卒業研究報告書

題 目

VSPMの構築

指 導 教 員

石 水　隆 助手

報 告 者

02–1–47–173

森 本　尚 樹

近畿大学理工学部情報学科

平成 18 年 2 月 10 日提出

概要

本研究ではVSPM(Virtual Parallel Stack Machine) の設計を行う. 多くの場合、並列アルゴリズムの設計やその計算量の評価はPRAM(Parallel Random Access Machine) 上で行われる.PRAM は共有メモリ型並列計算モデルであり、個々の演算による実行時間の違いや通信や同期のコストを無視した単純なモデルであるため、アルゴリズムの設計および評価を行い易いためである。しかし、大規模なプロセッサでのメモリの共有化や、通信や同期の高速化には様々な問題があるため、PRAM 自体の実現は困難である。そこで、PRAM アルゴリズムの実験的な評価を示すためにPRAM シミュレータが必要になるのだが、本研究で設計したVPSM はその PRAM シミュレータの一部となっている。

PRAM アルゴリズムの実行をシミュレートするPRAM シミュレータは以下の 4 要素から成る。(1) PRAM 用並列言語。(2) 並列アセンブラ。(3) PRAM コンパイラ。(4) VPSM (Virtual Parallel Stack Machine)。PRAM シミュレータのユーザは、PRAM 用並列言語で PRAM 上での並列処理を記述し、PRAM コンパイラにより並列アセンブラに変換する。次に VPSM 上で並列アセンブラを実行することにより PRAM の動作をシミュレートできる。

本研究で設計したVPSM は逐次状態と並列状態の 2 つの状態を持つ. 逐次状態では 1 台のスタックマシンのみが動作を行い、並列状態では、複数のスタックマシンが動作を行う. 並列アセンブラは、通常のアセンブラ命令に加えて並列処理の開始を表す命令 PARA、並列処理の終了とプロセッサ間の同期を表す命令 SYNC、プロセッサ番号をスタックに入れる命令 PUSHP を持つ。初期状態においてはVPSM は逐次状態にあり、命令 PARA を読み込むと並列状態に移行し、複数のスタックマシンが起動される。並列状態において命令 SYNC を読み込むと、ずべてのスタックマシンが SYNC に到着するまで処理を中断し、その後逐次状態に移行するようになっている。

本研究で設計したVPSM は、並列アセンブラの実行をシミュレートすることができる。また、PRAM 用並列言語を PRAM 上で実行させたときの実行時間を出力する機能を持っているため、PRAM アルゴリズムの計算量を実験的に評価することができる。

目 次

1 序論

1.1 本研究の背景

1.1.1 並列アルゴリズム

多くの分野で、地球規模の気象シミュレーションや天体の軌道計算など、計算量の大きな問題を短時間で解くことが必要とされている。これらの問題に対して、従来の 1 台のプロセッサから成る逐次計算機を用いた逐次処理では非常に大きな時間が掛かる。このため現在では、より適した手法として複数のプロセッサを持つ並列計算機 (Parallel Computer) による並列処理 (Parallel Processing) が注目されている。複数のプロセッサが協調してデータを処理することにより、計算量の大きな問題を短時間で解け、また、より複雑な問題を解くことができるようになる。しかし、並列処理を行うためには、プロセッサ間のデータのやり取りやメモリへのアクセス、プロセッサ間の同期等、並列特有の問題を解決せねばならない。このため、従来の逐次処理で用いられてきた逐次アルゴリズムをそのまま並列処理に用いることはできず、並列処理専用のアルゴリズム、すなわち並列アルゴリズム (Parallel Algorithm) が必要となる。現在様々な分野で、高速に処理を行う並列アルゴリズムが求められている。

1.1.2 並列計算機

並列計算機は複数のプロセッサを持ち並列処理を行うことができる計算機である。並列計算機は、全てのプロセッサが共通したメモリに対して読み書きを行い、プロセッサ間の通信はメモリを通して行う共有メモリ型並列計算機と、それぞれのプロセッサが局所メモリを持ち、プロセッサ間の通信はネットワークを通じて行う分散メモリ型並列計算機に大別される。共有メモリ型計算機はプロセッサ間の通信を高速に行うことができ、プロセッサ間での同期も取り易いため、通信および同期にかかるコストを気にせずに高速化を得ることができる。プロセッサ数の増加に従い、1 つの共有メモリに全てのプロセッサを繋ぐことは困難となる。このため、現在、プロセッサ数の多い並列計算機では分散メモリ型が主流となっている。また、複数の計算機をネットワークで繋ぎ、それ全体を仮想的な計算機として扱うクラスタ (Cluster) 処理やグリッド (Grid) 処理も幅広く行われている。

1.1.3 並列計算モデル

列アルゴリズムの設計およびその計算量の評価は多くの場合 PRAM(Parallel Random Access Machine) 上で行われる。

PRAM は共有メモリ型並列計算モデルであり、全ての演算が 1 単位時間で行われる、1 命令毎に同期が取られる、通信のコストが一切発生しない、等の仮定が設けられた理想的なモデルである。PRAM は個々の演算による実行時間の違いや通信や同期のコストを無視しした単純なモデルであるため、PRAM 上では並列アルゴリズムの設計および計算量の評価を比較的容易に行うことができる。しかし、大規模なプロセッサでのメモリの共有化や、通信や同期の高速化には様々な問題があるため、PRAM 自体の実現は困難である。

1.2 本研究の目的

多くの並列アルゴリズムは、PRAM 上で設計・解析が行われる。しかし、PRAM 自体の実現は困難であるため、PRAM アルゴリズムの正当性および時間計算量を実験的に評価するのは容易ではない。そのため、PRAM アルゴリズムの実験的な評価を行うために PRAM シミュレータ (PRAM Simulater) が必要となる。

PRAM シミュレータは、高級言語である PRAM 用並列言語を低級言語である並列アセンブラに変換する PRAM コンパイラと、並列アセンブラを実行する VPSM(Virtual Parallel Stack Machine) インタプリタから成る。

本研究では、JAVA 言語を用いて VPSM インタプリタの設計を行う。本研究で設計した VPSM インタプリタは、実行する並列アセンブラプログラムの原始プログラムである PRAM 用並列言語プログラムの PRAM 上での実行をシミュレートし、その結果を出力する。また、本研究で設計した VPSM インタプリタは、PRAM 用並列言語プログラムを PRAM 上で動作させた場合の実行時間を計測する機能を持ち、PRAM アルゴリズムの計算量を実験的に評価することができる。

1.3 本報告書の構成

2 節では、VPSM の設計の準備として、PRAM、PRAM シミュレータの詳細について表記する。3 節では、研究の本題となる VPSMI インタプリタの詳細、使用方法について表記する。4 節以降では、本研究での結果、結論などを記す。また、最後に VPSM インタプリタに必要となったプログラムのソースなどを付録として付けることにする。

2　準備

2.1　PRAM

図 1 のように、複数のプロセッサがメモリを共有したコンピュータを共有メモリ型並列コンピュータ (Shared Memory Parallel Computer) と呼ぶ。代表的な並列計算モデルである PRAM(Parallel Random Access Machine) は共有メモリ型並列計算機を抽象化したモデルである。

図 1: 共有メモリ型並列コンピュータ

PRAM は共有メモリとそれに接続された P 個のプロセッサからなる。並列アルゴリズムの実行中、P 個のプロセッサは入力データの読み出し、中間結果の読み出し、および書き込み、最終結果の書き込みをするために、共有メモリにアクセスする。PRAM の各プロセッサは、共有メモリ上の任意の位置にあるメモリセルに対して 1 単位時間でデータの読み書きができ、また全ての演算は 1 単位時間で行うことができると仮定されている。また、1 単位時間ごとに全てのプロセッサで同期がとられる完全同期型モデルである。

複数のプロセッサによる異なる位置のメモリセルへのアクセスに対しては全てのプロセッサが自由に読み書きを行うことができる。一方、複数のプロ

セッサによる同一セルへのアクセスについてはそれをどう処理するかにより PRAM が以下の 4 つに分類される。

1. **排他的読み出し排他的書き込み（EREW:Exclusive-Read Exclusive-Write）PRAM**

   メモリ位置へアクセスは排他的である。どの 2 つのプロセッサも同じメモリ位置から同時に読み出したり書き込んだりできない。

2. **同時読み出し排他的書き込み（CREW:Concurrent-Read Exclusive-Write）PRAM**

   複数のプロセッサが同時に同じメモリ位置から読み出すことができるが、書き込みの権利は排他的であり、どの 2 つのプロセッサも同じメモリ位置に同時に書き込むことはできない。

3. **排他的読み出し同時書き込み（ERCW:Exclusive-Read Concurrent-Write）PRAM**

   複数のプロセッサが同時に同じメモリ位置に書き込むことができるが、読み出しは排他的である。

4. **同時読み出し同時書き込み（CRCW:Concurrent-Read Concurrent-Write）PRAM**

   複数のプロセッサによる同じメモリ位置への同時読み出し、同時書き込みが認められている。

さらに、CRCWPRAM は同時書き込みが行われる際の処理方法で以下の 3 種に分類される。

( ) 優先型 (Priority)CRCWPRAM
同時書き込みが起こった時、最も優先順位が高いものが書き込みに成功する。
( ) 任意型 (Arbitrary)CRCWPRAM
同時書き込みが起こった時、どれか 1 つが書き込みに成功する。
( ) 共通型 (Common)CRCWPRAM
同時書き込みが起こった時、全てが同じ値を書き込もうとした時に成功し、その他はエラーとする。

2.2 PRAM シミュレータ

PRAM シミュレータは PRAM 上での並列アルゴリズムの実行をシミュレートを行い、その実行結果を出力しおよび実行時間を計測する機能を持つプログラムである。PRAM シミュレータは以下の 4 要素から成る

1. **PRAM 用並列言語**

2. **並列アセンブラ**

3. **PRAM コンパイラ**

4. **VPSM(Virtual Parallel Stack Machine) インタプリタ**

PRAM 用並列言語は、JAVA や C 等の高級言語に並列処理を行うための命令を加えたものであり、並列アセンブラは並列処理を行うための命令を加えたアセンブラである。PRAM コンパイラは、PRAM 用並列言語で記述されたプログラムを並列アセンブラプログラムに変換するコンパイラであり、VPSM インタプリタは並列アセンブラプログラムを実行できるインタプリタである。図 2 に PRAM シミュレータの実行の流れを示す。ユーザは PRAM 用並列言語を用いて PRAM 用並列アルゴリズムを記述する。次に PRAM 用並列言語プログラムを PRAM コンパイラを用いて並列アセンブラプログラムに変換し、VPSM インタプリタを用いて並列アセンブラプログラムを実行する。

図 2: PRAM シミュレータの実行の流れ

3 PRAM シミュレータ

3.1 PRAM 用並列言語

本研究で使用する PRAM 用並列言語は、C 風の手続き型言語である。この言語は、KC05 言語 [1] に並列処理を行う **parallel** 文および実行中のプロセッサ番号を表す特殊変数**$p** を加えたものである。*parallel* 文の文法は以下のように定義される

parallel (*exp1* , *exp2*) *statement*

exp1 および *exp2* は int 型の評価値を持つ式であり、*statement* は任意の文である。**parallel** 文を実行すると、プロセッサ番号 *exp1* から *exp2* までのプロセッサが後に続く *statement* を並列に実行する。また、*statement* 中に特殊変数**$p** を記述すると**$p** は実行中のプロセッサ番号の値を持つ変数として処理される。

PRAM 用並列言語プログラムは、プロセッサ P_0 のみが命令を実行する逐次モードと、**parallel** 文により指定された複数のプロセッサが命令を実行する並列モードの 2 つのモードを持つ。プログラム開始時は逐次モードであり、**parallel** 命令中の *statement* は並列モードとして実行され、それ以外の命令は逐次モードとして実行される。

3.2 並列アセンブラ

本研究で使用する並列アセンブラは、命令部オペレータと値部オペランドから成る 2 個組命令で構成される。この言語は、KC05 言語の目的言語として用いられるアセンブラ [1] の命令セットに以下の命令を加えたものである。

- **PUSHP**：スタックトップに命令を実行しているプロセッサのプロセッサ番号を挿入する
- **PARA**：逐次モードから並列モードに移行する。逐次モード実行中に **PARA** 命令を読み込んだ場合、スタックから 2 個のデータ d_1, d_2 が取り出される。この命令以降は、プロセッサ $P_{d_1}, P_{d_1+1}, P_{d_1+2}, ..., P_{d_2}$ が並列に命令を実行する。並列モード中に **PARA** 命令を読み込んだ場合、実行時エラーとなる。
- **SYNC**：並列モード実行中のプロセッサ間で同期を取り、逐次モードに移行する。並列モード中に **SYNC** 命令を読み込んだ場合、並列モード実行中の全てのプロセッサが **SYNC** 命令に到達するまで実行を中断する。全てのプロセッサが **SYNC** に到達すると、それ以降はプロセッサ P_0 のみが命令を実行する。逐次モード中に **SYNC** 命令が読み込まれると実行時エラーとなる。

付録 A.1 に本研究で使用する並列アセンブラの文法を示す

3.3 VPSM(Virtual Parallel Stack Machine) インタプリタ

本研究では JAVA 言語を用いて VPSM インタプリタの設計を行った。付録 A.2 に本研究で作成した VPSM インタプリタプログラムを示す。

3.3.1 VPSM インタプリタの構成

VPSM インタプリタは以下の 3 つの要素から成る。

1. **データセグメント (Data Segment)**：PRAM 用並列言語プログラム中で使用する変数の値を保存する。データセグメントは全てのプロセッサで共有される。
2. **命令セグメント (Instraction Sengemt)**：並列アセンブラプログラムを保存する。**Instraction Segment** は全てのプロセッサで共有される。
3. **仮想スタックマシン (Virtual Stack Machine**：1 個のスタックおよびプログラムカウンタから成る。仮想スタックマシンは各プロセッサに 1 台ずつ作成される。

各仮想スタックマシンは、PRAM の各プロセッサの動作をシミュレートする。各仮想スタックマシンは、プログラムカウンタが指す番地にある命令セグメントの命令を読み込み、読み込んだ命令に応じてデータセグメントへのデータの読み書き、各自のスタックへのデータの出し入れを行い、次の命令へ移る。

VPSM インタプリタは逐次状態と並列状態の 2 つの状態を持つ。逐次状態では、識別番号 0 を持つ 1 台の仮想スタックマシンのみが動作を行い、並列状態では PRAM 用並列言語プログラム中で指定された識別番号を持つ複数のスタックマシンが動作を行う。初期状態においては VPSM インタプリタは逐次状態にある。命令 **PARA** を読み込むと並列状態に以降し、複数の仮想スタックマシンが起動される。並列状態において命令 **SYNC** を読み込むと、全ての仮想スタックマシンが **SYNC** に到達するまで処理を中断し、その後逐次状態に移行する。

3.3.2 VPSM インタプリタの仕様

以下に本研究で作成したインタプリタの仕様を示す。(**VPSM インタプリタの使用方法**) PRAM 用並列アルゴリズム言語によって記述された PRAM

アルゴリズムを foo.pram とする。以下のコマンドを実行すると、foo.pram が並列アセンブラにコンパイルされ、*outputfile* に出力される。*outputfile* を省略した場合は OpCode.asm に出力される。

> java Pram [-*option*] foo.pram [outputfile]

foo.asm を並列アセンブラによって記述された PRAM アルゴリズムとする。以下のコマンドを実行すると、PRAM アルゴリズムの PRAM 上での実行がシミュレートされる。foo.asm を省略した場合は OpCode.asm を入力ファイルとして実行する。

> java VSM [-*option*] foo.asm

(**VPSM インタプリタのオプション**) VPSM インタプリタは実行時に以下のオプションを指定できる。

- **d**：VPSM をデバグモードで実行する。
- **t**：VPSM をトレスモードで実行する。
- **n**：並列アセンブラプログラムの文法チェックのみを行い実行しない。
- **c**：実行後に実行命令数、データセグメントサイズ、最大スタック深度を表示する。
- **o**：実行前に並列アセンブラプログラムを表示する。
- **p**：実行前に並列アセンブラプログラムを OpCode.asm に出力する。

c オプションを付けて VPSM を実行することにより、PRAM アルゴリズムを PRAM 上で実行した場合の実行時間を表示することができる。

4 結果および考察

本研究で作成した VPSM インタプリタが正しく PRAM アルゴリズムの動作をシミュレートしていることを確認するため、付録 A.3 に示す PRAM 用並列言語プログラムを用いて出力および実行時間を検証した。

付録 A.3 の PRAM 用並列言語プログラムは、n 個のデータの和を並列に求めるプログラムであり、PRAM 上では p 台のプロセッサを用いて $O(\frac{n}{p}+\log p)$ 時間で求めることができる。付録 A.3 の PRAM 用並列言語プログラムは、PRAM コンパイラにより付録 A.4 に示す並列アセンブラに変換される。本研究で作成した VPSM インタプリタを用いて付録 A.4 を実行すると、正しい解が出力され、また、PRAM 上での実行時間も出力された。図 3 に VPSM インタプリタが出力した PRAM アルゴリズムの実行時間と、理論値との比較を示す。図 3 より、実行時間と理論値はデータ数に比例していることからほぼ一致している。また、理論値と実行時間の間にデータ数に関わらず定数の差が出るのは変数の初期値設定時間や出力時間が加わるためである。

図 3: PRAM アルゴリズムの実行時間と、理論値との比較

5　結論・今後の課題

本研究では、JAVA 言語を用いて VPSM インタプリタの設計を行った。本研究で設計した VPSM インタプリタは、PRAM 上での並列アルゴリズムの実行をシミュレートすることができる。また、PRAM 上での並列アルゴリズムの実行時間を出力する機能を持ち、PRAM アルゴリズムの計算量を実験的に評価することができる。

今後の課題として、PRAM 用並列言語を拡張し、それに対応できるシミュレータを作成することなどが挙げられる。

6　謝辞

本研究を行うにあたって、アルゴリズムの基礎から並列処理についてまで、御指導や助言など大変尽力していただきまして、有難うございました。遅くまで研究室に残っていただいてくれたことなど、大変ご迷惑をおかけしましたが、おかげで充実した日々になりました。本当に、感謝の気持ちを申し上げます。

参考文献

[1] 加藤暢:”平成 17 年度第 5 セメスター 情報・システムプロジェクト I 指導書,” 近畿大学理工学部情報学科 (2005).

[2] 辻野嘉宏:”情報工学入門選書 10 コンパイラ,” 昭晃堂 (1996).

[3] 疋田輝雄, 石畑清:”コンパイラの理論と実現,” 共立出版 (1988).

A　付録

A.1　並列アセンブラの文法

以下に並列アセンブラの文法を記述する。スタックトップの値を $SP1$ とし、$SP1$ の一つ前の値が $SP2$ というように示す。

(**算術演算命令**)

- **ADD**：スタックより $SP1, SP2$ を取り出し、$SP2 + SP1$ をスタックに挿入する。
- **SUB**：スタックより $SP1, SP2$ を取り出し、$SP2 - SP1$ をスタックに挿入する。
- **MUL**：スタックより $SP1, SP2$ を取り出し、$SP2 * SP1$ をスタックに挿入する。
- **DIV**:スタックより $SP1, SP2$ を取り出し、$val1 \neq 0$ であれば $SP2/SP1$ をスタックに挿入する。$S1 = 0$ の時は、実行時エラーとなる。
- **MOD**:スタックより $SP1, SP2$ を取り出し、$val1 \neq 0$ であれば $SP2\%SP1$ をスタックに挿入する。
- **CSIGN**：$SP1$ を $-SP1$ に変更する。
- **INC**：$SP1$ を 1 増やす。
- **DEC**：$SP1$ を 1 減らす。

(**論理演算命令**)

- **AND**：スタックより $SP1, SP2$ を取り出し、$SP1 \neq 0$ かつ $SP2 \neq 0$ ならば 1 を、それ以外ならば 0 をスタックに挿入する。
- **OR**：スタックより $SP1, SP2$ を取り出し、$SP1 \neq 0$ または $SP2 \neq 0$ ならば 1 を、それ以外ならば 0 をスタックに挿入する。
- **NOT**：スタックより $SP1$ を取り出し、$SP1 = 0$ ならば 1 を、$SP1 \neq 0$ ならば 0 をスタックに挿入する。
- **XOR**：スタックより $SP1, SP2$ を取り出し、$SP1 \neq 0$ かつ $SP2 = 0$、または $SP1 = 0$ かつ $SP2 \neq 0$ ならば 1 を、それ以外ならば 0 をスタックに挿入する。
- **COMP**：スタックより $SP1, SP2$ を取り出し、$SP1 < SP2$ ならば $SP2$ を 1 に、$SP2 < SP1$ ならば $SP2$ を -1 に $SP1 = SP2$ ならば $SP2$ を 0 に置き換えスタックに挿入する。

(**スタックへの値の出し入れおよびデータセグメントへの読み書き命令**)

- **ASSIGN**：以下の操作を行う。

  1. スタックより値 $SP1$ および値 $addr$ を取り出す。
  2. データセグメントの $addr$ 番地に値 $SP1$ を書き込む。
  3. スタックに値 $SP1$ を挿入する。

  $addr$ は Dseg の番地とする。

- **PUSH**：PUSH $addr$ で、データセグメントの $addr$ の値をスタックに挿入する。

- **PUSHI**：PUSHI SP で、値 val をスタックに挿入する。

- **POP**：POP $addr$ で、スタックより値 val を取り出し、データセグメントの $addr$ 番地に値 val を書き込む。

- **LOAD**：スタックより値 $addr$ を取り出し、データセグメントの $addr$ 番地の値 val をスタックに挿入する。

- **REMOVE**：スタックよりデータを 1 つ取り除く。

- **COPY**：スタックトップのデータ val をスタックに挿入する。

- **PUSHP**：スタックに命令を実行しているプロセッサのプロセッサ番号を挿入する。

(**ジャンプ命令**)

- **JUMP** $addr$：プログラムカウンタの値を $addr$ に変更する。

- **BEQ** $addr$：スタックから値 val を取り出し、$val = 0$ ならばプログラムカウンタの値を $addr$ に変更する。

- **BNE** $addr$：スタックから値 val を取り出し、$val \neq 0$ ならばプログラムカウンタの値を $addr$ に変更する。

- **BLT** $addr$：スタックから値 val を取り出し、$val < 0$ ならばプログラムカウンタの値を $addr$ に変更する。

- **BLE** $addr$：スタックから値 val を取り出し、$val \leq 0$ ならばプログラムカウンタの値を $addr$ に変更する。

- **BGT** $addr$：スタックから値 val を取り出し、$val > 0$ ならばプログラムカウンタの値を $addr$ に変更する。

- **BGE** $addr$：スタックから値 val を取り出し、$val \geq 0$ ならばプログラムカウンタの値を $addr$ に変更する。
- **CALL** $addr$：スタックに現在のプログラムカウンタの値を挿入し、プログラムカウンタの値を $addr$ に変更する。
- **RET**：スタックから値 $addr$ を取り出し、プログラムカウンタの値を $addr$ に変更する。

(**算術演算およびデータセグメントへの読み書き命令**)

- **ADDLHS**：以下の操作を行う。
  1. スタックより値 $SP1$ および値 $addr$ を取り出す。
  2. データセグメントの $addr$ 番地の値 $SP2$ を読み込む。
  3. データセグメントの $addr$ 番地に値 $SP2 + SP1$ を書き込む。
  4. スタックに値 $SP2 + SP1$ を挿入する。
- **SUBLHS**：以下の操作を行う。
  1. スタックより値 $SP1$ および値 $addr$ を取り出す。
  2. データセグメントの $addr$ 番地の値 $SP2$ を読み込む。
  3. データセグメントの $addr$ 番地に値 $SP2 - SP1$ を書き込む。
  4. スタックに値 $SP2 - SP1$ を挿入する。
- **MULLHS**：以下の操作を行う。
  1. スタックより値 $SP1$ および値 $addr$ を取り出す。
  2. データセグメントの $addr$ 番地の値 $SP2$ を読み込む。
  3. データセグメントの $addr$ 番地に値 $SP2 * SP1$ を書き込む。
  4. スタックに値 $SP2 * SP1$ を挿入する。
- **DIVLHS**：以下の操作を行う。ただし、$SP1 = 0$ ならば実行時エラーとなる。
  1. スタックより値 $SP1$ および値 $addr$ を取り出す。
  2. データセグメントの $addr$ 番地の値 $SP2$ を読み込む。
  3. データセグメントの $addr$ 番地に値 $SP2/SP1$ を書き込む。
  4. スタックに値 $SP2/SP1$ を挿入する。
- **DIVLHS**：以下の操作を行う。ただし、$SP1 = 0$ ならば実行時エラーとなる。

1. スタックより値 $SP1$ および値 $addr$ を取り出す。
2. データセグメントの $addr$ 番地の値 $SP2$ を読み込む。
3. データセグメントの $addr$ 番地に値 $SP2\%SP1$ を書き込む。
4. スタックに値 $SP2\%SP1$ を挿入する。

- **POSTDEC** $addr$：以下の操作を行う。
  1. データセグメントの $addr$ 番地の値 val を読み込む。
  2. データセグメントの $addr$ 番地に値 $val - 1$ を書き込む。
  3. スタックに値 val を挿入する。
- **PREINC** $addr$：以下の操作を行う。
  1. データセグメントの $addr$ 番地の値 val を読み込む。
  2. データセグメントの $addr$ 番地に値 $val + 1$ を書き込む。
  3. スタックに値 $val + 1$ を挿入する。
- **PREDEC** $addr$：以下の操作を行う。
  1. データセグメントの $addr$ 番地の値 val を読み込む。
  2. データセグメントの $addr$ 番地に値 $val - 1$ を書き込む。
  3. スタックに値 $val - 1$ を挿入する。
- **POSTINC** $addr$：以下の操作を行う。
  1. データセグメントの $addr$ 番地の値 val を読み込む。
  2. データセグメントの $addr$ 番地に値 $val + 1$ を書き込む。
  3. スタックに値 val を挿入する。

(**入出力命令**)

- **INPUT**：標準入力より整数 i を読み込み、i をスタックに挿入する。
- **INPUTC**：標準入力より文字 c を読み込み、c の文字コードをスタックに挿入する。
- **OUTPUT**：スタックより整数値 i を取り出し、i を標準出力に出力する。
- **OUTPUT**：スタックより整数値 i を取り出し、文字コート i の文字 c を標準出力に出力する。
- **OUTPUTL**：標準出力に改行コードを出力する。

(**フラグレジスタ操作命令**)

- **SETFR** *addr*：フラグレジスタの値を *addr* に変更する。
- **INCFR** *addr*：フラグレジスタの値に *addr* を加える。
- **DECFR** *addr*：フラグレジスタの値から *addr* を引く。

(**並列命令**)

- **PARA**：逐次モードから並列モードに移行する。逐次モード実行中に **PARA** 命令を読み込んだ場合、スタックから 2 個のデータ d_1, d_2 が取り出される。この命令以降は、プロセッサ $P_{d_1}, P_{d_1+1}, P_{d_1+2}, ..., P_{d_2}$ が並列に命令を実行する。並列モード中に **PARA** 命令を読み込んだ場合、実行時エラーとなる。
- **SYNC**：並列モード実行中のプロセッサ間で同期を取り、逐次モードに移行する。並列モード中に **SYNC** 命令を読み込んだ場合、並列モード実行中の全てのプロセッサが **SYNC** 命令に到達するまで実行を中断する。全てのプロセッサが **SYNC** に到達すると、それ以降はプロセッサ P_0 のみが命令を実行する。逐次モード中に **SYNC** 命令が読み込まれると実行時エラーとなる。

(**その他の命令**)

- **NOP**：何も行わない。
- **HALT**：VPSM を停止する。
- **EOF**：実行時エラーとなる。
- **ERROR**：実行時エラーとなる。

A.2 VPSM インタプリタプログラム

VPSM インタプリタプログラムは以下に挙げるプログラムから成る。

- **DataSegment.java**： データセグメント
- **InputFile.java**： ファイルへの入出力 (Pram コンパイラと共通)
- **Instraction.java**： 命令コード (Pram コンパイラと共通)
- **InstractionSegment.java**： 命令コードセグメント (Pram コンパイラと共通)
- **Operators.java**： 命令表 (Pram コンパイラと共通)

- **PramMode.java**： Dseg の同時読み同時書きモード
- **Stack.java**： スタック
- **VirtualStackMachine**： 仮想スタックマシーン
- **VSM.java**： メイン
- **VSMLexer.java**： 字句解析機

以下にそれぞれのソースファイルを表記する。

A.2.1 DataSegment.java

```
public class DataSegment {
static final int DSEGSIZE = 1024;
int[] dseg;
int[] tempDseg;        // 作業用配列
boolean[] readCheck;   // 同時読みチェック
boolean[] writeCheck; // 同時書きチェック
int maxAddr;           // 使用アドレスの最大値
int mode;              // PRAM mode
int size;              // サイズ

public DataSegment(int m) {
size = DSEGSIZE;
dseg = new int[DSEGSIZE];
tempDseg = new int[DSEGSIZE];
readCheck = new boolean[DSEGSIZE];
writeCheck = new boolean[DSEGSIZE];
maxAddr = -1;
mode = m;

for (int i=0; i<DSEGSIZE; i++) { // データの初期化
dseg[i] = 0;
tempDseg[i] = 0;
readCheck[i] = false;
writeCheck[i] = false;
}
}

    // アドレス addr のデータを読む
```

```
public int read(int addr) {
if (addr < 0 || addr >= size)
executeError("Illegal dseg address : " + addr);
if (addr>maxAddr) maxAddr = addr;
readCheck[addr] = true;
return dseg[addr];
}

    // アドレス addr にデータ val を書く
public void write(int addr, int val) {
if (addr < 0 || addr >= size)
executeError("Illegal dseg address : " + addr);
if (addr>maxAddr) maxAddr = addr;
writeCheck[addr] = true;
tempDseg[addr] = val;          /* 更新データは tmp 配列に */
return;
}

// データ更新し、チェックフラグを初期化する
public void set() {
for(int i=0; i<=maxAddr; i++) {
dseg[i] = tempDseg[i];     /* tmp 配列のデータをコピー */
readCheck[i] = false;
writeCheck[i] = false;
}
return;
}

public void executeError (String err_mes) {   /* 実行時エラー */
System.out.println("Processor "+VSM.vsm.pr);
System.out.println("Execute error at line " + VSM.vsm.pctr);
System.out.println(err_mes);
VSM.iseg.print(VSM.vsm.pctr);
System.out.println();
System.exit(1);
}
}
```

A.2.2 InputFile.java

```
import ioTools.*;
import java.io.*;

public class InputFile {
BufferedReader buffer;   /* 入力ファイルのバッファ */
String line;             /* 入力ファイルの 1 行分の文字列 */
int linenum;             /* 入力ファイルの行番号 */
int columnnum;           /* 入力ファイルの列番号 */
char currentc;           /* 読み込んだ文字 */
char nextc;              /* 次に読み込む文字 */

    /* コンストラクタでは，inputFileName というファイルを開き
       そのファイルを今後 buffer で参照する．また linenum，
       columnnum，currentc，nextc を初期化する */
public InputFile(String inputFileName) {
buffer = FileIo.fRead(inputFileName);
linenum = 0;
columnnum = 0;
//入力ファイルから一行読む
readInputFile();
//最初の一文字目を読んで，その文字を nextc に格納する．
nextc = ' ';
nextChar();
}

//buffer から一行読み，文字列変数 line にその行を格納するメソッド．
public void readInputFile() {
try {
line = buffer.readLine();
} catch(IOException error_report) {
            /* 読込みエラーが発生したら，キャッチした例外を表示し，
               ファイルを閉じ，処理系を終了させる */
System.out.println(error_report);
closeFile();
System.exit(1);
}
}
```

```
//入力ファイルを閉じるメソッド.
public void closeFile() {
try {
buffer.close();
} catch(IOException error_report) {
System.out.println(error_report);
System.exit(1);
}
}

//次の文字を得るメソッド.
public char nextChar() {
if (line == null) { //ファイル末に達したら'\0' を返す.
currentc = nextc;
nextc = '\0';
return currentc;
} else if (columnnum >= line.length()) { //行末に達したら'\n' を
返す
readInputFile();
currentc = nextc;
nextc = '\n';
linenum++;
columnnum = 0;
return currentc;
} else { //通常の動作. 読んだ一文字を nextc に格納し, その値を返す.

currentc = nextc;
nextc = line.charAt(columnnum);
columnnum++;
return currentc;
}
}
}
```

A.2.3 Instraction.java

```
class Instraction implements Operators {
int operator;        /* オペレータ */
int operand;         /* オペランド */
```

```
boolean register;    /* アドレス修飾 */

public Instraction(int i, int j) {
operator = i;
operand = j;
register = false;
}

public Instraction(int i) {
operator = i;
operand = -1;
register = false;
}

public Instraction(int i, int j, boolean r) {
operator = i;
operand = j;
register = r;
}

public Instraction(int i, boolean r) {
operator = i;
operand = -1;
register = r;
}

public String toString() {
switch(operator) {
case PUSH:
case PUSHI:
case POP:
case BEQ:
case BNE:
case BLE:
case BLT:
case BGE:
case BGT:
case JUMP:
case CALL:
```

```
return opName() + "\t" + operand + "\t";
                          /* オペランドを持つ場合 */
default:
return opName() + "\t\t";
                          /* オペランドを持たない場合 */
}
}

    // オペランドコードをオペランド名に変換
public String opName() {
switch(operator) {
case NOP:      return "NOP    ";     // no operation
case ASSGN:    return "ASSGN  ";     // assign
case ADD:      return "ADD    ";     // +
case ADDLHS:   return "ADDLHS ";     // +=
case SUB:      return "SUB    ";     // -
case SUBLHS:   return "SUBLHS ";     // -=
case MUL:      return "MUL    ";     // *
case MULLHS:   return "MULLHS ";     // *=
case DIV:      return "DIV    ";     // /
case DIVLHS:   return "DIVLHS ";     // /=
case MOD:      return "MOD    ";     // %
case MODLHS:   return "MODLHS ";     // %=
case CSIGN:    return "CSIGN  ";     // 単項-
case AND:      return "AND    ";     // and
case OR:       return "OR     ";     // or
case NOT:      return "NOT    ";     // not
case XOR:      return "XOR    ";     // exclusive or
case COMP:     return "COMP   ";     // comp
case COPY:     return "COPY   ";     // copy
case PUSH:     return "PUSH   ";     // push
case PUSHI:    return "PUSHI  ";     // push integer
case POP:      return "POP    ";     // pop
case REMOVE:   return "REMOVE ";     // remove
case LOAD:     return "LOAD   ";     // load
case INC:      return "INC    ";     // ++
case DEC:      return "DEC    ";     // --
case PREINC:   return "PREINC ";     // 前置++
case PREDEC:   return "PREDEC ";     // 前置--
```

```
case POSTINC:  return "POSTINC";     // 後置++
case POSTDEC:  return "POSTDEC";     // 後置--
case SETFR:    return "SETFR  ";     // set frame register
case INCFR:    return "INCFR  ";     // inc frame register
case DECFR:    return "DECFR  ";     // dec frame register
case JUMP:     return "JUMP   ";     // jump
case BEQ:      return "BEQ    ";     // == ?
case BNE:      return "BNE    ";     // != ?
case BLT:      return "BLT    ";     // < ?
case BLE:      return "BLE    ";     // <= ?
case BGT:      return "BGT    ";     // > ?
case BGE:      return "BGE    ";     // >= ?
case CALL:     return "CALL   ";     // call
case RET:      return "RET    ";     // return
case INPUT:    return "INPUT  ";     // input integer
case INPUTC:   return "INPUTC ";     // input character
case INPUTS:   return "INPUTS ";     // input string
case OUTPUT:   return "OUTPUT ";     // output integer
case OUTPUTC:  return "OUTPUTC";     // output character
case OUTPUTL:  return "OUTPUTL";     // output line
case OUTPUTS:  return "OUTPUTS";     // output string
case RAND:     return "RAND   ";     // random
case HALT:     return "HALT   ";     // halt
case ASSGNS:   return "ASSGNS ";     // assign string
case LOADS:    return "LOADS  ";     // load stirng
case PARA:     return "PARA   ";     // parallel
case SYNC:     return "SYNC   ";     // syncronous
case PUSHP:    return "PUSHP  ";     // push processor number
case EOF:      return "EOF    ";     // end of file
default:       return "ERROR  ";     // error
}
}
}
```

A.2.4 InstractionSegment.java

```
import ioTools.*;  //ファイル入出力用
import java.io.*;  //ファイル入出力用
import java.util.Vector; //ベクトル用
```

```
public class InstractionSegment implements Operators {
Vector iseg;
int isegPtr;
int size;

boolean debugSW;

public InstractionSegment(boolean dsw) {
iseg = new Vector();
isegPtr = 0;
size = 0;
debugSW = dsw;
}

    // Iseg に命令を加える
public int appendCode(Instraction inst) {
if (size == isegPtr) {
if (debugSW) System.out.println(isegPtr+": "+inst);
iseg.addElement(inst);
size++;
} else {
Instraction oldInst = ((Instraction) iseg.get(isegPtr));
if (debugSW) System.out.print(isegPtr+": "+oldInst);
iseg.removeElementAt(isegPtr);
iseg.insertElementAt(inst, isegPtr);
if (debugSW) System.out.println("-> "+inst);
}
isegPtr++;
return isegPtr-1;
}

public int appendCode(int operator, int operand, boolean register) {
if (size == isegPtr) {
Instraction inst = new Instraction(operator, operand, register);
if (debugSW) System.out.println(isegPtr+": "+inst);
iseg.addElement(inst);
size++;
} else {
Instraction inst = ((Instraction) iseg.get(isegPtr));
```

```
if (debugSW) System.out.print(isegPtr+": "+inst);
inst.operator = operator;
inst.operand = operand;
inst.register = register;
if (debugSW) System.out.println("-> "+inst);
}
isegPtr++;
return isegPtr-1;
}

public int appendCode(int operator, boolean register) {
if (size == isegPtr) {
Instraction inst = new Instraction(operator, register);
if (debugSW) System.out.println(isegPtr+": "+inst);
iseg.addElement(inst);
size++;
} else {
Instraction inst = ((Instraction) iseg.get(isegPtr));
if (debugSW) System.out.print(isegPtr+": "+inst);
inst.operator = operator;
inst.operand = -1;
inst.register = register;
if (debugSW) System.out.println("-> "+inst);
}
isegPtr++;
return isegPtr-1;
}

public int appendCode(int operator, int operand) {
if (size == isegPtr) {
Instraction inst = new Instraction(operator, operand);
if (debugSW) System.out.println(isegPtr+": "+inst);
iseg.addElement(inst);
size++;
} else {
Instraction inst = ((Instraction) iseg.get(isegPtr));
if (debugSW) System.out.print(isegPtr+": "+inst);
inst.operator = operator;
inst.operand = operand;
```

```
inst.register = false;
if (debugSW) System.out.println("-> "+inst);
}
isegPtr++;
return isegPtr-1;
}

public int appendCode(int operator) {
if (size == isegPtr) {
Instraction inst = new Instraction(operator);
if (debugSW) System.out.println(isegPtr+": "+inst);
iseg.addElement(inst);
size++;
} else {
Instraction inst = ((Instraction) iseg.get(isegPtr));
if (debugSW) System.out.print(isegPtr+": "+inst);
inst.operator = operator;
inst.operand = -1;
inst.register = false;
if (debugSW) System.out.println("-> "+inst);
}
isegPtr++;
return isegPtr-1;
}

public int operator(int addr) {      /* オペレータを返す */
if (addr < 0 || addr >= size)
executeError("Illegal iseg address ", addr);
return ((Instraction) iseg.get(addr)).operator;
}

public int operand(int addr) {       /* オペランドを返す */
if (addr < 0 || addr >= size)
executeError("Illegal iseg address ", addr);
return ((Instraction) iseg.get(addr)).operand;
}

public boolean register (int addr) { /* レジスタを返す */
if (addr < 0 || addr >= size)
```

```
executeError("Illegal iseg address ", addr);
return ((Instraction) iseg.get(addr)).register;
}

    // 命令のジャンプ先のアドレスをチェック
public void checkAddress(int addr) {
Instraction inst = (Instraction) iseg.get(addr);
switch(inst.operator) {
case JUMP:
case BEQ:
case BNE:
case BLT:
case BLE:
case BGT:
case BGE:
case CALL:
if(inst.operand < 0 || inst.operand >= size)
syntaxError("Illegal iseg address : " + inst, addr);
break;
default:
break;
}
}

    // 指定したアドレスの命令を表示
public void print(int addr) {;
System.out.print(addr + ": " + (Instraction) iseg.get(addr));
}

    // Iseg を表示
public void dump() {
for (int i=0; i<isegPtr; i++)
System.out.println(i + ": " + (Instraction) iseg.get(i));
}

    // Iseg をデフォルトファイルに出力
public void dumpToFile() {
PrintWriter outputFile = FileIo.fWrite("OpCode.asm", false);
for (int i=0; i<isegPtr; i++)
```

```
outputFile.println((Instraction) iseg.get(i));
outputFile.close();
}

    // Iseg を指定したファイルに出力
public void dumpToFile(String fileName) {
PrintWriter outputFile = FileIo.fWrite(fileName, false);
for (int i=0; i<isegPtr; i++)
outputFile.println((Instraction) iseg.get(i));
outputFile.close();
}

    // addr 番目の命令の オペレータ, オペランド を operator, operand に
変更する
public void replaceCode(int addr, int operator, int operand) {
Instraction inst = ((Instraction) iseg.get(addr));
if (debugSW)
System.out.print(addr + ": " + inst);
inst.operator = operator;
inst.operand = operand;
if (debugSW)
System.out.println("-> " + inst);
}

// addr 番目の命令のオペランド を operand に変更する
public void replaceCode(int addr, int operand) {
Instraction inst = ((Instraction) iseg.get(addr));
if (debugSW)
System.out.print(addr + ": " + inst);
inst.operand = operand;
if (debugSW)
System.out.println("-> " + inst);
}

// addr 番目の命令を inst に変更する
public void replaceCode(int addr, Instraction inst) {
Instraction oldInst = ((Instraction) iseg.get(addr));
if (debugSW) System.out.print(addr+": "+oldInst);
iseg.removeElementAt(addr);
```

```
iseg.insertElementAt(inst, addr);
if (debugSW) System.out.println("-> "+inst);
}

public void syntaxError(String err_mes, int addr) {     /* 文法
エラー */
System.out.println("Syntax error at line " + addr);
System.out.println(err_mes);
System.out.println((Instraction) iseg.get(addr));
System.exit(1);
}

public void executeError (String err_mes, int addr) {   /* 実行
時エラー */
System.out.println("Execute error at line " + addr);
System.out.println(err_mes);
System.exit(1);
}
}
```

A.2.5 Operators.java

```
interface Operators {
static final int NOP     =  0;  // no operation
static final int ASSGN   =  1;  // assign
static final int ADD     =  2;  // +
static final int ADDLHS  =  3;  // +=
static final int SUB     =  4;  // -
static final int SUBLHS  =  5;  // -=
static final int MUL     =  6;  // *
static final int MULLHS  =  7;  // *=
static final int DIV     =  8;  // /
static final int DIVLHS  =  9;  // /=
static final int MOD     = 10;  // %
static final int MODLHS  = 11;  // %=
static final int CSIGN   = 12;  // 単項-
static final int AND     = 13;  // and
static final int OR      = 14;  // or
static final int NOT     = 15;  // not
static final int XOR     = 16;  // exclusive or
```

```
static final int COMP    = 17;  // comp
static final int COPY    = 18;  // copy
static final int PUSH    = 19;  // push
static final int PUSHI   = 20;  // push integer
static final int REMOVE  = 21;  // remove
static final int LOAD    = 22;  // load
static final int POP     = 23;  // pop
static final int INC     = 24;  // ++
static final int DEC     = 25;  // --
static final int PREINC  = 26;  // 前置++
static final int PREDEC  = 27;  // 前置--
static final int POSTINC = 28;  // 後置++
static final int POSTDEC = 29;  // 後置--
static final int SETFR   = 30;  // set frame register
static final int INCFR   = 31;  // inc frame register
static final int DECFR   = 32;  // dec frame register
static final int JUMP    = 33;  // jump
static final int BLT     = 34;  // < ?
static final int BLE     = 35;  // <= ?
static final int BEQ     = 36;  // == ?
static final int BNE     = 37;  // != ?
static final int BGE     = 38;  // > ?
static final int BGT     = 39;  // >= ?
static final int CALL    = 40;  // call
static final int RET     = 41;  // return
static final int INPUT   = 42;  // input integer
static final int INPUTC  = 43;  // input character
static final int OUTPUT  = 44;  // output integer
static final int OUTPUTC = 45;  // output character
static final int OUTPUTL = 46;  // output line
static final int RAND    = 47;  // random
static final int HALT    = 48;  // halt
static final int PUSHS   = 49;  // push string
static final int POPS    = 50;  // pop string
static final int ASSGNS  = 51;  // assign string
static final int LOADS   = 52;  // load string
static final int INPUTS  = 53;  // input string
static final int OUTPUTS = 54;  // output string
static final int PARA    = 55;  // parallel
```

```
static final int SYNC    = 56;  // synchronous
static final int PUSHP   = 57;  // push processor number
static final int EOF     =255;  // end of file
static final int ERROR   = -1;  // error
}
```

A.2.6 PramMode.java

```
interface PramMode {
static final int M_EREW = 0;
static final int M_CREW = 1;
static final int M_COMMON_CWCW = 2;
static final int M_ARBITRARY_CRCW = 3;
static final int M_PRIORITY_CRCW = 4;
```

A.2.7 Stack.java

```
public class Stack {
static final int STACKSIZE = 128;
int[] st;         // Stack
int sp;           // StackPointer
int size;         // StackSize
int maxSp;        // Max Stack Pointer

public Stack () {
size = STACKSIZE;
st = new int[STACKSIZE];
sp = -1;
maxSp = -1;
}

public int pop () {
if (sp < 0) executeError("Stack Underflow");
int val = st[sp];
sp--;
return val;
}

public void push (int val) {
```

```
sp++;
if (sp >= size) executeError("Stack Overflow");
st[sp] = val;
if (sp > maxSp) maxSp = sp;
return;
}

public String toString() {
if (sp >= 0) return sp + "\t" + st[sp] + "\t";
else return "\t\t";
}

public void executeError (String err_mes) {   /* 実行時エラー */
System.out.println("Processor "+VSM.vsm.pr);
System.out.println("Execute error at line " + VSM.vsm.pctr);
System.out.println(err_mes);
VSM.iseg.print(VSM.vsm.pctr);
System.out.println();
System.exit(1);
}
}
```

A.2.8 VertualStackMachine

```
import ioTools.*;

public class VirtualStackMachine implements Operators, PramMode {
InstractionSegment iseg;// Instranciton Segment
DataSegment dseg;       // Data Segment
Stack stack;            // Stack
int pctr;               // Program Counter
boolean isRunning;      // status
boolean fregP;
int freg;               // Frame Register
int minFreg;
int callCtr;            // Call Counter
int insCtr;             // Instrument Counter
int pr;                 // Processor Number

public VirtualStackMachine(InstractionSegment is, DataSegment ds, int p, boolean fp) {
```

```
iseg = is;
dseg = ds;
stack = new Stack();
pctr = 0;
isRunning = false;
callCtr = 0;
insCtr = 0;
pr = p;
freg = 0;
fregP = fp;
minFreg = ds.size;
}

public int execute(boolean traceSW) {
int operator = iseg.operator(pctr);
int operand = iseg.operand(pctr);
if (iseg.register(pctr) && fregP)
operand += freg;

if (isRunning) {
int val, val1, val2, addr;
int i;
String str;
insCtr++;

if (traceSW) iseg.print(pctr);

switch(operator) {
case NOP:
pctr++;
break;
case ASSGN:
val = stack.pop();
addr = stack.pop();
dseg.write(addr, val);
stack.push(val);
pctr++;
break;
case ADD:
```

```
val1 = stack.pop();
val2 = stack.pop();
stack.push(val2 + val1);
pctr++;
break;
case ADDLHS:
val1 = stack.pop();
addr = stack.pop();
val2 = dseg.read(addr);
val = val2 + val1;
dseg.write(addr, val);
stack.push(val);
pctr++;
break;
case SUB:
val1 = stack.pop();
val2 = stack.pop();
stack.push(val2 - val1);
pctr++;
break;
case SUBLHS:
val1 = stack.pop();
addr = stack.pop();
val2 = dseg.read(addr);
val = val2 - val1;
dseg.write(addr, val);
stack.push(val);
pctr++;
break;
case MUL:
val1 = stack.pop();
val2 = stack.pop();
stack.push(val2 * val1);
pctr++;
break;
case MULLHS:
val1 = stack.pop();
addr = stack.pop();
val2 = dseg.read(addr);
```

```
val = val2 * val1;
dseg.write(addr, val);
stack.push(val);
pctr++;
break;
case DIV:
val1 = stack.pop();
if (val1 == 0)
executeError("Zero divider detected");
val2 = stack.pop();
val = val2 / val1;
stack.push(val);
pctr++;
break;
case DIVLHS:
val1 = stack.pop();
if (val1 == 0)
executeError("Zero divider detected");
addr = stack.pop();
val2 = dseg.read(addr);
val = val2 / val1;
dseg.write(addr, val);
stack.push(val);
pctr++;
break;
case MOD:
val1 = stack.pop();
if (val1 == 0)
executeError("Zero divider detected");
val2 = stack.pop();
val = val2 % val1;
stack.push(val);
pctr++;
break;
case MODLHS:
val1 = stack.pop();
if (val1 == 0)
executeError("Zero divider detected");
addr = stack.pop();
```

```
val2 = dseg.read(addr);
val = val2 % val1;
dseg.write(addr, val);
stack.push(val);
pctr++;
break;
case CSIGN:
val = stack.pop();
stack.push(-val);
pctr++;
break;
case AND:
val1 = stack.pop();
val2 = stack.pop();
if (val2 != 0 && val1 != 0) stack.push(1);
else stack.push(0);
pctr++;
break;
case OR:
val1 = stack.pop();
val2 = stack.pop();
if (val2 != 0 || val1 != 0) stack.push(1);
else stack.push(0);
pctr++;
break;
case NOT:
val = stack.pop();
if (val == 0) stack.push(1);
else stack.push(0);
pctr++;
break;
case XOR:
val1 = stack.pop();
val2 = stack.pop();
if ((val2 != 0 && val1 == 0) || (val2 == 0 && val1 != 0)) stack.push(1);
else stack.push(0);
pctr++;
break;
case COMP:
```

```
val1 = stack.pop();
val2 = stack.pop();
if (val2 == val1) stack.push(0);
else if (val2 > val1) stack.push(1);
else stack.push(-1);
pctr++;
break;
case COPY:
val = stack.pop();
stack.push(val);
stack.push(val);
pctr++;
break;
case PUSH:
stack.push(dseg.read(operand));
pctr++;
break;
case PUSHI:
stack.push(operand);
pctr++;
break;
case PUSHP:
stack.push(pr);
pctr++;
break;
case POP:
dseg.write(operand, stack.pop());
pctr++;
break;
case REMOVE:
stack.pop();
pctr++;
break;
case LOAD:
addr = stack.pop();
stack.push(dseg.read(addr));
pctr++;
break;
case INC:
```

```
val = stack.pop();
val++;
stack.push(val);
pctr++;
break;
case DEC:
val = stack.pop();
val--;
stack.push(val);
pctr++;
break;
case PREINC:
addr = stack.pop();
val = dseg.read(addr);
val++;
dseg.write(addr, val);
stack.push(val);
case PREDEC:
addr = stack.pop();
val = dseg.read(addr);
val--;
dseg.write(addr, val);
stack.push(val);
case POSTINC:
addr = stack.pop();
val = dseg.read(addr);
stack.push(val);
val++;
dseg.write(addr, val);
case POSTDEC:
addr = stack.pop();
val = dseg.read(addr);
stack.push(val);
val--;
dseg.write(addr, val);
case SETFR:
freg = operand;
if (freg > dseg.size)
executeError("Freg overflow");
```

```
else if (freg < minFreg) minFreg = freg;
pctr++;
break;
case INCFR:
freg += operand;
if (freg > dseg.size)
executeError("Freg overflow");
pctr++;
break;
case DECFR:
freg -= operand;
if (freg < minFreg) minFreg = freg;
pctr++;
break;
case JUMP:
pctr = operand;
break;
case BEQ:
val = stack.pop();
if (val == 0) pctr = operand;
else pctr++;
break;
case BNE:
val = stack.pop();
if (val != 0) pctr = operand;
else pctr++;
break;
case BLT:
val = stack.pop();
if (val < 0) pctr = operand;
else pctr++;
break;
case BLE:
val = stack.pop();
if (val <= 0) pctr = operand;
else pctr++;
break;
case BGT:
val = stack.pop();
```

```
if (val > 0) pctr = operand;
else pctr++;
break;
case BGE:
val = stack.pop();
if (val >= 0) pctr = operand;
else pctr++;
break;
case CALL:
stack.push(pctr);
pctr = operand;
callCtr++;
break;
case RET:
pctr = stack.pop();
break;
case INPUT:
System.out.print("Integer : ");
val = Console.ReadInteger();
stack.push(val);
pctr++;
break;
case INPUTC:
System.out.print("Character : ");
val = (int) Console.ReadCharacter();
stack.push(val);
pctr++;
break;
case OUTPUT:
val = stack.pop();
System.out.print(val + " ");
pctr++;
break;
case OUTPUTC:
val = stack.pop();
if (traceSW)
switch(val) {
case '\0':
System.out.print("\\0");
```

```
break;
case '\b':
System.out.print("\\b");
break;
case '\n':
System.out.print("\\n");
break;
case '\r':
System.out.print("\\r");
break;
case '\t':
System.out.print("\\t");
break;
default:
System.out.print((char) val);
break;
}
else System.out.print((char) val);
pctr++;
break;
case OUTPUTL:
if (traceSW) System.out.print("\\n");
else System.out.println();
pctr++;
break;
case RAND:
val = (int) (stack.pop() * Math.random());
stack.push(val);
pctr++;
break;
case PUSHS:
str = "";
for(i=0; true; i++) {
str += (char) dseg.read(operand+i);
if (str.charAt(i) == '\0') break;
}
for(i=str.length()-1; i>=0; i--)
stack.push((int) str.charAt(i));
pctr++;
```

```
break;
case POPS:
str="";
for(i=0; true; i++) {
str += (char) stack.pop();
dseg.write(operand+i, (int) str.charAt(i));
if (str.charAt(i) == '\0') break;
}
pctr++;
break;
case ASSGNS:
str = "";
for(i=0; true; i++) {
str += (char) stack.pop();
if (str.charAt(i) == '\0') break;
}
addr = stack.pop();
for(i=0; i<str.length(); i++)
dseg.write(addr+i, (int) str.charAt(i));
stack.push(str.length());
pctr++;
break;
case LOADS:
str = "";
addr = stack.pop();
for(i=0; true; i++) {
str += (char) dseg.read(addr+i);
if (str.charAt(i) == '\0') break;
}
for(i=str.length()-1; i>=0; i--)
stack.push((int) str.charAt(i));
pctr++;
break;
case INPUTS:
System.out.print("String : ");
str = Console.ReadString();
stack.push('\0');
for(i=str.length()-1; i>=0; i--)
stack.push((int) str.charAt(i));
```

```
pctr++;
break;
case OUTPUTS:
str = "";
for (i=0; true; i++) {
str += (char) stack.pop();
if (str.charAt(i) == '\0') break;
if (traceSW)
switch(str.charAt(i)) {
case '\0':
System.out.print("\\0");
break;
case '\b':
System.out.print("\\b");
break;
case '\n':
System.out.print("\\n");
break;
case '\r':
System.out.print("\\r");
break;
case '\t':
System.out.print("\\t");
break;
default:
System.out.print((char) str.charAt(i));
break;
}
else System.out.print((char) str.charAt(i));
}
pctr++;
break;
case PARA:
if (VSM.inParallel)
executeError("Already in parallel mode");
int h = stack.pop();
int l = stack.pop();
VSM.setParallel(l, h);
VSM.inParallel = true;
```

```
break;
case SYNC:
if (!VSM.inParallel)
executeError("Already in sequential mode");
isRunning = false;
break;
case HALT:
isRunning = false;
break;
case EOF:
executeError("Illegal end of file");
break;
default:
syntaxError("Illegal operator");
}
if (traceSW)
System.out.println(stack);
}
return operator;
}
public void setProgramCounter(int addr) {
pctr = addr;
}

public void incProgramCounter() {
pctr++;
}

public void awake() {
isRunning = true;
}

public void syntaxError () {                /* 文法エラー */
System.out.println("Processor "+pr);
System.out.println("Syntax error at line " + pctr);
iseg.print(pctr);
System.out.println();
System.exit(1);
}
```

```
public void syntaxError (String err_mes) {   /* 文法エラー */
System.out.println("Processor "+pr);
System.out.println("Systax error at line " + pctr);
System.out.println(err_mes);
iseg.print(pctr);
System.out.println();
System.exit(1);
}

public void executeError () {                /* 実行時エラー */
System.out.println("Processor "+pr);
System.out.println("Execute error at line " + pctr);
iseg.print(pctr);
System.out.println();
System.exit(1);
}

public void executeError (String err_mes) {   /* 実行時エラー */
System.out.println("Processor "+pr);
System.out.println("Execute error at line " + pctr);
System.out.println(err_mes);
iseg.print(pctr);
System.out.println();
System.exit(1);
}
}
```

A.2.9 VSM.java

```
import ioTools.*;

public class VSM implements Operators, PramMode {
static final int PROCESSORMAX = 256;
static final int MODE = M_EREW;         // 同時読み書きモード
static final boolean FREGP = false;     //フレームレジスタ修飾

static VSMLexer lexer;                  // 字句解析機
/* Lexer は並列対応時でも 1 個のまま変更しない */
```

```
static VirtualStackMachine vsm[];        // Virtual Stack Machine
/* 1 台のプロセッサは 1 台の VSM である */
/* 並列対応時は VirtualStackMachine 型の配列にする */
// static VirtualStackMachine vsm[];

static InstractionSegment iseg;         // Instraction Segment
static DataSegment dseg;                // Data Segment
 /* Iseg, Dseg は並列対応時でも各 1 個ずつのまま変更しない */

static boolean inParallel;      // true: 並列処理中 false: 逐次
処理中
static int low_processor;
static int high_processor;
/* 並列処理時は vsm[low_processor]〜vsm[high_processor] が動く
       逐次処理時は vsm[0] が動く */

static boolean execSW = true;           /* コンパイルだけ */
static boolean objOutSW = false;        /* 目的コードの表示 */
static boolean objPrtSW = false;        /* 目的コードの表示 */
static boolean traceSW = true;         /* トレースモード */
static boolean statSW = true;          /* 実行データの表示 */
static boolean debugSW = false;         /* デバッグモード */
static boolean symPrtSW = false;        /* 記号表の表示 */

public static void main (String[] args) {
String sourceFile = setUpOption(args);

dseg = new DataSegment(MODE);           /* data segment */

lexer = new VSMLexer(sourceFile);       /* 字句解析器 */
iseg = lexer.makeIseg(debugSW);         /* instraction segment */
lexer.inFile.closeFile();      /* 解析が終了したらファイルを閉じ
る */

if (objOutSW) iseg.dump();
if (objPrtSW) iseg.dumpToFile();

if (execSW) {
            vsm = new VirtualStackMachine[256];
```

```
for (int i=0; i<256; i++)
vsm[i]= new VirtualStackMachine(iseg, dseg, i, FREGP);

startVSM(0);                              /* VSM 開始 */

if (statSW) execReport();
else System.out.println("Execution finishied");
}
}

public static void startVSM(int startAddr) {
int[] operator;
operator=new int[PROCESSORMAX];

vsm[0].setProgramCounter(startAddr);
vsm[0].awake();
inParallel = false;
low_processor = 0;
high_processor = 0;
boolean syncck=true;

while (true) {
if(inParallel){
syncck=true;
for(int i=low_processor;i<=high_processor; i++){
operator[i]=vsm[i].execute(traceSW);            // execute() を
実行する
if (operator[i] == HALT)                       // 命令が PARA また
は HALT ならばエラー
syntaxError(vsm[i].pctr);
else if(operator[i] == PARA)
syntaxError(vsm[i].pctr);
}

for (int i=low_processor;i<=high_processor; i++){
if(operator[i]!=SYNC)
syncck=false;
}
if(syncck==true){                         // SYNC 到達チェック用変数
```

```
が true ならば逐次処理へ移行
vsm[0].pctr=vsm[low_processor].pctr;                // vsm[0] のプ
ログラムカウンタを vsm[low_prpcessor] のプログラムカウンタに合わせる
vsm[0].incProgramCounter();
inParallel = false;
low_processor=0;
high_processor=0;
vsm[0].awake();                    // vsm[0] を起こす
}
}
else {
operator[0]=vsm[0].execute(traceSW);   // execute() を実行する
if (operator[0] == HALT) break;     // 命令が HALT なら while ルー
プ脱出 プログラム終了へ
if (operator[0] == SYNC)            // 命令が SYNC ならエラー
syntaxError(vsm[0].pctr);
else if (operator[0] == PARA) {         // 命令が PARA なら並列処
理に移行
inParallel=true;
int proc=vsm[0].pctr;
for(int i=low_processor; i<=high_processor; i++){
vsm[i].pctr=proc;
vsm[i].incProgramCounter();
}
for(int i=low_processor; i<=high_processor; i++)
vsm[i].awake();
}
}
dseg.set();
}
}

public static String setUpOption (String[] args) {
int i;
for (i=0; i<args.length; i++) {
if (args[i].charAt(0) == '-') {
if (args[i].indexOf('c') != -1) statSW = true;     /* 実行データ
の表示 */
if (args[i].indexOf('d') != -1) debugSW = true;    /* デバッグモー
```

```
ド */
if (args[i].indexOf('n') != -1) execSW = false;    /* コンパイル
だけ */
if (args[i].indexOf('o') != -1) objOutSW = true;   /* 目的コード
の表示 */
if (args[i].indexOf('p') != -1) objPrtSW = true;   /* 目的コード
の表示 */
if (args[i].indexOf('s') != -1) symPrtSW = true;   /* 記号表の表
示 */
if (args[i].indexOf('t') != -1) traceSW = true;    /* トレースモー
ド */
} else break;
}
if (i<args.length) return args[i];
else return "OpCode.asm";
}

public static void setParallel(int l, int h) {
low_processor = l;
high_processor = h;
inParallel = true;
}

public static void execReport() {
System.out.println();
System.out.println("Object code size: "+iseg.size);
System.out.print("Max stack depth: ");
int maxstdp =0;
for(int i=0;i<PROCESSORMAX;i++){
if (vsm[i].stack.maxSp > maxstdp)
maxstdp=vsm[i].stack.maxSp;
}
System.out.println(maxstdp);
System.out.print("Max frame size: ");
System.out.println(dseg.size - vsm[0].minFreg);
System.out.print("Function calls: ");
int Fccl=0;
for(int i=0;i<PROCESSORMAX;i++){
Fccl=Fccl+vsm[i].callCtr;
```

```
}
System.out.println(Fccl);
System.out.print("Executiin count : ");
System.out.println(vsm[0].insCtr);
}

public static void syntaxError (int addr) {                  /* 文
法エラー */
System.out.println("Syntax error at line " + addr);
iseg.print(addr);
System.out.println();
System.exit(1);
}

public static void syntaxError (String err_mes, int addr) {  /* 文
法エラー */
System.out.println("Systax error at line " + addr);
System.out.println(err_mes);
iseg.print(addr);
System.out.println();
System.exit(1);
}

public static void executeError (int addr) {                 /* 実
行時エラー */
System.out.println("Execute error at line " + addr);
iseg.print(addr);
System.out.println();
System.exit(1);
}

public static void executeError (String err_mes, int addr) { /* 実
行時エラー */
System.out.println("Execute error at line " + addr);
System.out.println(err_mes);
iseg.print(addr);
System.out.println();
System.exit(1);
}
```

```
}
```

A.2.10 VSMLexer.java

```
import java.util.Vector; //ベクトル用

public class VSMLexer implements Operators {
InputFile inFile;    /* InputFile クラスのインスタンス（入力ファイル）*/

//コンストラクタでは，入力ファイルの読込みと，各種初期化を行う.
public VSMLexer(String fname) {
//入力ファイルを開く
inFile = new InputFile(fname);
}

public InstractionSegment makeIseg(boolean debugSW) {
InstractionSegment iseg = new InstractionSegment(debugSW);
while (true) {
Instraction inst = nextOperator();
if (inst.operator == EOF) break;
iseg.appendCode(inst);
}
for (int i=0; i<iseg.size; i++) iseg.checkAddress(i);
return iseg;
}

public Instraction nextOperator() {                    /* 字句解析 次のオペレータを得る */
int operator = NOP;
int operand = -1;
String str = "";
char c;
boolean csign = false;
boolean existOp = false;

do {
c = skipSpace();
} while(c == '\n');                                    /* 空白をスキップ */
```

```
if (c == '\0') operator = EOF;                  /* End of file */
else if (Character.isUpperCase(c)) {
     str = extractWord(c);
switch (c) {
case 'A':
if (str.equals("ADD")) operator = ADD;              /* ADD */
else if (str.equals("ADDLHS")) operator = ADDLHS;   /* ADDLHS */
else if (str.equals("AND")) operator = AND;         /* AND */
else if (str.equals("ASSGN")) operator = ASSGN;     /* ASSGN */
else if (str.equals("ASSGNS")) operator = ASSGNS;   /* ASSGNS */
else syntaxError("Unknown operator : "+str);
break;
case 'B':
if (str.equals("BEQ")) operator = BEQ;              /* BEQ */
else if (str.equals("BGE")) operator = BGE;         /* BGE */
else if (str.equals("BGT")) operator = BGT;         /* BGT */
else if (str.equals("BLE")) operator = BLE;         /* BLE */
else if (str.equals("BLT")) operator = BLT;         /* BLT */
else if (str.equals("BNE")) operator = BNE;         /* BNE */
else syntaxError("Unknown operator : "+str);
break;
case 'C':
if (str.equals("CALL")) operator = CALL;            /* CALL */
else if (str.equals("COMP")) operator = COMP;       /* COMP */
else if (str.equals("COPY")) operator = COPY;       /* COPY */
else if (str.equals("CSIGN")) operator = CSIGN;     /* CSIGN */
else syntaxError("Unknown operator : "+str);
break;
case 'D':
if (str.equals("DEC")) operator = DEC;              /* DECFR */
else if (str.equals("DECDR")) operator = DECFR;     /* DECFR */
else if (str.equals("DIV")) operator = DIV;         /* DIV */
else if (str.equals("DIVLHS")) operator = DIVLHS;   /* DIVLHS */
else syntaxError("Unknown operator : "+str);
break;
case 'E':
if (str.equals("ERROR")) operator = ERROR;          /* ERROR */
else syntaxError("Unknown operator : "+str);
```

```
break;
case 'H':
if (str.equals("HALT")) operator = HALT;            /* HALT */
else syntaxError("Unknown operator : "+str);
break;
case 'I':
if (str.equals("INC")) operator = INC;              /* INC */
else if (str.equals("INCFR")) operator = INCFR;     /* INCFR */
else if (str.equals("INPUT")) operator = INPUT;     /* INPUT */
else if (str.equals("INPUTC")) operator = INPUTC;   /* INPUTC */
else if (str.equals("INPUTS")) operator = INPUTS;   /* INPUTS */
else syntaxError("Unknown operator : "+str);
break;
case 'J':
if (str.equals("JUMP")) operator = JUMP;            /* JUMP */
else syntaxError("Unknown operator : "+str);
break;
case 'L':
if (str.equals("LOAD")) operator = LOAD;            /* LOAD */
else if (str.equals("LOADS")) operator = LOADS;     /* LOADS */
else syntaxError("Unknown operator : "+str);
break;
case 'M':
if (str.equals("MOD")) operator = MOD;              /* MOD */
else if (str.equals("MODLHS")) operator = MODLHS;   /* MODLHS */
else if (str.equals("MUL")) operator = MUL;         /* MUL */
else if (str.equals("MULLHS")) operator = MULLHS;   /* MULLHS */
else syntaxError("Unknown operator : "+str);
break;
case 'N':
if (str.equals("NOP")) operator = NOP;              /* NOP */
else if (str.equals("NOT")) operator = NOT;         /* NOT */
else syntaxError("Unknown operator : "+str);
break;
case 'O':
if (str.equals("OR")) operator = OR;                /* OR */
else if (str.equals("OUTPUT")) operator = OUTPUT;   /* OUTPUT */
else if (str.equals("OUTPUTC")) operator = OUTPUTC; /* OUTPUTC */
else if (str.equals("OUTPUTL")) operator = OUTPUTL; /* OUTPUTL */
```

```
else if (str.equals("OUTPUTS")) operator = OUTPUTS; /* OUTPUTS */
else syntaxError("Unknown operator : "+str);
break;
case 'P':
if (str.equals("PARA")) operator = PARA;            /* PARA */
else if (str.equals("POP")) operator = POP;         /* POP */
else if (str.equals("POPS")) operator = POPS;       /* POPS */
else if (str.equals("POSTDEC")) operator = POSTDEC; /* POSTDEC */
else if (str.equals("POSTINC")) operator = POSTINC; /* POSTINC */
else if (str.equals("PREDEC")) operator = PREDEC;   /* PREDEC */
else if (str.equals("PREINC")) operator = PREINC;   /* PREINC */
else if (str.equals("PUSH")) operator = PUSH;       /* PUSH */
else if (str.equals("PUSHI")) operator = PUSHI;     /* PUSHI */
else if (str.equals("PUSHP")) operator = PUSHP;     /* PUSHP */
else if (str.equals("PUSHS")) operator = PUSHS;     /* PUSHS */
else syntaxError("Unknown operator : "+str);
break;
case 'R':
if (str.equals("RAND")) operator = RAND;            /* RAND */
else if (str.equals("RET")) operator = RET;         /* RET */
else if (str.equals("REMOVE")) operator = REMOVE;   /* REMOVE */
else syntaxError("Unknown operator : "+str);
break;
case 'S':
if (str.equals("SETFR")) operator = SETFR;          /* SETFR */
else if (str.equals("SUB")) operator = SUB;         /* SUB */
else if (str.equals("SUBLHS")) operator = SUBLHS;   /* SUBLHS */
else if (str.equals("SYNC")) operator = SYNC;       /* SYNC */
else syntaxError("Unknown operator : "+str);
break;
case 'X':
if (str.equals("XOR")) operator = XOR;              /* XOR */
else syntaxError("Unknown operator : "+str);
break;
     default:
syntaxError("Unknown operator : "+str);
break;
}
} else syntaxError("Illegal charactor : "+c);
```

```
c = skipSpace();                                    /* 空白をスキップ */

if (c == '-') {                                     /* 符号 */
csign = true;
c = inFile.nextChar();
}
if (c == '0') {                                     /* 符号無し整数
(0) */
operand = 0;
c = skipSpace();
if ( c != '\n' && c != '\0' ) syntaxError("Illegal charactor : "+c);
existOp = true;
} else if (Character.isDigit(c)) {                  /* 符号無し整数 */
operand = extractIntValue(c);
c = skipSpace();
if ( c != '\n' && c != '\0' ) syntaxError("Illegal charactor : "+c);
existOp = true;
} else if (c != '\n' && c != '\0' )
syntaxError("Illegal charactor : "+c);

if (csign) operand *= -1;
checkOperand(operator, operand, existOp);

return new Instraction(operator, operand);
    }

public int extractIntValue(char c) {        /* cで始まる整数を得
る */
int v = Character.digit(c, 10);          /* 文字を整数に変換 */
while (Character.isDigit(inFile.nextc)) {
c = inFile.nextChar();
v = v * 10 + Character.digit(c, 10);
}
return v;
}

public String extractWord(char c) {         /* cで始まる文字列を
得る */
```

```
String s = String.valueOf(c);
while (Character.isUpperCase(inFile.nextc)) {
c = inFile.nextChar();
s = s + c;
}
return s;
    }

public char skipSpace() {                /* 空白をスキップ */
char c;
do {
c = inFile.nextChar();
if (c == '#')
do {
c = inFile.nextChar();
} while (c != '\n' && c != '\0');
if (c == '\n' || c == '\0') break;
} while (c == ' ' || c == '\t');
return c;
}

public void checkOperand(int operator, int operand, boolean e) {
switch(operator) {
case NOP:
case ASSGN:
case ADD:
case ADDLHS:
case SUB:
case SUBLHS:
case MUL:
case MULLHS:
case DIV:
case DIVLHS:
case MOD:
case MODLHS:
case CSIGN:
case AND:
case OR:
case NOT:
```

```
case XOR:
case COMP:
case COPY:
case PUSHP:
case REMOVE:
case LOAD:
case INC:
case DEC:
case PREINC:
case PREDEC:
case POSTINC:
case POSTDEC:
case RET:
case INPUT:
case INPUTC:
case OUTPUT:
case OUTPUTC:
case OUTPUTL:
case RAND:
case ASSGNS:
case LOADS:
case INPUTS:
case OUTPUTS:
case PARA:
case HALT:
case SYNC:
case EOF:
if (e)
syntaxError("Illegal operand" + operand);
break;
case PUSH:
case PUSHI:
case POP:
case SETFR:
case INCFR:
case DECFR:
case JUMP:
case BEQ:
case BNE:
```

```
case BLT:
case BLE:
case BGT:
case BGE:
case CALL:
case PUSHS:
case POPS:
if (!e)
syntaxError("Illegal operand");
break;
default:
syntaxError("Illegal operator");
break;
}
}

public void syntaxError() {                 /* 文法エラー */
System.out.println("Systax error at line " + inFile.linenum);
System.out.println(inFile.line);
inFile.closeFile();
System.exit(1);
}

public void syntaxError(String err_mes) {   /* 文法エラー */
System.out.println("Systax error at line " + inFile.linenum);
System.out.println(err_mes);
System.out.println(inFile.line);
inFile.closeFile();
System.exit(1);
}
}
```

A.3 n 個のデータの和を計算する PRAM 用並列言語プログラム

```
main(){
int a[16];
int i;
```

```
parallel(0,15)
a[\$p]=\$p+1;

for(i=1; i<16; i*=2)
parallel(0,16/2/i)
a[\$p]=a[2*\$p]+a[2*\$p+1];

write(a[0]);
}
```

A.4 付録 3 を PRAM コンパイラによって変換した並列アセンブラプログラム

```
PUSHI   0
PUSHI   15
PARA
PUSHI   0
PUSHP
ADD
PUSHP
PUSHI   1
ADD
ASSGN
REMOVE
SYNC
PUSHI   16
PUSHI   1
ASSGN
REMOVE
PUSH    16
PUSHI   16
COMP
BGE     56
JUMP    29
PUSHI   16
COPY
LOAD
```

```
PUSHI   2
MUL
ASSGN
REMOVE
JUMP    16
PUSHI   0
PUSHI   8
PUSH    16
DIV
PARA
PUSHI   0
PUSHP
ADD
PUSHI   0
PUSHI   2
PUSHP
MUL
ADD
LOAD
PUSHI   0
PUSHI   2
PUSHP
MUL
PUSHI   1
ADD
ADD
LOAD
ADD
ASSGN
REMOVE
SYNC
JUMP    21
PUSH    0
OUTPUT
HALT
```