i] = newMethodInfo(stream);
    attributes = newAttributes(stream, this);
}
```

```
/**
 * Write this into </code>stream</code> with the style of Java classfile.
 */
public void write(OutputStream os) throws IOException {
    ClassFileOutputStream stream = new ClassFileOutputStream(os);
    stream.writeU4(magic);
    stream.writeU2(minorVersion);
    stream.writeU2(majorVersion);
    constantPool.write(stream);
    stream.writeU2(accessFlags);
    stream.writeU2(thisClass);
    stream.writeU2(superClass);
    int count = interfaces.length;
    stream.writeU2(count);
    for (int i = 0; i < count; i++)
        stream.writeU2(interfaces[i]);
    count = fields.length;
    stream.writeU2(count);
    for (int i = 0; i < count; i++)
        fields[i].write(stream);
    count = methods.length;
    stream.writeU2(count);
    for (int i = 0; i < count; i++)
        methods[i]. write(stream);
    attributes.write(stream);
}

/**
 * Pretty printer of ClassFile object.
 */
public void print(PrintStream out) {
```

```
IndentedPrintStream iout = new IndentedPrintStream(out, 4);
iout.println("ClassFile {");
iout.inc();
iout.println("u4 magic = 0x" + Integer.toHexString(magic));
iout.println("u2 minor_version = " + Integer.toString(minorVersion));
iout.println("u2 major_version = " + Integer.toString(majorVersion));
constantPool.print(iout);
iout.println("u2 access_flags = 0x"
             + Integer.toHexString(accessFlags));
iout.println("u2 this_class = " + Integer.toString(thisClass));
iout.println("u2 super_class = " + Integer.toString(superClass));
int count = interfaces.length;
iout.println("u2 interfaces_count = " + Integer.toString(count));
if (count > 0) {
    StringBuffer buf = new StringBuffer();
    buf.append("{ " + Integer.toString(interfaces[0]));
    for (int i = 1; i < count; i++)
        buf.append(", " + Integer.toString(interfaces[i]));
    buf.append(" }");
    iout.println("u2 interfaces[] = " + buf.toString());
} else
    iout.println("u2 interfaces[]");
count = fields.length;
iout.println("u2 fields_count = " + Integer.toString(count));
iout.inc();
for (int i = 0; i < count; i++)
    fields[i].print(iout);
iout.dec();
count = methods.length;
iout.println("u2 methods_count = " + Integer.toString(count));
iout.inc();
```

```
        for (int i = 0; i < count; i++)
            methods[i].print(iout);
        iout.dec();
        attributes.print(iout);
        iout.println("}");
    }

    /**
     * Accessors.
     */
    public ConstantPool constantPool() {
        return constantPool;
    }

    public String thisClassName() {
        return constantPool.resolveClassName(thisClass);
    }

    public String superClassName() {
        return constantPool.resolveClassName(superClass);
    }
}
```

A.1.5.5 public class ClassFileInputStream

```
/*
 * $Id: ClassFileInputStream.java,v 1.4 1999/01/02 07:57:39 maruyama Exp $
 */

package OpenJIT.frontend.classfile;

import java.io.InputStream;
```

```
import java.io.IOException;

public class ClassFileInputStream {
    private InputStream stream;

    public ClassFileInputStream(InputStream stream) {
        this.stream = stream;
    }

    public int readU1() throws IOException {
        int d = stream.read();
        if (d == -1)
            throw new IOException("End of file");
        return d;
    }

    public int readU2() throws IOException {
        return (readU1() << 8) | readU1();
    }

    public int readU4() throws IOException {
        return (readU2() << 16) | readU2();
    }

    public long readU8() throws IOException {

        long high = readU4();
        long low = readU4();
        long result = (high << 32) | low;
        return result;
    }
```

```
    public float readFloat() throws IOException {
        return Float.intBitsToFloat(readU4());
    }

    public double readDouble() throws IOException {
        return Double.longBitsToDouble(readU8());
    }

    public byte[] readBytes(int length) throws IOException {
        byte result[] = new byte[length];
        for (int i = 0; i < length; i++)
            result[i] = (byte)(readU1() & 0xff);
        return result;
    }
}
```

A.1.5.6 public class ClassFileOutputStream

```
/*
 * $Id: ClassFileOutputStream.java,v 1.2 1998/11/24 02:15:01 maruyama Exp $
 */

package OpenJIT.frontend.classfile;

import java.io.OutputStream;
import java.io.IOException;

public class ClassFileOutputStream {
    private OutputStream stream;

    public ClassFileOutputStream(OutputStream stream) {
```

```
    this.stream = stream;
}

public void writeU1(int d) throws IOException {
    stream.write(d);
}

public void writeU2(int d) throws IOException {
    writeU1((d >> 8) & 0xff);
    writeU1(d & 0xff);
}

public void writeU4(int d) throws IOException {
    writeU2((d >> 16) & 0xffff);
    writeU2(d & 0xffff);
}

public void writeU8(long d) throws IOException {
    writeU4((int)((d >> 32) & 0xffffffff));
    writeU4((int)(d & 0xffffffff));
}

public void writeFloat(float d) throws IOException {
    writeU4(Float.floatToIntBits(d));
}

public void writeDouble(double d) throws IOException {
    writeU8(Double.doubleToLongBits(d));
}

public void writeBytes(byte bytes[]) throws IOException {
```

```
        int length = bytes.length;
        for (int i = 0; i < length; i++)
            writeU1(bytes[i]);
    }
}
```

A.1.5.7 public class CodeAttribute extends AttributeInfo

```
/*
 * $Id: CodeAttribute.java,v 1.7 1998/12/15 14:40:35 maruyama Exp $
 */

package OpenJIT.frontend.classfile;

import OpenJIT.ExceptionHandler;
import OpenJIT.frontend.util.IndentedPrintStream;
import java.io.IOException;

/**
 * Each instances of the class </code>CodeAttribute</code> represent the
 * body of the method contining it.
 */
public class CodeAttribute extends AttributeInfo {
    protected int maxStack;
    protected int maxLocals;
    protected byte code[];
    protected ExceptionHandler exceptionTable[];
    protected Attributes attributes;

    /**
     * Constructor.  This is used to construct from parts of classfile.
     */
```

```
public CodeAttribute(int nameIndex, ClassFileInputStream stream,
                     ClassFile cF) throws IOException {
    super(nameIndex, stream, cF);
    maxStack = stream.readU2();
    maxLocals = stream.readU2();
    code = stream.readBytes(stream.readU4());
    int count = stream.readU2();
    exceptionTable = new ExceptionHandler[count];
    for (int i = 0; i < count; i++) {
        ExceptionHandler handler = new ExceptionHandler();
        handler.startPC = stream.readU2();
        handler.endPC = stream.readU2();
        handler.handlerPC = stream.readU2();
        handler.catchType = stream.readU2();
        exceptionTable[i] = handler;
    }
    attributes = cF.newAttributes(stream, cF);
}

/**
 * Write this into </code>stream</code> with the style of Java classfile.
 */
public void write(ClassFileOutputStream stream) throws IOException {
    super.write(stream);
    stream.writeU2(maxStack);
    stream.writeU2(maxLocals);
    int length = code.length;
    stream.writeU4(length);
    for (int i = 0; i < length; i++)
        stream.writeU1(code[i]);
    length = exceptionTable.length;
```

```
    stream.writeU2(length);
    for (int i = 0; i < length; i++) {
        ExceptionHandler handler = exceptionTable[i];
        stream.writeU2(handler.startPC);
        stream.writeU2(handler.endPC);
        stream.writeU2(handler.handlerPC);
        stream.writeU2(handler.catchType);
    }
    attributes.write(stream);
}

/**
 * Return simple description of this object in a String.
 */
public String toString() {
    return "Code_attribute { " + super.toString()
        + ", max_stack = " + Integer.toString(maxStack)
        + ", max_locals = " + Integer.toString(maxLocals)
        + ", code_length = " + Integer.toString(code.length)
        + ", code[], exception_table_length = "
        + Integer.toString(exceptionTable.length)
        + attributes.toString()
        + " }";
}

/**
 * Pretty printer.
 */
public void print(IndentedPrintStream out) {
    out.println("Code_attribute {");
    out.inc();
```

```
        out.println("u2 attribute_name_index = "
                    + Integer.toString(attributeNameIndex) + ";");
        out.println("u4 attribute_length = "
                    + Integer.toString(attributeLength) + ";");
        out.println("u2 max_stack = " + Integer.toString(maxStack) + ";");
        out.println("u2 max_locals = " + Integer.toString(maxLocals) + ";");
        out.println("u4 code_length = 0x"
                    + Integer.toHexString(code.length) + ";");
        out.println("u1 code[];");
        int count = exceptionTable.length;
        out.println("u2 exception_table_length = "
                    + Integer.toString(count) + ";");
        out.println("exception_table[] = {");
        out.inc();
        for (int i = 0; i < count; i++) {
            ExceptionHandler handler = exceptionTable[i];
            out.println("{ " + Integer.toString(handler.startPC)
                        + ", " + Integer.toString(handler.endPC)
                        + ", " + Integer.toString(handler.handlerPC)
                        + ", " + classFile.constantPool
                                     .resolveString(handler.catchType)
                        + "}");
        }
        out.dec();
        out.println("}");
        attributes.print(out);
        out.dec();
        out.println("}");
    }
    /**
     * Accessors.
```

```
     */
    public int maxLocals() {
        return maxLocals;
    }
    public int maxStack() {
        return maxStack;
    }
    public byte[] code() {
        return code;
    }
    public int codeLength() {
        return code.length;
    }
    public ExceptionHandler[] exceptionTable() {
        return exceptionTable;
    }
}
```

A.1.5.8 public class ConstantClass extends ConstantPoolItem

```
/*
 * $Id: ConstantClass.java,v 1.6 1998/12/15 14:40:35 maruyama Exp $
 */

package OpenJIT.frontend.classfile;

import OpenJIT.frontend.util.IndentedPrintStream;
import java.io.IOException;

/**
 * Each instances of the class </code>ConstantClass</code> represent a
 * CONSTANT_Class_info structure of which exists in classfile.
```

```
 */
public class ConstantClass extends ConstantPoolItem {
    protected int nameIndex;

    /**
     * Constructor.  This is used to construct from parts of classfile.
     */
    public ConstantClass(ClassFileInputStream stream) throws IOException {
        super(CONSTANT_CLASS);
        nameIndex = stream.readU2();
    }

    /**
     * Constructor.  This is used to construct appropriate instance with
     * </code>index</code> into ConstantPool containing Constant_UTF8_info
     * which represent the classname.
     */
    public ConstantClass(int index) {
        super(CONSTANT_CLASS);
        nameIndex = index;
    }

    /**
     * Returns hashcode calculated from its nameIndex.
     */
    public int hashCode() {
        return nameIndex;
    }

    /**
     * Returns whether the contents are same as given </code>obj</code>.
```

```
 */
public boolean equals(Object obj) {
    if (!(obj instanceof ConstantClass))
        return false;
    return nameIndex == ((ConstantClass)obj).nameIndex;
}

int nameIndex() {
    return nameIndex;
}

/**
 * Write this into </code>stream</code> with the style of Java classfile.
 */
public void write(ClassFileOutputStream stream) throws IOException {
    stream.writeU1(tag);
    stream.writeU2(nameIndex);
}

/**
 * Return simple description of this object in a String.
 */
public String toString() {
    return "class name #" + Integer.toString(nameIndex);
}

/**
 * Pretty printer.
 */
public void print(IndentedPrintStream out) {
    out.println("CONSTANT_Class {");
```

```
        out.inc();
        out.println("u1 tag = " + Integer.toString(tag) + ";");
        out.println("u2 name_index = " + Integer.toString(nameIndex) + ";");
        out.dec();
        out.println("}");
    }
}
```

A.1.5.9 public class ConstantDouble extends ConstantPoolItem

```
/*
 * $Id: ConstantDouble.java,v 1.5 1998/11/28 11:13:28 maruyama Exp $
 */

package OpenJIT.frontend.classfile;

import OpenJIT.frontend.util.IndentedPrintStream;
import java.io.IOException;

/**
 * Each instances of the class </code>ConstantDouble</code> represent a
 * CONSTANT_Double_info structure of which exists in classfile.
 */
public class ConstantDouble extends ConstantPoolItem {
    protected double value;

    /**
     * Constructor.  This is used to construct from parts of classfile.
     */
    public ConstantDouble(ClassFileInputStream stream) throws IOException {
        super(CONSTANT_DOUBLE);
        value = stream.readDouble();
```

```
}

/**
 * Constructor.  This is used to construct appropriate instance from
 * </code>Double</code> object.
 */
public ConstantDouble(Double value) {
    super(CONSTANT_DOUBLE);
    this.value = value.doubleValue();
}

/**
 * Returns its content as </code>Double</code> object.
 */
public Double toDouble() {
    return new Double(value);
}

/**
 * Write this into </code>stream</code> with the style of Java classfile.
 */
public void write(ClassFileOutputStream stream) throws IOException {
    stream.writeU1(tag);
    stream.writeDouble(value);
}

/**
 * Return simple description of this object in a String.
 */
public String toString() {
    return Double.toString(value);
```

```
    }

    /**
     * Pretty printer.
     */
    public void print(IndentedPrintStream out) {
        out.println("CONSTANT_Double {");
        out.inc();
        out.println("u1 tag = " + Integer.toString(tag) + ";");
        out.println("u8 bytes = " + Double.toString(value) + ";");
        out.dec();
        out.println("}");
    }
}
```

A.1.5.10 public class ConstantFieldref extends ConstantRef

```
/*
 * $Id: ConstantFieldref.java,v 1.5 1998/11/28 11:13:28 maruyama Exp $
 */

package OpenJIT.frontend.classfile;

import OpenJIT.frontend.util.IndentedPrintStream;
import java.io.IOException;

/**
 * Each instances of the class </code>ConstantFieldref</code> represent a
 * CONSTANT_Fieldref_info structure of which exists in classfile.
 */
public class ConstantFieldref extends ConstantRef {
    /**
```

```
 * Constructor.  This is used to construct from parts of classfile.
 */
public ConstantFieldref(ClassFileInputStream stream) throws IOException {
    super(CONSTANT_FIELD, stream);
}

/**
 * Constructor.  This is used to construct appropriate instance with
 * </code>classIndex</code> and </code>nameAndTypeIndex</code> into
 * ConstantPool.
 */
public ConstantFieldref(int classIndex, int nameAndTypeIndex) {
    super(CONSTANT_FIELD, classIndex, nameAndTypeIndex);
}

/**
 * Returns whether the contents are same as given </code>obj</code>.
 */
public boolean equals(Object obj) {
    if (!(obj instanceof ConstantFieldref))
        return false;
    ConstantFieldref item = (ConstantFieldref)obj;
    return (classIndex == item.classIndex
            && nameAndTypeIndex == item.nameAndTypeIndex);
}

/**
 * Return simple description of this object in a String.
 */
public String toString() {
    return "Fieldref class#" + Integer.toString(classIndex) + " sig#"
```

```
            + Integer.toString(nameAndTypeIndex);
    }

    /**
     * Pretty printer.
     */
    public void print(IndentedPrintStream out) {
        out.println("CONSTANT_Fieldref {");
        out.inc();
        out.println("u1 tag = " + Integer.toString(tag) + ";");
        out.println("u2 class_index = " + Integer.toString(classIndex) + ";");
        out.println("u2 name_and_type_index = "
                    + Integer.toString(nameAndTypeIndex) + ";");
        out.dec();
        out.println("}");
    }
}
```

A.1.5.11 public class ConstantFloat extends ConstantPoolItem

```
/*
 * $Id: ConstantFloat.java,v 1.5 1998/11/28 11:13:28 maruyama Exp $
 */

package OpenJIT.frontend.classfile;

import OpenJIT.frontend.util.IndentedPrintStream;
import java.io.IOException;

/**
 * Each instances of the class </code>ConstantFloat</code> represent a
 * CONSTANT_Float_info structure of which exists in classfile.
```

```
 */
public class ConstantFloat extends ConstantPoolItem {
    protected float value;

    /**
     * Constructor.  This is used to construct from parts of classfile.
     */
    public ConstantFloat(ClassFileInputStream stream) throws IOException {
        super(CONSTANT_FLOAT);
        value = stream.readFloat();
    }

    /**
     * Constructor.  This is used to construct appropriate instance from
     * </code>Float</code> object.
     */
    public ConstantFloat(Float value) {
        super(CONSTANT_FLOAT);
        this.value = value.floatValue();
    }

    /**
     * Returns its content as </code>Float</code> object.
     */
    public Float toFloat() {
        return new Float(value);
    }

    /**
     * Write this into </code>stream</code> with the style of Java classfile.
     */
```

```
    public void write(ClassFileOutputStream stream) throws IOException {
        stream.writeU1(tag);
        stream.writeFloat(value);
    }

    /**
     * Return simple description of this object in a String.
     */
    public String toString() {
        return Float.toString(value);
    }

    /**
     * Pretty printer.
     */
    public void print(IndentedPrintStream out) {
        out.println("CONSTANT_Float {");
        out.inc();
        out.println("u1 tag = " + Integer.toString(tag) + ";");
        out.println("u4 bytes = " + Float.toString(value) + ";");
        out.dec();
        out.println("}");
    }
}
```

A.1.5.12 public class ConstantInteger extends ConstantPoolItem

```
/*
 * $Id: ConstantInteger.java,v 1.5 1998/11/28 11:13:28 maruyama Exp $
 */

package OpenJIT.frontend.classfile;
```

```
import OpenJIT.frontend.util.IndentedPrintStream;
import java.io.IOException;

/**
 * Each instances of the class </code>ConstantInteger</code> represent a
 * CONSTANT_Integer_info structure of which exists in classfile.
 */
public class ConstantInteger extends ConstantPoolItem {
    protected int value;

    /**
     * Constructor.  This is used to construct from parts of classfile.
     */
    public ConstantInteger(ClassFileInputStream stream) throws IOException {
        super(CONSTANT_INTEGER);
        value = stream.readU4();
    }

    /**
     * Constructor.  This is used to construct appropriate instance from
     * </code>Integer</code> object.
     */
    public ConstantInteger(Integer integer) {
        super(CONSTANT_INTEGER);
        value = integer.intValue();
    }

    /**
     * Returns its content as Integer object.
     */
```

```
    public Integer toInteger() {
        return new Integer(value);
    }

    /**
     * Write this into </code>stream</code> with the style of Java classfile.
     */
    public void write(ClassFileOutputStream stream) throws IOException {
        stream.writeU1(tag);
        stream.writeU4(value);
    }

    /**
     * Return simple description of this object in a String.
     */
    public String toString() {
        return Integer.toString(value);
    }

    /**
     * Pretty printer.
     */
    public void print(IndentedPrintStream out) {
        out.println("CONSTANT_Integer {");
        out.inc();
        out.println("u1 tag = " + Integer.toString(tag) + ";");
        out.println("u4 bytes = " + Integer.toString(value) + ";");
        out.dec();
        out.println("}");
    }
}
```

A.1.5.13 public class ConstantInterfaceMethodref extends ConstantRef

```
/*
 * $Id: ConstantInterfaceMethodref.java,v 1.6 1998/11/28 11:13:28 maruyama Exp $
 */

package OpenJIT.frontend.classfile;

import OpenJIT.frontend.util.IndentedPrintStream;
import java.io.IOException;

/**
 * Each instances of the class </code>ConstantInterfaceMethodref</code>
 * represent a CONSTANT_InterfaceMethodref_info structure of which
 * exists in classfile.
 */
public class ConstantInterfaceMethodref extends ConstantRef {
    /**
     * Constructor.  This is used to construct from parts of classfile.
     */
    public ConstantInterfaceMethodref(ClassFileInputStream stream)
        throws IOException {
        super(CONSTANT_INTERFACEMETHOD, stream);
    }

    /**
     * Constructor.  This is used to construct appropriate instance with
     * </code>classIndex</code> and </code>nameAndTypeIndex</code> into
     * ConstantPool.
     */
    public ConstantInterfaceMethodref(int classIndex, int nameAndTypeIndex) {
        super(CONSTANT_INTERFACEMETHOD, classIndex, nameAndTypeIndex);
```

```
}

/**
 * Returns whether the contents are same as given </code>obj</code>.
 */
public boolean equals(Object obj) {
    if (!(obj instanceof ConstantInterfaceMethodref))
        return false;
    ConstantInterfaceMethodref item = (ConstantInterfaceMethodref)obj;
    return (classIndex == item.classIndex
            && nameAndTypeIndex == item.nameAndTypeIndex);
}

/**
 * Return simple description of this object in a String.
 */
public String toString() {
    return "InterfaceMethodref class#" + Integer.toString(classIndex)
        + " sig#" + Integer.toString(nameAndTypeIndex);
}

/**
 * Pretty printer.
 */
public void print(IndentedPrintStream out) {
    out.println("CONSTANT_InterfaceMethodref {");
    out.inc();
    out.println("u1 tag = " + Integer.toString(tag) + ";");
    out.println("u2 class_index = " + Integer.toString(classIndex) + ";");
    out.println("u2 name_and_type_index = "
                + Integer.toString(nameAndTypeIndex) + ";");
```

```
        out.dec();
        out.println("}");
    }
}
```

A.1.5.14 public class ConstantLong extends ConstantPoolItem

```
/*
 * $Id: ConstantLong.java,v 1.5 1998/11/28 11:13:29 maruyama Exp $
 */

package OpenJIT.frontend.classfile;

import OpenJIT.frontend.util.IndentedPrintStream;
import java.io.IOException;

/**
 * Each instances of the class </code>ConstantLong</code> represent a
 * CONSTANT_Long_info structure of which exists in classfile.
 */
public class ConstantLong extends ConstantPoolItem {
    protected long value;

    /**
     * Constructor.  This is used to construct from parts of classfile.
     */
    public ConstantLong(ClassFileInputStream stream) throws IOException {
        super(CONSTANT_LONG);
        value = stream.readU8();
    }

    /**
```

```
 * Constructor.  This is used to construct appropriate instance from
 * </code>Long</code> object.
 */
public ConstantLong(Long value) {
    super(CONSTANT_LONG);
    this.value = value.longValue();
}

/**
 * Returns its content as </code>Long</code> object.
 */
public Long toLong() {
    return new Long(value);
}

/**
 * Write this into </code>stream</code> with the style of Java classfile.
 */
public void write(ClassFileOutputStream stream) throws IOException {
    stream.writeU1(tag);
    stream.writeU8(value);
}

/**
 * Return simple description of this object in a String.
 */
public String toString() {
    return Long.toString(value);
}

/**
```

```
     * Pretty printer.
     */
    public void print(IndentedPrintStream out) {
        out.println("CONSTANT_Long {");
        out.inc();
        out.println("u1 tag = " + Integer.toString(tag) + ";");
        out.println("u8 bytes = " + Long.toString(value) + ";");
        out.dec();
        out.println("}");
    }
}
```

A.1.5.15 public class ConstantMethodref extends ConstantRef

```
/*
 * $Id: ConstantMethodref.java,v 1.6 1998/11/28 11:13:29 maruyama Exp $
 */

package OpenJIT.frontend.classfile;

import OpenJIT.frontend.util.IndentedPrintStream;
import java.io.IOException;

/**
 * Each instances of the class </code>ConstantMethodref</code> represent a
 * CONSTANT_Methodref_info structure of which exists in classfile.
 */
public class ConstantMethodref extends ConstantRef {
    /**
     * Constructor.  This is used to construct from parts of classfile.
     */
    public ConstantMethodref(ClassFileInputStream stream) throws IOException {
```

```
    super(CONSTANT_METHOD, stream);
}

/**
 * Constructor.  This is used to construct appropriate instance with
 * </code>classIndex</code> and </code>nameAndTypeIndex</code> into
 * ConstantPool.
 */
public ConstantMethodref(int classIndex, int nameAndTypeIndex) {
    super(CONSTANT_METHOD, classIndex, nameAndTypeIndex);
}

/**
 * Returns whether the contents are same as given </code>obj</code>.
 */
public boolean equals(Object obj) {
    if (!(obj instanceof ConstantMethodref))
        return false;
    ConstantMethodref item = (ConstantMethodref)obj;
    return (classIndex == item.classIndex
            && nameAndTypeIndex == item.nameAndTypeIndex);
}

/**
 * Return simple description of this object in a String.
 */
public String toString() {
    return "Methodref class#" + Integer.toString(classIndex) + " sig#"
        + Integer.toString(nameAndTypeIndex);
}
```

```
    /**
     * Pretty printer.
     */
    public void print(IndentedPrintStream out) {
        out.println("CONSTANT_Methodref {");
        out.inc();
        out.println("u1 tag = " + Integer.toString(tag) + ";");
        out.println("u2 class_index = " + Integer.toString(classIndex) + ";");
        out.println("u2 name_and_type_index = "
                    + Integer.toString(nameAndTypeIndex) + ";");
        out.dec();
        out.println("}");
    }
}
```

A.1.5.16 public class ConstantNameAndType extends ConstantPoolItem

```
/*
 * $Id: ConstantNameAndType.java,v 1.5 1998/11/28 11:13:29 maruyama Exp $
 */

package OpenJIT.frontend.classfile;

import OpenJIT.frontend.util.IndentedPrintStream;
import java.io.IOException;

/**
 * Each instances of the class </code>ConstantNameAndType</code> represent
 * a CONSTANT_NameAndType_info structure of which exists in classfile.
 */
public class ConstantNameAndType extends ConstantPoolItem {
    protected int nameIndex;
```

```
protected int descriptorIndex;

/**
 * Constructor.  This is used to construct from parts of classfile.
 */
public ConstantNameAndType(ClassFileInputStream stream)
    throws IOException {
    super(CONSTANT_NAMEANDTYPE);
    nameIndex = stream.readU2();
    descriptorIndex = stream.readU2();
}

/**
 * Constructor.  This is used to construct appropriate instance with
 * </code>nameIndex</code> and </code>typeIndex</code> into
 * ConstantPool.
 */
public ConstantNameAndType(int nameIndex, int typeIndex) {
    super(CONSTANT_NAMEANDTYPE);
    this.nameIndex = nameIndex;
    descriptorIndex = typeIndex;
}

/**
 * Returns its hashcode calculated from nameIndex and descriptorIndex.
 */
public int hashCode() {
    return (nameIndex & 0xff) | ((descriptorIndex & 0xff) << 8)
        | ((nameIndex & 0xff00) << 8) | ((descriptorIndex & 0xff00) << 16);
}
```

```
/**
 * Returns whether the contents are same as given </code>obj</code>.
 */
public boolean equals(Object obj) {
    if (!(obj instanceof ConstantNameAndType))
        return false;
    ConstantNameAndType item = (ConstantNameAndType)obj;
    return (nameIndex == item.nameIndex
            && descriptorIndex == item.descriptorIndex);
}

/**
 * Write this into </code>stream</code> with the style of Java classfile.
 */
public void write(ClassFileOutputStream stream) throws IOException {
    stream.writeU1(tag);
    stream.writeU2(nameIndex);
    stream.writeU2(descriptorIndex);
}

/**
 * Return simple description of this object in a String.
 */
public String toString() {
    return "NameAndType name#" + Integer.toString(nameIndex)
        + " descriptor#" + Integer.toString(descriptorIndex);
}

/**
 * Pretty printer.
 */
```

```
    public void print(IndentedPrintStream out) {
        out.println("CONSTANT_NameAndType {");
        out.inc();
        out.println("u1 tag = " + Integer.toString(tag) + ";");
        out.println("u2 name_index = " + Integer.toString(nameIndex) + ";");
        out.println("u2 descriptor_index = "
                    + Integer.toString(descriptorIndex) + ";");
        out.dec();
        out.println("}");
    }
}
```

A.1.5.17 public class ConstantPool implements RuntimeConstants

```
/*
 * $Id: ConstantPool.java,v 1.8 1998/12/15 14:40:35 maruyama Exp $
 */

package OpenJIT.frontend.classfile;

import OpenJIT.frontend.java.RuntimeConstants;
import OpenJIT.frontend.util.IndentedPrintStream;
import OpenJIT.frontend.util.LookupHashtable;
import java.io.IOException;
import java.util.Vector;
import java.util.Hashtable;

/**
 * Each instances of the class </code>ConstantPool</code> represent a
 * set of </code>ConstantPoolItem</code> objects.  Each subclass of
 * ConstantPoolItem is used to represent some constant value for
 * classfile itself and bytecode.
```

```
 */
public class ConstantPool implements RuntimeConstants {
    /**
     * Factory methods for various kind of </code>ConstantPoolItem</code>
     */
    protected ConstantPoolItem readUTF8(ClassFileInputStream stream)
        throws IOException {
        return new ConstantUTF8(stream);
    }
    protected ConstantPoolItem makeUTF8(String value) {
        return new ConstantUTF8(value);
    }

    protected ConstantPoolItem readUnicode(ClassFileInputStream stream)
        throws IOException {
        throw new UnknownFileException("CONSTANT_Unicode appears");
    }
    protected ConstantPoolItem makeUnicode(String value) {
        throw new UnknownFileException("CONSTANT_Unicode appears");
    }

    protected ConstantPoolItem readInteger(ClassFileInputStream stream)
        throws IOException {
        return new ConstantInteger(stream);
    }
    protected ConstantPoolItem makeInteger(Integer value) {
        return new ConstantInteger(value);
    }

    protected ConstantPoolItem readFloat(ClassFileInputStream stream)
        throws IOException {
```

```
    return new ConstantFloat(stream);
}
protected ConstantPoolItem makeFloat(Float value) {
    return new ConstantFloat(value);
}

protected ConstantPoolItem readLong(ClassFileInputStream stream)
    throws IOException {
    return new ConstantLong(stream);
}
protected ConstantPoolItem makeLong(Long value) {
    return new ConstantLong(value);
}

protected ConstantPoolItem readDouble(ClassFileInputStream stream)
    throws IOException {
    return new ConstantDouble(stream);
}
protected ConstantPoolItem makeDouble(Double value) {
    return new ConstantDouble(value);
}

protected ConstantPoolItem readClass(ClassFileInputStream stream)
    throws IOException {
    return new ConstantClass(stream);
}
protected ConstantPoolItem makeClass(int nameIndex) {
    return new ConstantClass(nameIndex);
}

protected ConstantPoolItem readString(ClassFileInputStream stream)
```

```
    throws IOException {
    return new ConstantString(stream);
}
protected ConstantPoolItem makeString(int stringIndex) {
    return new ConstantString(stringIndex);
}

protected ConstantPoolItem readField(ClassFileInputStream stream)
    throws IOException {
    return new ConstantFieldref(stream);
}
protected ConstantPoolItem makeField(int classIndex, int descIndex) {
    return new ConstantFieldref(classIndex, descIndex);
}

protected ConstantPoolItem readMethod(ClassFileInputStream stream)
    throws IOException {
    return new ConstantMethodref(stream);
}
protected ConstantPoolItem makeMethod(int classIndex, int descIndex) {
    return new ConstantMethodref(classIndex, descIndex);
}

protected ConstantPoolItem readInterface(ClassFileInputStream stream)
    throws IOException {
    return new ConstantInterfaceMethodref(stream);
}
protected ConstantPoolItem makeInterface(int classIndex, int descIndex) {
    return new ConstantInterfaceMethodref(classIndex, descIndex);
}
```

```
protected ConstantPoolItem readNameAndType(ClassFileInputStream stream)
    throws IOException {
    return new ConstantNameAndType(stream);
}
protected ConstantPoolItem makeNameAndType(int nameIndex, int descIndex) {
    return new ConstantNameAndType(nameIndex, descIndex);
}

/**
 * </code>constantItems</code> contains a set of ConstantPoolItems
 * in variable length array.
 */
protected Vector constantItems;

/**
 * These are for guarantee that each ConstantPoolItem are the unique
 * object in one ConstantPool.
 */
private LookupHashtable utf8s = new LookupHashtable() {
    protected Object resolve(Object key) {
        ConstantPoolItem item = makeUTF8((String)key);
        int index;
        synchronized (constantItems) {
            index = constantItems.size();
            constantItems.addElement(item);
        }
        return new Integer(index);
    }
};

private LookupHashtable integers = new LookupHashtable() {
```

```
    protected Object resolve(Object key) {
        ConstantPoolItem item = makeInteger((Integer)key);
        int index;
        synchronized (constantItems) {
            index = constantItems.size();
            constantItems.addElement(item);
        }
        return new Integer(index);
    }
};

private LookupHashtable floats = new LookupHashtable() {
    protected Object resolve(Object key) {
        ConstantPoolItem item = makeFloat((Float)key);
        int index;
        synchronized (constantItems) {
            index = constantItems.size();
            constantItems.addElement(item);
        }
        return new Integer(index);
    }
};

private LookupHashtable longs = new LookupHashtable() {
    protected Object resolve(Object key) {
        ConstantPoolItem item = makeLong((Long)key);
        int index;
        synchronized (constantItems) {
            index = constantItems.size();
            constantItems.addElement(item);
            constantItems.addElement(null);
```

```
        }
        return new Integer(index);
    }
};

private LookupHashtable doubles = new LookupHashtable() {
    protected Object resolve(Object key) {
        ConstantPoolItem item = makeDouble((Double)key);
        int index;
        synchronized (constantItems) {
            index = constantItems.size();
            constantItems.addElement(item);
            constantItems.addElement(null);
        }
        return new Integer(index);
    }
};

private LookupHashtable classes = new LookupHashtable() {
    protected Object resolve(Object key) {
        ConstantPoolItem item = (ConstantClass)key;
        int index;
        synchronized (constantItems) {
            index = constantItems.size();
            constantItems.addElement(key);
        }
        return new Integer(index);
    }
};

private LookupHashtable strings = new LookupHashtable() {
```

```
    protected Object resolve(Object key) {
        ConstantPoolItem item = (ConstantString)key;
        int index;
        synchronized (constantItems) {
            index = constantItems.size();
            constantItems.addElement(key);
        }
        return new Integer(index);
    }
};

private LookupHashtable fieldrefs = new LookupHashtable() {
    protected Object resolve(Object key) {
        ConstantPoolItem item = (ConstantFieldref)key;
        int index;
        synchronized (constantItems) {
            index = constantItems.size();
            constantItems.addElement(key);
        }
        return new Integer(index);
    }
};

private LookupHashtable methodrefs = new LookupHashtable() {
    protected Object resolve(Object key) {
        ConstantPoolItem item = (ConstantMethodref)key;
        int index;
        synchronized (constantItems) {
            index = constantItems.size();
            constantItems.addElement(key);
        }
```

```
        return new Integer(index);
    }
};

private LookupHashtable interfacerefs = new LookupHashtable() {
    protected Object resolve(Object key) {
        ConstantPoolItem item = (ConstantInterfaceMethodref)key;
        int index;
        synchronized (constantItems) {
            index = constantItems.size();
            constantItems.addElement(key);
        }
        return new Integer(index);
    }
};

private LookupHashtable nameAndTypes = new LookupHashtable() {
    protected Object resolve(Object key) {
        ConstantPoolItem item = (ConstantNameAndType)key;
        int index;
        synchronized (constantItems) {
            index = constantItems.size();
            constantItems.addElement(key);
        }
        return new Integer(index);
    }
};

/**
 * Constructor.  This reads bytestream from </code>stream</code> and
 * initialize constantItems.
```

```
 */
public ConstantPool(ClassFileInputStream stream) throws IOException {
    constantItems = new Vector();
    constantItems.addElement(null);
    int count = stream.readU2();
    for (int i = 1; i < count; i++) {
        int tag = stream.readU1();
        ConstantPoolItem item;
        switch (tag) {
        case CONSTANT_UTF8:
            item = readUTF8(stream);
            utf8s.put(item.toString(), new Integer(i));
            constantItems.addElement(item);
            break;
            /*
             * case CONSTANT_UNICODE:
             *     item = readUnicode(stream);
             *     unicodes.put(item, new Integer(i));
             *     constantItems.addElement(item);
             *     break;
             */
        case CONSTANT_INTEGER:
            item = readInteger(stream);
            integers.put(((ConstantInteger)item).toInteger(),
                         new Integer(i));
            constantItems.addElement(item);
            break;
        case CONSTANT_FLOAT:
            item = readFloat(stream);
            floats.put(((ConstantFloat)item).toFloat(), new Integer(i));
            constantItems.addElement(item);
```

```
            break;
        case CONSTANT_LONG:
            item = readLong(stream);
            longs.put(((ConstantLong)item).toLong(), new Integer(i));
            constantItems.addElement(item);
            constantItems.addElement(null);
            i++;
            break;
        case CONSTANT_DOUBLE:
            item = readDouble(stream);
            doubles.put(((ConstantDouble)item).toDouble(), new Integer(i));
            constantItems.addElement(item);
            constantItems.addElement(null);
            i++;
            break;
        case CONSTANT_CLASS:
            item = readClass(stream);
            classes.put(item, new Integer(i));
            constantItems.addElement(item);
            break;
        case CONSTANT_STRING:
            item = readString(stream);
            strings.put(item, new Integer(i));
            constantItems.addElement(item);
            break;
        case CONSTANT_FIELD:
            item = readField(stream);
            fieldrefs.put(item, new Integer(i));
            constantItems.addElement(item);
            break;
        case CONSTANT_METHOD:
```

```
            item = readMethod(stream);
            methodrefs.put(item, new Integer(i));
            constantItems.addElement(item);
            break;
        case CONSTANT_INTERFACEMETHOD:
            item = readInterface(stream);
            interfacerefs.put(item, new Integer(i));
            constantItems.addElement(item);
            break;
        case CONSTANT_NAMEANDTYPE:
            item = readNameAndType(stream);
            nameAndTypes.put(item, new Integer(i));
            constantItems.addElement(item);
            break;
        default:
            throw new UnknownFileException("Unknown tag ("
                                           + Integer.toString(tag)
                                           + ") appears");
        }
    }
}

/**
 * Write this into </code>stream</code> with the style of Java classfile.
 */
public void write(ClassFileOutputStream stream) throws IOException {
    int count = constantItems.size();
    stream.writeU2(count);
    for (int i = 1; i < count; i++) {
        ConstantPoolItem d = (ConstantPoolItem)constantItems.elementAt(i);
        if (d != null)
```

```
            d.write(stream);
    }
}

/**
 * Accessor.
 */
public ConstantPoolItem itemAt(int index) {
    return (ConstantPoolItem)constantItems.elementAt(index);
}

/**
 * Pretty printer.
 */
public void print(IndentedPrintStream out) {
    int count = constantItems.size();
    out.println("u2 constant_pool_count = "
                + Integer.toString(count) + ";");
    out.println("cp_info constant_pool[] = {");
    out.inc();
    for (int i = 1; i < count; i++) {
        ConstantPoolItem item
            = (ConstantPoolItem)constantItems.elementAt(i);
        out.println("// [" + Integer.toString(i) + "]");
        if (item == null) {
            out.println("null");
        } else
            item.print(out);
    }
    out.dec();
    out.println("}");
```

```
}

/**
 * Returns a String object appropriate for given </code>index</code>.
 */
public String resolveString(int index) {
    ConstantPoolItem item
        = (ConstantPoolItem)constantItems.elementAt(index);
    if (item.isString())
        return resolveString(((ConstantString)item).stringIndex);
    if (item.isClass())
        return resolveString(((ConstantClass)item).nameIndex);
    return item.toString();
}

/**
 * Returns the class name for given </code>index</code> as String object.
 */
public String resolveClassName(int index) {
    ConstantPoolItem item
        = (ConstantPoolItem)constantItems.elementAt(index);
    if (item.isClass())
        return resolveString(((ConstantClass)item).nameIndex);
    if (item.isField() || item.isMethod() || item.isInterface())
        return resolveString(((ConstantRef)item).classIndex);
    throw new ResolveError("Invalid use of resolveClassName()");
}

/**
 * Returns the name of member for given </code>index</code>
 * as String object .
```

```
 */
public String resolveMemberName(int index) {
    ConstantPoolItem item
        = (ConstantPoolItem)constantItems.elementAt(index);
    if (item.isField() || item.isMethod() || item.isInterface()) {
        item = (ConstantPoolItem)constantItems
            .elementAt(((ConstantRef)item).nameAndTypeIndex);
        return resolveString(((ConstantNameAndType)item).nameIndex);
    }
    throw new ResolveError("Invalid use of resolveClassName()");
}

/**
 * Returns the descriptor of member for given </code>index</code>
 * as String object.
 */
public byte[] resolveMemberDescriptor(int index) {
    ConstantPoolItem item
        = (ConstantPoolItem)constantItems.elementAt(index);
    if (item.isField() || item.isMethod() || item.isInterface()) {
        item = (ConstantPoolItem)constantItems
            .elementAt(((ConstantRef)item).nameAndTypeIndex);
        return ((ConstantUTF8)
                constantItems.elementAt(((ConstantNameAndType)item)
                                        .descriptorIndex)).bytes;
    }
    throw new ResolveError("Invalid use of resolveClassName()");
}

/**
 * Returns an appropriate int value for given </code>index</code>
```

```
 * to CONSTANT_Integer.
 */
public int resolveInt(int index) {
    ConstantPoolItem item
        = (ConstantPoolItem)constantItems.elementAt(index);
    if (item.isInteger()) {
        return ((ConstantInteger)item).value;
    }
    throw new ResolveError("Invalid use of resolveClassName()");
}

/**
 * Returns an appropriate float value for given </code>index</code>
 * to CONSTANT_Float.
 */
public float resolveFloat(int index) {
    ConstantPoolItem item
        = (ConstantPoolItem)constantItems.elementAt(index);
    if (item.isFloat()) {
        return ((ConstantFloat)item).value;
    }
    throw new ResolveError("Invalid use of resolveClassName()");
}

/**
 * Returns an appropriate double value for given </code>index</code>
 * to CONSTANT_Double.
 */
public double resolveDouble(int index) {
    ConstantPoolItem item
        = (ConstantPoolItem)constantItems.elementAt(index);
```

```
    if (item.isDouble()) {
        return ((ConstantDouble)item).value;
    }
    throw new ResolveError("Invalid use of resolveClassName()");
}

/**
 * Returns an appropriate long value for given </code>index</code>
 * to CONSTANT_Long.
 */
public long resolveLong(int index) {
    ConstantPoolItem item
        = (ConstantPoolItem)constantItems.elementAt(index);
    if (item.isLong()) {
        return ((ConstantLong)item).value;
    }
    throw new ResolveError("Invalid use of resolveClassName()");
}

/**
 * Returns ConstantPoolItem's index into appropriate item for
 * given constant.
 */
public int lookupUTF8(String value) {
    return ((Integer)utf8s.lookup(value)).intValue();
}

public int lookupInt(int value) {
    return ((Integer)integers.lookup(new Integer(value))).intValue();
}
```

```
public int lookupFloat(float value) {
    return ((Integer)floats.lookup(new Float(value))).intValue();
}

public int lookupLong(long value) {
    return ((Integer)longs.lookup(new Long(value))).intValue();
}

public int lookupDouble(double value) {
    return ((Integer)doubles.lookup(new Double(value))).intValue();
}

public int lookupClass(String name) {
    return ((Integer)classes.lookup(makeClass(lookupUTF8(name))))
        .intValue();
}

public int lookupString(String value) {
    return ((Integer)strings.lookup(makeString(lookupUTF8(value))))
        .intValue();
}

public int lookupNameAndType(String name, String type) {
    return ((Integer)nameAndTypes
            .lookup(makeNameAndType(lookupUTF8(name), lookupUTF8(type))))
        .intValue();
}

public int lookupFieldref(String className, String name, String type) {
    return ((Integer)fieldrefs
            .lookup(makeField(lookupClass(className),
```

```
                                lookupNameAndType(name, type)))).intValue();
    }

    public int lookupMethodref(String className, String name, String type) {
        return ((Integer)methodrefs
                .lookup(makeMethod(lookupClass(className),
                                   lookupNameAndType(name, type)))).intValue();
    }

    public int lookupInterfaceMethodref(String className, String name,
                                        String type) {
        return ((Integer)interfacerefs
                .lookup(makeInterface(lookupClass(className),
                                      lookupNameAndType(name, type))))
            .intValue();
    }
}
```

A.1.5.18 public abstract class ConstantPoolItem implements RuntimeConstants

```
/*
 * $Id: ConstantPoolItem.java,v 1.6 1998/12/15 14:40:35 maruyama Exp $
 */

package OpenJIT.frontend.classfile;

import OpenJIT.frontend.java.RuntimeConstants;
import OpenJIT.frontend.util.IndentedPrintStream;
import java.io.IOException;

/**
```

```
 * Each instances of the subclass of class </code>ConstantPoolItem</code>
 * represent a CONSTANT_*_info structure of which exists in classfile.
 */
public abstract class ConstantPoolItem implements RuntimeConstants {
    protected int tag;

    /**
     * Constructor.
     */
    protected ConstantPoolItem(int tag) {
        this.tag = tag;
    }

    /**
     * Each subclasses of </code>ConstantPoolItem</code> must be override
     * these methods.
     */
    abstract public void write(ClassFileOutputStream stream)
        throws IOException;
    abstract public String toString();
    abstract public void print(IndentedPrintStream out);

    /**
     * For decision of the kind of an subclass's object.
     */
    public final boolean isUTF8() {
        return tag == CONSTANT_UTF8;
    }
    public final boolean isUnicode() {
        return tag == CONSTANT_UNICODE;
    }
```

```
public final boolean isInteger() {
    return tag == CONSTANT_INTEGER;
}
public final boolean isFloat() {
    return tag == CONSTANT_FLOAT;
}
public final boolean isLong() {
    return tag == CONSTANT_LONG;
}
public final boolean isDouble() {
    return tag == CONSTANT_DOUBLE;
}
public final boolean isClass() {
    return tag == CONSTANT_CLASS;
}
public final boolean isString() {
    return tag == CONSTANT_STRING;
}
public final boolean isField() {
    return tag == CONSTANT_FIELD;
}
public final boolean isMethod() {
    return tag == CONSTANT_METHOD;
}
public final boolean isInterface() {
    return tag == CONSTANT_INTERFACEMETHOD;
}
public final boolean isNameAndType() {
    return tag == CONSTANT_NAMEANDTYPE;
}
public final int tag() {
```

```
        return tag;
    }
}
```

A.1.5.19 public abstract class ConstantRef extends ConstantPoolItem

```
/*
 * $Id: ConstantRef.java,v 1.3 1998/11/28 11:13:29 maruyama Exp $
 */

package OpenJIT.frontend.classfile;

import java.io.IOException;

/**
 * Each instances of the subclass of class </code>ConstantRef</code>
 * represent a CONSTANT_*ref_info structure of which exists in classfile.
 * The only differences between those subclasses are thier tags.
 */
public abstract class ConstantRef extends ConstantPoolItem {
    protected int classIndex;
    protected int nameAndTypeIndex;

    /**
     * Constructor.  This is used to construct from parts of classfile.
     */
    protected ConstantRef(int tag, ClassFileInputStream stream)
        throws IOException {
        super(tag);
        classIndex = stream.readU2();
        nameAndTypeIndex = stream.readU2();
    }
```

```
    /**
     * Constructor.  This is used to construct appropriate instance with
     * </code>tag</code>, </code>classIndex</code> and
     * </code>nameAndTypeIndex</code>.
     */
    protected ConstantRef(int tag, int classIndex, int nameAndTypeIndex) {
        super(tag);
        this.classIndex = classIndex;
        this.nameAndTypeIndex = nameAndTypeIndex;
    }

    /**
     * Returns its hashcode calculated from classIndex and nameAndTypeIndex.
     */
    public int hashCode() {
        return (classIndex & 0xff) | ((nameAndTypeIndex & 0xff) << 8) |
            ((classIndex & 0xff00) << 8) | ((nameAndTypeIndex & 0xff00) << 16);
    }

    /**
     * Write this into </code>stream</code> with the style of Java classfile.
     */
    public void write(ClassFileOutputStream stream) throws IOException {
        stream.writeU1(tag);
        stream.writeU2(classIndex);
        stream.writeU2(nameAndTypeIndex);
    }
}
```

A.1.5.20 public class ConstantString extends ConstantPoolItem

```
/*
 * $Id: ConstantString.java,v 1.5 1998/11/28 11:13:29 maruyama Exp $
 */

package OpenJIT.frontend.classfile;

import OpenJIT.frontend.util.IndentedPrintStream;
import java.io.IOException;

/**
 * Each instances of the class </code>ConstantString</code> represent a
 * CONSTANT_String_info structure of which exists in classfile.
 */
public class ConstantString extends ConstantPoolItem {
    protected int stringIndex;

    /**
     * Constructor.  This is used to construct from parts of classfile.
     */
    public ConstantString(ClassFileInputStream stream) throws IOException {
        super(CONSTANT_STRING);
        stringIndex = stream.readU2();
    }

    /**
     * Constructor.  This is used to construct appropriate instance with
     * </code>index</code> into ConstantPool containing Constant_UTF8_info
     * which represent the contents of a </code>String</code> object.
     */
    public ConstantString(int index) {
        super(CONSTANT_STRING);
```

```
    stringIndex = index;
}

/**
 * Returns hashcode calculated from its nameIndex.
 */
public int hashCode() {
    return stringIndex;
}

/**
 * Returns whether the contents are same as given </code>obj</code>.
 */
public boolean equals(Object obj) {
    if (!(obj instanceof ConstantString))
        return false;
    return stringIndex == ((ConstantString)obj).stringIndex;
}

int stringIndex() {
    return stringIndex;
}

/**
 * Write this into </code>stream</code> with the style of Java classfile.
 */
public void write(ClassFileOutputStream stream) throws IOException {
    stream.writeU1(tag);
    stream.writeU2(stringIndex);
}
```

```
    /**
     * Return simple description of this object in a String.
     */
    public String toString() {
        return "string at #" + Integer.toString(stringIndex);
    }

    /**
     * Pretty printer.
     */
    public void print(IndentedPrintStream out) {
        out.println("CONSTANT_String {");
        out.inc();
        out.println("u1 tag = " + Integer.toString(tag) + ";");
        out.println("u2 string_index = "
                    + Integer.toString(stringIndex) + ";");
        out.dec();
        out.println("}");
    }
}
```

A.1.5.21 public class ConstantUTF8 extends ConstantPoolItem

```
/*
 * $Id: ConstantUTF8.java,v 1.6 1998/12/20 18:45:22 maruyama Exp $
 */

package OpenJIT.frontend.classfile;

import OpenJIT.frontend.util.IndentedPrintStream;
import java.io.IOException;
import java.io.UnsupportedEncodingException;
```

```
/**
 * Each instances of the class </code>ConstantUTF8</code> represent a
 * CONSTANT_UTF8_info structure of which exists in classfile.
 */
public class ConstantUTF8 extends ConstantPoolItem {
    protected byte[] bytes;
    protected String string;

    /**
     * Constructor.  This is used to construct from parts of classfile.
     */
    public ConstantUTF8(ClassFileInputStream stream) throws IOException {
        super(CONSTANT_UTF8);
        int length = stream.readU2();
        bytes = stream.readBytes(length);
    }

    /**
     * Constructor.  This is used to construct appropriate instance from
     * </code>String</code> object.
     */
    public ConstantUTF8(String string) {
        super(CONSTANT_UTF8);
        try {
            bytes = string.getBytes("UTF8");
        } catch (UnsupportedEncodingException e) {
            throw new Error("Why UTF8 cannot use");
        }
    }
```

```
/**
 * Write this into </code>stream</code> with the style of Java classfile.
 */
public void write(ClassFileOutputStream stream) throws IOException {
    stream.writeU1(tag);
    stream.writeU2(bytes.length);
    stream.writeBytes(bytes);
}

/**
 * Return simple description of this object in a String.
 */
public synchronized String toString() {
    if (string != null)
        return string;
    try {
        string = new String(bytes, "UTF8");
    } catch (UnsupportedEncodingException e) {
        throw new Error("Why UTF8 cannot use");
    }
    return string;
}

/**
 * Pretty printer.
 */
public void print(IndentedPrintStream out) {
    out.println("CONSTANT_Utf8 {");
    out.inc();
    out.println("u1 tag = " + Integer.toString(tag) + ";");
    out.println("u2 length = "
```

```
                    + Integer.toString(bytes.length) + ";");
        out.println("u1 bytes[] = \"" + toString() + "\";");
        out.dec();
        out.println("}");
    }

    /**
     * Returns hashCode calculated from bytes.
     */
    public int hashCode() {
        return toString().hashCode();
    }

    public byte[] bytes() {
        return bytes;
    }
}
```

A.1.5.22 public class ConstantValueAttribute extends AttributeInfo

```
/*
 * $Id: ConstantValueAttribute.java,v 1.7 1998/12/20 18:45:22 maruyama Exp $
 */

package OpenJIT.frontend.classfile;

import OpenJIT.frontend.util.IndentedPrintStream;
import java.io.IOException;

/**
 * Each instances of the class </code>ConstantValueAttribute</code> represent
 * that field containing it is a constant field.
```

```
 */
public class ConstantValueAttribute extends AttributeInfo {
    /**
     * It holds an index into ConstantPool that represent the constant value.
     */
    protected int constantValueIndex;

    /**
     * Constructor.  This is used to construct from parts of classfile.
     */
    public ConstantValueAttribute(int nameIndex, ClassFileInputStream stream,
                                  ClassFile cF) throws IOException {
        super(nameIndex, stream, cF);
        constantValueIndex = stream.readU2();
    }

    /**
     * Write this into </code>stream</code> with the style of Java classfile.
     */
    public void write(ClassFileOutputStream stream) throws IOException {
        super.write(stream);
        stream.writeU2(constantValueIndex);
    }

    /**
     * Return simple description of this object in a String.
     */
    public String toString() {
        return "ConstantValue_attribute { " + super.toString()
            + ", constantvalue_index = " + Integer.toString(constantValueIndex)
            + " }";
```

```
    }

    /**
     * Pretty printer.
     */
    public void print(IndentedPrintStream out) {
        out.println("ConstantValue_attribute {");
        out.inc();
        out.println("u2 attribute_name_index = "
                    + Integer.toString(attributeNameIndex) + ";");
        out.println("u4 attribute_length = "
                    + Integer.toString(attributeLength) + ";");
        out.println("u2 constantvalue_index = "
                    + Integer.toString(constantValueIndex) + "; // "
                    + classFile.constantPool
                          .resolveString(constantValueIndex));
        out.dec();
        out.println("}");
    }

    public int constantValueIndex() {
        return constantValueIndex;
    }
}
```

A.1.5.23 public class ExceptionsAttribute extends AttributeInfo

```
/*
 * $Id: ExceptionsAttribute.java,v 1.6 1998/12/15 14:40:36 maruyama Exp $
 */
```

```
package OpenJIT.frontend.classfile;

import OpenJIT.frontend.util.IndentedPrintStream;
import java.io.IOException;

/**
 * Each instances of the class </code>ExceptionsAttribute</code> represent
 * a set of exceptions that method containing it may throw.
 */
public class ExceptionsAttribute extends AttributeInfo {
    /**
     * Each content of </code>exceptionIndexTable[]</code> is an index
     * into ConstantPool that represent a Class of the exception.
     */
    protected int exceptionIndexTable[];

    /**
     * Constructor.  This is used to construct from parts of classfile.
     */
    public ExceptionsAttribute(int nameIndex, ClassFileInputStream stream,
                               ClassFile cF) throws IOException {
        super(nameIndex, stream, cF);
        int length = stream.readU2();
        exceptionIndexTable = new int[length];
        for (int i = 0; i < length; i++)
            exceptionIndexTable[i] = stream.readU2();
    }

    /**
     * Write this into </code>stream</code> with the style of Java classfile.
     */
```

```
public void write(ClassFileOutputStream stream) throws IOException {
    super.write(stream);
    int length = exceptionIndexTable.length;
    stream.writeU2(length);
    for (int i = 0; i < length; i++)
        stream.writeU2(exceptionIndexTable[i]);
}

/**
 * Return simple description of this object in a String.
 */
public String toString() {
    return "Exceptions_attribute { " + super.toString()
        + ", number_of_exceptions = "
        + Integer.toString(exceptionIndexTable.length)
        + ", exception_index_table[] }";
}

/**
 * Pretty printer.
 */
public void print(IndentedPrintStream out) {
    out.println("Exceptions_attribute {");
    out.inc();
    out.println("u2 attribute_name_index = "
                + Integer.toString(attributeNameIndex) + ";");
    out.println("u4 attribute_length = "
                + Integer.toString(attributeLength) + ";");
    int count = exceptionIndexTable.length;
    out.println("u2 number_of_exceptions = "
                + Integer.toString(count) + ";");
```

```
        if (count > 0) {
            StringBuffer buf = new StringBuffer();
            buf.append("exception_index_table[] = { ");
            buf.append(classFile.constantPool
                       .resolveString(exceptionIndexTable[0]));
            for (int i = 1; i < count; i++) {
                buf.append(", ");
                buf.append(classFile.constantPool
                           .resolveString(exceptionIndexTable[i]));
            }
            buf.append(" };");
            out.println(buf.toString());
        } else
            out.println("exception_index_table[];");
        out.dec();
        out.println("}");
    }
}
```

A.1.5.24 public class FieldInfo extends MemberInfo

```
/*
 * $Id: FieldInfo.java,v 1.6 1998/12/01 20:27:48 maruyama Exp $
 */

package OpenJIT.frontend.classfile;

import java.io.IOException;

/**
 * Each instances of the class </code>FieldInfo</code> represent a
 * set of informations for a field contained in a classfile.
```

```
 * Its contents are just same as methods' information, so its are
 * treated as </code>MemberInfo</code> -- superclass of FieldInfo.
 */
public class FieldInfo extends MemberInfo {
    /**
     * Constructor.  This is used to construct from parts of classfile.
     */
    public FieldInfo(ClassFileInputStream stream, ClassFile cF)
        throws IOException {
        super(stream, cF);
    }

    /**
     * Return simple description of this object in a String.
     */
    public String toString() {
        return "field" + super.toString();
    }

    /**
     * used by MethodInfo for pretty printing.
     */
    public String header() {
        return "field_info";
    }
}
```

A.1.5.25 public class GenericAttribute extends AttributeInfo

```
/*
 * $Id: GenericAttribute.java,v 1.4 1998/11/26 12:36:09 maruyama Exp $
 */
```

```
package OpenJIT.frontend.classfile;

import OpenJIT.frontend.util.IndentedPrintStream;
import java.io.IOException;

public class GenericAttribute extends AttributeInfo {
    protected byte info[];

    public GenericAttribute(int nameIndex, ClassFileInputStream stream,
                            ClassFile cF) throws IOException {
        super(nameIndex, stream, cF);
        info = stream.readBytes(attributeLength);
    }

    public void write(ClassFileOutputStream stream) throws IOException {
        super.write(stream);
        int length = info.length;
        for (int i = 0; i < length; i++)
            stream.writeU1(info[i]);
    }

    public String toString() {
        return "attribute_info { " + super.toString()
            + ", info[] }";
    }

    public void print(IndentedPrintStream out) {
        out.println("Generic_attribute {");
        out.inc();
        out.println("u2 attribute_name_index = "
```

```
                    + Integer.toString(attributeNameIndex) + ";");
        out.println("u4 attribute_length = "
                    + Integer.toString(attributeLength) + ";");
        out.println("u1 info[];");
        out.dec();
        out.println("}");
    }
}
```

A.1.5.26 public class LineNumber

```
/*
 * $Id: LineNumber.java,v 1.3 1998/12/01 20:27:48 maruyama Exp $
 */

package OpenJIT.frontend.classfile;

import OpenJIT.frontend.util.IndentedPrintStream;
import java.io.IOException;

/**
 * Each instances of the class </code>LineNumber</code> represent a set
 * of informations of line number.
 */
public class LineNumber {
    protected int startPC;
    protected int lineNumber;

    /**
     * Constructor.  This is used to construct from parts of classfile.
     */
    public LineNumber(ClassFileInputStream stream) throws IOException {
```

```
        startPC = stream.readU2();
        lineNumber = stream.readU2();
    }

    /**
     * Write this into </code>stream</code> with the style of Java classfile.
     */
    public void write(ClassFileOutputStream stream) throws IOException {
        stream.writeU2(startPC);
        stream.writeU2(lineNumber);
    }

    /**
     * Return simple description of this object in a String.
     */
    public String toString() {
        return "line_number { " + Integer.toString(startPC)
            + ", " + Integer.toString(lineNumber) + " }";
    }

    /**
     * Pretty printer.
     */
    public void print(IndentedPrintStream out) {
        out.println(toString());
    }
}
```

A.1.5.27 public class LineNumberTableAttribute extends AttributeInfo

```
/*
 * $Id: LineNumberTableAttribute.java,v 1.4 1998/12/01 20:27:48 maruyama Exp $
```

```
 */

package OpenJIT.frontend.classfile;

import OpenJIT.frontend.util.IndentedPrintStream;
import java.io.IOException;

/**
 * Each instances of the class </code>LineNumberTableAttribute</code> represent
 * a set of informations for debugging purpose -- line number.
 */
public class LineNumberTableAttribute extends AttributeInfo {
    protected LineNumber lineNumberTable[];

    /**
     * Constructor.  This is used to construct from parts of classfile.
     */
    public LineNumberTableAttribute(int nameIndex, ClassFileInputStream stream,
                                    ClassFile cF) throws IOException {
        super(nameIndex, stream, cF);
        int count = stream.readU2();
        lineNumberTable = new LineNumber[count];
        for (int i = 0; i < count; i++)
            lineNumberTable[i] = new LineNumber(stream);
    }

    /**
     * Write this into </code>stream</code> with the style of Java classfile.
     */
    public void write(ClassFileOutputStream stream) throws IOException {
        super.write(stream);
```

```
    int count = lineNumberTable.length;
    stream.writeU2(count);
    for (int i = 0; i < count; i++)
        lineNumberTable[i].write(stream);
}

/**
 * Return simple description of this object in a String.
 */
public String toString() {
    return "LineNumberTable_attribute { " + super.toString()
        + ", line_number_table_length = "
        + Integer.toString(lineNumberTable.length)
        + ", line_number_table[] }";
}

/**
 * Pretty printer.
 */
public void print(IndentedPrintStream out) {
    out.println("LineNumberTable_attribute {");
    out.inc();
    out.println("u2 attribute_name_index = "
                + Integer.toString(attributeNameIndex) + ";");
    out.println("u4 attribute_length = "
                + Integer.toString(attributeLength) + ";");
    int count = lineNumberTable.length;
    out.println("u2 line_number_table_length = "
                + Integer.toString(count) + ";");
    if (count > 0) {
        out.println("line_number_table[] = {");
```

```
            out.inc();
            for (int i = 0; i < count; i++)
                lineNumberTable[i].print(out);
            out.dec();
            out.println("}");
        } else
            out.println("line_number_table[];");
        out.dec();
        out.println("}");
    }
}
```

A.1.5.28 public class LocalVariable

```
/*
 * $Id: LocalVariable.java,v 1.5 1998/12/15 14:40:36 maruyama Exp $
 */

package OpenJIT.frontend.classfile;

import OpenJIT.frontend.util.IndentedPrintStream;
import java.io.IOException;

/**
 * Each instances of the class </code>LocalVariable</code> represent a set
 * of informations of local variable.
 */
public class LocalVariable {
    protected ClassFile classFile;

    protected int startPC;
    protected int length;
```

```
protected int nameIndex;
protected int descriptorIndex;
protected int index;

/**
 * Constructor.  This is used to construct from parts of classfile.
 */
public LocalVariable(ClassFileInputStream stream, ClassFile cF)
    throws IOException {
    classFile = cF;
    startPC = stream.readU2();
    length = stream.readU2();
    nameIndex = stream.readU2();
    descriptorIndex = stream.readU2();
    index = stream.readU2();
}

/**
 * Write this into </code>stream</code> with the style of Java classfile.
 */
public void write(ClassFileOutputStream stream) throws IOException {
    stream.writeU2(startPC);
    stream.writeU2(length);
    stream.writeU2(nameIndex);
    stream.writeU2(descriptorIndex);
    stream.writeU2(index);
}

/**
 * Return simple description of this object in a String.
 */
```

```
    public String toString() {
        return "local_variable { " + Integer.toString(startPC)
            + ", " + Integer.toString(length)
            + ", " + Integer.toString(nameIndex)
            + ", " + Integer.toString(descriptorIndex)
            + ", " + Integer.toString(index) + " }";
    }

    /**
     * Pretty printer.
     */
    public void print(IndentedPrintStream out) {
        out.println("local_variable {");
        out.inc();
        out.println("u2 start_pc = "
                    + Integer.toString(startPC) + ";");
        out.println("u2 length = " + Integer.toString(length) + ";");
        out.println("u2 name_index = " + Integer.toString(nameIndex) + "; // "
                    + classFile.constantPool.resolveString(nameIndex));
        out.println("u2 descriptor_index = "
                    + Integer.toString(descriptorIndex) + "; // "
                    + classFile.constantPool.resolveString(descriptorIndex));
        out.println("u2 index = " + Integer.toString(index) + ";");
        out.dec();
        out.println("}");
    }
}
```

A.1.5.29 public class LocalVariableTableAttribute extends AttributeInfo

```
/*
 * $Id: LocalVariableTableAttribute.java,v 1.4 1998/12/01 20:27:49 maruyama Exp
```

```
 */

package OpenJIT.frontend.classfile;

import OpenJIT.frontend.util.IndentedPrintStream;
import java.io.IOException;

/**
 * Each instances of the class </code>LocalVariableTableAttribute</code>
 * represent a set of informations for debugging purpose -- local variable.
 */
public class LocalVariableTableAttribute extends AttributeInfo {
    protected LocalVariable localVariableTable[];

    /**
     * Constructor.  This is used to construct from parts of classfile.
     */
    public LocalVariableTableAttribute(int nameIndex,
                                       ClassFileInputStream stream,
                                       ClassFile cF) throws IOException {
        super(nameIndex, stream, cF);
        int count = stream.readU2();
        localVariableTable = new LocalVariable[count];
        for (int i = 0; i < count; i++)
            localVariableTable[i] = new LocalVariable(stream, cF);
    }

    /**
     * Write this into </code>stream</code> with the style of Java classfile.
     */
    public void write(ClassFileOutputStream stream) throws IOException {
```

```
    super.write(stream);
    int count = localVariableTable.length;
    stream.writeU2(count);
    for (int i = 0; i < count; i++)
        localVariableTable[i].write(stream);
}

/**
 * Return simple description of this object in a String.
 */
public String toString() {
    return "LocalVariableTable_attribute { " + super.toString()
        + ", local_variable_table_length = "
        + Integer.toString(localVariableTable.length)
        + ", local_variable_table[] }";
}

/**
 * Pretty printer.
 */
public void print(IndentedPrintStream out) {
    out.println("LocalVariableTable_attribute {");
    out.inc();
    out.println("u2 attribute_name_index = "
                + Integer.toString(attributeNameIndex) + ";");
    out.println("u4 attribute_length = "
                + Integer.toString(attributeLength) + ";");
    int count = localVariableTable.length;
    out.println("u2 local_variable_table_length = "
                + Integer.toString(count) + ";");
    if (count > 0) {
```

```
            out.println("local_variable_table[] = {");
            out.inc();
            for (int i = 0; i < count; i++)
                localVariableTable[i].print(out);
            out.dec();
            out.println("}");
        } else
            out.println("local_variable_table[];");
        out.dec();
        out.println("}");
    }
}
```

A.1.5.30 public abstract class MemberInfo

```
/*
 * $Id: MemberInfo.java,v 1.8 1998/12/15 14:40:36 maruyama Exp $
 */

package OpenJIT.frontend.classfile;

import OpenJIT.frontend.util.IndentedPrintStream;
import java.io.IOException;

/**
 * Each instances of the class </code>MemberInfo</code> represent a
 * set of informations for a field/method contained in a classfile.
 */
public abstract class MemberInfo {
    protected int accessFlags;
    protected int nameIndex;
    protected int descriptorIndex;
```

```
protected Attributes attributes;
protected ClassFile classFile;

public MemberInfo(ClassFileInputStream stream, ClassFile cF)
    throws IOException {
    accessFlags = stream.readU2();
    nameIndex = stream.readU2();
    descriptorIndex = stream.readU2();
    classFile = cF;
    attributes = cF.newAttributes(stream, cF);
}

public void write(ClassFileOutputStream stream) throws IOException {
    stream.writeU2(accessFlags);
    stream.writeU2(nameIndex);
    stream.writeU2(descriptorIndex);
    attributes.write(stream);
}

public String toString() {
    return " { access_flags = " + Integer.toString(accessFlags)
        + ", name_index = " + Integer.toString(nameIndex)
        + ", descriptor_index = " + Integer.toString(descriptorIndex)
        + " };";
}
abstract public String header();

public void print(IndentedPrintStream out) {
    out.println(header() + " {");
    out.inc();
    out.println("u2 access_flags = 0x"
```

```
                    + Integer.toHexString(accessFlags) + ";");
        out.println("u2 name_index = " + Integer.toString(nameIndex) + "; // "
                    + classFile.constantPool.resolveString(nameIndex));
        out.println("u2 descriptor_index = "
                    + Integer.toString(descriptorIndex) + "; // "
                    + classFile.constantPool.resolveString(descriptorIndex));
        attributes.print(out);
        out.dec();
        out.println("}");
    }
    /**
     * Accessors
     */
    public int accessFlags() {
        return accessFlags;
    }
    public int nameIndex() {
        return nameIndex;
    }
    public int descriptorIndex() {
        return descriptorIndex;
    }
    public Attributes attributes() {
        return attributes;
    }
    public ClassFile classFile() {
        return classFile;
    }
}
```

A.1.5.31 public class MethodInfo extends MemberInfo

```
/*
 * $Id: MethodInfo.java,v 1.6 1998/12/01 20:27:49 maruyama Exp $
 */

package OpenJIT.frontend.classfile;

import java.io.IOException;

/**
 * Each instances of the class </code>MethodInfo</code> represent a
 * set of informations for a method contained in a classfile.
 * Its contents are just same as fields' information, so its are
 * treated as </code>MemberInfo</code> -- superclass of MethodInfo.
 */
public class MethodInfo extends MemberInfo {
    public MethodInfo(ClassFileInputStream stream, ClassFile cF)
        throws IOException {
        super(stream, cF);
    }

    public String toString() {
        return "method" + super.toString();
    }

    public String header() {
        return "method_info";
    }
}
```

A.1.5.32 public class ResolveError extends Error

```
/*
```

```
 * $Id: ResolveError.java,v 1.2 1998/12/15 14:40:36 maruyama Exp $
 */

package OpenJIT.frontend.classfile;

public class ResolveError extends Error {
    Throwable e;

    public ResolveError(String msg) {
        super(msg);
        this.e = this;
    }

    public ResolveError(Exception e) {
        super(e.getMessage());
        this.e = e;
    }
}
```

A.1.5.33 public class SourceFileAttribute extends AttributeInfo

```
/*
 * $Id: SourceFileAttribute.java,v 1.6 1998/12/15 14:40:36 maruyama Exp $
 */

package OpenJIT.frontend.classfile;

import OpenJIT.frontend.util.IndentedPrintStream;
import java.io.IOException;

/**
 * Each instances of the class </code>SourceFileAttribute</code> represent a
```

```
 * file name of the source file.
 */
public class SourceFileAttribute extends AttributeInfo {
    /**
     * It holds an index into ConstantPool that represent the filename.
     */
    protected int sourceFileIndex;

    /**
     * Constructor.  This is used to construct from parts of classfile.
     */
    public SourceFileAttribute(int nameIndex, ClassFileInputStream stream,
                               ClassFile cF) throws IOException {
        super(nameIndex, stream, cF);
        sourceFileIndex = stream.readU2();
    }

    /**
     * Write this into </code>stream</code> with the style of Java classfile.
     */
    public void write(ClassFileOutputStream stream) throws IOException {
        super.write(stream);
        stream.writeU2(sourceFileIndex);
    }

    /**
     * Return simple description of this object in a String.
     */
    public String toString() {
        return "SourceFile_attribute { " + super.toString()
            + ", sourcefile_index = " + Integer.toString(sourceFileIndex)
```

```
            + " }";
    }

    /**
     * Pretty printer.
     */
    public void print(IndentedPrintStream out) {
        out.println("SourceFile_attribute {");
        out.inc();
        out.println("u2 attribute_name_index = "
                    + Integer.toString(attributeNameIndex) + ";");
        out.println("u4 attribute_length = "
                    + Integer.toString(attributeLength) + ";");
        out.println("u2 sourcefile_index = "
                    + Integer.toString(sourceFileIndex) + "; // "
                    + classFile.constantPool.resolveString(sourceFileIndex));
        out.dec();
        out.println("}");
    }
}
```

A.1.5.34 public class UnknownFileException extends Error

```
/*
 * $Id: UnknownFileException.java,v 1.2 1998/11/24 18:14:01 maruyama Exp $
 */

package OpenJIT.frontend.classfile;

public class UnknownFileException extends Error {
    Throwable e;
```

```
    public UnknownFileException(String msg) {
        super(msg);
        this.e = this;
    }

    public UnknownFileException(Exception e) {
        super(e.getMessage());
        this.e = e;
    }
}
```

A.2 OpenJIT バックエンドシステム

A.2.1 メソッド情報取得試験用クラス

```
package OpenJIT;
class Test extends Compile {
    void parseBytecode() {
        System.out.println(this);
        System.exit(0);
    }
    void convertRTL() {}
    void optimizeRTL() {}
    void genNativeCode() {}
    void regAlloc() {}
}
```

A.2.2 バイトコードアクセス試験用クラス

```
package OpenJIT;
class Test extends Compile {
    void parseBytecode() { }
    void convertRTL() {}
    void optimizeRTL() {}
    void genNativeCode() {}
    void regAlloc() {}
    public boolean compile() {
      if (clazz.getName().regionMatches(0, "java.lang.Thread", 0, 16)) {
          System.out.println(this + " size:" + bytecode.length);
      }
      return false;
    }
}
```

A.2.3 生成コードメモリ管理試験用クラス

```
package OpenJIT;
class Test extends Compile {
    void parseBytecode() { }
    void convertRTL() {}
    void optimizeRTL() {}
    void genNativeCode() {}
    void regAlloc() {}
    public boolean compile() {
        if (clazz != this.getClass()) return false;
        NativeCodeAlloc(65536 * 4);
        for (int pc = 0; pc < 65536; pc++) {
            setNativeCode(pc, pc);
        }
        for (int pc = 0; pc < 65536; pc++) {
            if (getNativeCode(pc) != pc)
                return false;
        }
        System.out.println("Test passed.");
        return false;
    }
}
```

A.2.4 中間言語変換試験用クラス

```
package OpenJIT;
class Test extends ParseBytecode {
    void convertRTL() {}
    void optimizeRTL() {}
    void genNativeCode() {}
    void regAlloc() {}
    public boolean compile() {
        if (clazz != this.getClass()) return false;
        bcinfo = new BCinfo [bytecode.length];
        parseBytecode();
        System.out.println("Method:" + this);
        for (int pc = 0; pc < bytecode.length; pc++) {
            BCinfo bc = bcinfo[pc];
            if (bc == null) continue;
            System.out.println(pc + "\t" + opcName(pc));
            for (ILnode il = bc.next; il != null; il = il.next) {
                System.out.println("\t" + il);
            }
        }
        return false;
    }
}
```

A.2.5 バックエンド中間コード試験用クラス

```
package OpenJIT;
class Test extends ParseBytecode {
    void convertRTL() {}
    void optimizeRTL() {}
    void genNativeCode() {}
    void regAlloc() {}
    public boolean compile() {
        if (methodName().compareTo("test") != 0) return false;
        bcinfo = new BCinfo [bytecode.length];
        parseBytecode();
        System.out.println("Method:" + this);
        for (int pc = 0; pc < bytecode.length; pc++) {
            BCinfo bc = bcinfo[pc];
            if (bc == null) continue;
            System.out.println(pc + "\t" + opcName(pc));
            for (ILnode il = bc.next; il != null; il = il.next) {
                System.out.println("\t" + il);
            }
        }
        return false;
    }
}
```

A.2.6 コントロールフロー解析試験用クラス

```
class test {
    public static void main(String argv[]) { test(1); }
    static int m(int a, int b, int c, int d) { return a+b+c+d; }
    static int test(int x) {
        return m(1, 2, 3, (x > 4) ? 5 : 6);
    }
}
```