TM-1139

# A'UM-90 並列版処理系 使用説明書

小西 弘一、柳田 伸二、
丹下 利雄（日電）

December, 1991



ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

# A'UM-90 並列版処理系

# 使用説明書

第1.00版 (1991年11月11日)

## 本説明書の構成

本説明書は A'UM-90 並列版処理系$\beta$リリースに含まれるシステムの使用説明書を集めたものであり、以下に示す3部から構成される。

第一部 A'UM-90 コンパイラ AUMC 使用説明書

コンパイラ AUMC の使用法を述べている。

第二部 PAS(Parallel A'Um System) 使用説明書

実行時システム PAS の使用法を述べている。

第三部 PAS(Parallel A'Um System) Debugger 利用マニュアル

デバッガの使用法を述べており、またコマンドの参照マニュアルを含む。

# A'UM-90 コンパイラ

# AUMC

# 使用説明書

第1.02版

Nov. 11. 1991

本説明書は、A'UM-90 コンパイラ aumc の以下の版を対象にしている。

aumc version 0.2

# 1 概要

```
aumc [-qvdncrps] [source-filename]
```

A'UM コンパイラは、A'UM プログラムをコンパイルして、A'UM 抽象命令のニモニック列を作成する。A'UM コンパイラの出力は、そのまま A'UM 並列実行系 PAS の入力として与えることができる。

各オプションの機能については See Chapter 2 [オプション], page 3

'source-filename'には、ソースファイル名を与える。省略した場合、標準入力が入力となる。

コンパイル結果はファイルに出力される。出力ファイル名は、ソースファイル名を元に決められる。ソースファイル名が与えられていない場合には、コンパイル結果は出力されない。

## 1.1 ソースファイル

ソースファイルは、A'UM プログラムを含むファイルである。ソースファイル名は、末尾が '.aum'で終っていなければならない。

## 1.2 出力ファイル

コンパイル結果はファイルに出力される。出力ファイルは、A'UM 抽象命令のニモニック列に加えて、A'UM 並列実行系 PAS に内蔵されたアセンブラが必要とする疑似命令を含んでおり、そのまま PAS の入力とすることができる。

出力ファイル名は、入力ファイル名の末尾から '.aum'を取り除き、代わりに '.aao'を加えて作られる。

例:

入力ファイル名 'prime.aum' -> 出力ファイル名 'prime.aao'

# 2 オプション

現在利用可能なオプションのほとんどは、コンパイラ自身のデバッグのためのオプションである。これらのオプションによる出力は、'-q'オプションを指定しない限り、生成されるコードと同じファイルに出力される。

'-q' コードを生成せず、解析だけを行う。他のオプションによる結果を標準出力に向ける。

'-v' 起動された aumc のバージョン番号を標準出力に表示する。

'-d' Yacc で書かれた構文解析部がデバッグメッセージを出力する。

'-n' 構文解析の結果を元のプログラムに近い形式で出力する。

'-c' ボラタイルクラス定義の入れ子構造に関する、変数のスコープの範囲についての解析結果を出力する。

'-r' 変数の参照回数についての解析結果を出力する。ボラタイルクラス定義の内外両方に現れる変数は、中と外を別の変数として扱っている。

'-p' クラス定義中に現れるメッセージプロトコルの一覧を出力する。

'-s' 各ボラタイルクラス定義について、それらの持つメソッドのプロトコルの引数の最大個数を出力する。

# 3 エラーメッセージ

AUMC は次の形式でエラーメッセージを出力する。

ファイル名':'行番号':'クラス名':'エラーメッセージ

複数のエラーメッセージが出力される場合、2つ目以降のエラーメッセージは必ずしも正しくない。

エラーメッセージの意味を以下に示す。

parse error
: 構文上の誤りがある。

variable ... has two inlets
: 同一スコープ内に、同じ名前のインレットが複数現れている。これに続くメッセージは、ボラタイルクラス定義の入れ子関係による木構造中における、検出された二つのインレットの位置関係を示す。

（続くメッセージがない場合）
: 同じボラタイルクラス定義中にある。

at different levels in nested volatile classes
: 先祖 - 子孫関係にある異なるボラタイルクラス定義中にある。

exported to different volatile classes in a method
: 先祖 - 子孫関係にない、共通の先祖を持つ異なるボラタイルクラス定義中にある。

また、以下に示すメッセージはプログラム中のエラーではなく、AUMC に起因する問題が生じたことを表す。

register ... may have been destroyed.
: メッセージの送信前に宛先が閉鎖される結果、レジスタ上の値がゴミになっている恐れがある。AUMC の既知の問題点の一つ (see Chapter 4 [問題点], page 5)。入力プログラムに誤りがあるわけではないが、関係するアクションの順序を入れ換えることによって回避できる。

System error:...
: AUMC 自身になんらかの（現在の版では実現されていない機能などの）問題がある。

# 4 問題点

AUMC の既知の問題点を以下に挙げる。

ストリームへの送信と閉鎖の順序

ある変数がプログラム中最後に出現する場所が、送信式の宛先である場合、その送信以前にその変数に割り当てられたレジスタの値を破壊するコードを生成することがある。例えば、次のプログラムでは、メッセージ':bar'の宛先'X'はレジスタ R0 に割り当てられるが、R0 の値は、':bar'が送信される前に閉鎖されてしまう。

例：

```
:foo(^X, X) -> X :bar.
```

この場合、メッセージの宛先が変数の最後の出現とならないように書き変えれば実行可能なコードが出力される。上記の例では、'X'の最後の出現をダミーの変数'Y'に置き換え、この送信式よりも後ろに、'^Y'を'X'に接続する式をおけばよい（下記の例）。

例：

```
:foo(^X, X) -> Y :bar, X = ^Y.
```

ボラタイルクラス定義

現在 AUMC でコンパイルできるボラタイルクラス定義は、以下の形式を満たすもののみである。

```
(比較演算式) ? (
:'true -> ... .
:'false -> ... .
)
```

但し

比較演算式
　以下の二項演算子のなかの一つによる式。
　'==', '>', '<', '=<', '>='

すなわち、コンパイルできるのは':'true'と':'false'をセレクタとする二つのメソッド定義だけを持ち、インターフェース式として比較を行う式を持つボラタイルクラス定義である。

特に、比較演算を行う式であっても演算子'!='はコンパイルできないので注意が必要である。

# 5 インストール

AUMC ソースファイルは、以下のディレクトリ階層に収められている。

'Aumc'
- 'Aumc/scope'
- 'Aumc/syntax'
- 'Aumc/code'
- 'Aumc/register'
- 'Aumc/struct'

コマンド aumc を作成するには、ディレクトリ 'Aumc' においてmakeを実行する。

# 6 コンパイル例

```
csseq1:79% ls
hello.aum
csseq1:80% cat hello.aum
class aum.
:go(^_) ->
    #hello :sayIt.
end.

class hello.
:sayIt ->
    #tio :print("Hello, sailor").
end.
csseq1:81% aumc hello.aum
csseq1:82% ls
hello.aao   hello.aum
csseq1:83% cat hello.aao
     %
     .class #aum;
     .classref #hello;
     .pid 'sayIt', sayIt;
     .pid 'go/+', "go", 1, +;
     .method 'go/+', "go", 1, 0x1;
'go/+':
     close R0;
     create_instance R0, #'hello';
     send R0, 'sayIt';
     close R0;
     descend;
     .end #aum;
     %
     .class #hello;
     .classref #tio;
     .pid 'print/+', "print", 1, +;
     .pid 'sayIt', sayIt;
     .method 'sayIt', sayIt;
'sayIt':
     create_instance R0, #'tio';
     create_string2 R1, "Hello, sailor";
     send R0, 'print/+', R1;
     close R0;
     descend;
     .end #hello;
csseq1:84%
```

# PAS (Parallel A'Um System)

# 使用説明書

第 1.14 版 (1991 年 9 月 4 日)

# 1 概要

A'Um 言語処理系 PAS (Parallel A'Um System) は、AUM-90 抽象マシン命令で書かれたファイルを入力として、その実行を行なうプログラムである．実行オブジェクト名は pas であり，以下のようにして起動する．

```
pas [options] file1 [ file2 .. filen] [-- [arg1 arg2 .. argn]]
```

ここで file1, file2, .., filen は、AUM-90 抽象マシン命令で書かれたファイルである．このファイルは A'Um 言語コンパイラ AUMC (A'Um Compiler) によって作成される．

引数の '--' は、pas の入力引数と A'Um プログラムのトップレベルに渡す引数を区切るために使う．'--' のあとに続く引数 arg1, arg2, .., argn は、A'Um プログラムに文字列の配列として渡される．

## 1.1 A'Um プログラムへの引数の渡し方

オプション '--' 以下に続く引数は，A'Um プログラムへの引数として解釈する．以下の例のようにして pas を起動すると;

```
% pas -- jack 11 queen 12 king 13
```

各引数は文字列としてプログラムに渡される．すなわちプログラム;

```
class aum.
    :go(^Args) ->...
end.
```

の変数 *Args* には次の様な配列となる．

*Args* ⇒ {"jack", "11", "queen", "12", "king", "13"}

## 1.2 通信モードについて

pas は非共有メモリシステムを想定した並列処理系である． このため異なる PE に存在するオブジェクト間で通信を行なおうとすると，特別なメッセージ通信形態をとる必要がある．この通信形態を外部通信モードと呼ぶ．

これに対して，同一のPEに存在するオブジェクト間での通信形態もある．これを内部通信モードと呼ぶ．

特別なオプションが指定されない限り，同一のPEに存在するオブジェクト同士の通信は‘内部通信モード’に，別PEに存在するオブジェクト同士の通信は‘外部通信モード’に設定される．

### 1.2.1 外部通信モード

外部通信モードはオブジェクト間通信の一つの形態である．通常は異なるPEにあるオブジェクト間の通信に利用する．特別なオプション(例えば‘-external’)が指定された場合には，同一のPE内でも外部通信モードで通信を行なう場合がある．

外部通信モードでのメッセージ通信は，メッセージの宛先ならびにメッセージの引数は外部番地として表現される．外部番地とはPE番号と輸出表内の番号のペアによって表される．

### 1.2.2 内部通信モード

内部通信モードはオブジェクト間通信の一つの形態である．この通信モードは同一のPE内にあるオブジェクト間通信に使われる．外部通信モードに較べて，内部番地を外部番地に，あるいはその逆の変換がないのでオブジェクト間で高速にメッセージを通信することができる．

このモードで異なるPEにあるオブジェクト同士の通信を行なうことはできない．

# 2 オプションの説明

現在 pas には以下のオプションがある.

'-help' オプションの簡単な説明を表示する.

'-verbose'
実行ファイルのロード等に関する詳しい情報を表示する.

'-trace' メソッドの実行, スロットの更新ならびにメッセージの送受信のトレースを行なう.

'-itrace' AUM-90 抽象命令レベルのトレースを行なう.

'-ptrace' PE 間のメッセージ通信のトレースを行なう. ただしこのオプションは, '-Npe *N*' を指定して 2 台以上の PE を用いるモードで pasを立ち上げた時, あるいは '-external' のオプションが指定された時に有効である. '-showrc' オプションが指定されていれば, 外部参照回数のトレースも表示する.

'-showrc' トレース時に内部参照回数の表示を行なう. '-ptrace' オプションが指定されていれば, 外部参照回数の表示も行なう.

'-log *file*' '-trace', '-itrace', '-ptrace' の出力ファイルを *file*.0 とする. オプション '-Npe *N*' によって 2 台以上の PE を用いる場合には pas が起動されていれば, 各 PE 番号に応じてそれぞれ *file*.0, *file*.1, .., *file*.*N* が出力ファイルとなる.

'-external'
全てのオブジェクト間通信を外部通信モードで行なう.

'-external-odd'
システムデバッグ用のオプション.

'-external-even'
システムデバッグ用のオプション.

'-statistics'
pas の実行終了時に使用した処理時間等の情報を出力する.

'-Npe *N*' 何台の PE を使うか指定する. このオプションが指定されなければ *N* として 1 が指定されたものとみなす. また, *N* として 0 が指定されたならば, そのシステムで利用できる最大の PE 数が指定されたものとみなす.

'-T*name size*'
*name* で指定されるテーブルの大きさを指定する. name として以下のものが指定できる.

| | |
|---|---|
| *atom* | アトムテーブルの大きさを指定する. デフォルトの大きさは 4096 である. |
| *protocol* | プロトコルテーブルの大きさを指定する. デフォルトの大きさは 1024 である. |
| *export* | 輸出表の大きさを指定する. デフォルトの大きさは 4096 である. |
| *import* | 輸入表の大きさを指定する. デフォルトの大きさは 4096 である. |
| *timeslice* | スケジューリングのタイムスライス値を指定する. 一般オブジェクトの実行は, 届いたメッセージがある限りそのオブジェクトを続けて実行する. ただし, そ |

のオブジェクトがタイムスライス値を越える回数のメッセージを送信すると，実行を一時中断し，別なオブジェクトの実行を試みようとする．*timeslice*はこのメッセージ回数を与える．デフォルトの大きさは 10000 である．

'-S*name*'

新しくインスタンスオブジェクトを作る時に他の PE か同一の PE かを決める方法を指定する．*name* として以下のものが指定できる．ただし，1 PE で処理系が起動された時にはこのオプションは意味をなさない．デフォルトはラウンドロビン方式である．

*round-robin*

ラウンドロビン方式の生成を行なう．各 PE は個別にどの PE にオブジェクトを生成するかのカウンタを持っている．

'-Wall' 実行終了時に輸出入表にオブジェクトが残っている場合，警告メッセージを出力する．

'-D*key*' *key* で指定されるキーワードのデバッグスイッチをオンにする．*key*で指定できるキーワードには以下のものがある．

*connect* ストリーム連結時のデバッグ情報を出力する．

# 3 問題点

以下に現在のバージョンの pas で解っているバグについて述べる.

- PE 間で文字列を渡す時に実行効率を考えてコピーを行なっている. このため文字列を書き換えるプログラムの場合, 1 PE で実行した時と $N$ PE で実行した時に実行結果が異なる場合がある. 将来は陽に指定しない限り PE 間での文字列のコピーは行なわないようにする予定である. また, 書き込み禁止の文字列を用意するかも知れない.
- 組込メソッドを実行する時に引数が未接続の場合, メッセージを反転して送信するようにしている. ところがリストや配列の場合はこの方法ではうまくいかないことが解っている. 例えば次のようなプログラムで

  *Vector* `:elt(`*Index*`, ^`*X*`) :set(`*2*`, `*0*`).`

  *Index* が決まっていなかったら, 先に `set/2` メソッドの実行をすることがある. 以下のメソッドがこの問題点を持っている.

  `elt/++, elt/+-, set/++, set/+-, dir/+-`
- PE 間で倍精度浮動小数点数を送ると, 引数に倍精度小数点数がある場合はその値のコピーが送られるが, PE 間での接続の場合には倍精度小数点数のある番地が送られる. これはバグではないが実行効率上問題となるので別な方法を考える必要がある.

# PAS(Parallel A'Um System) Debugger

# 利用マニュアル

第 1.3 版

(1991 年 11 月 5 日)

# Table of Contents

# 1 はじめに

## 1.1 はじめに

本デバッガは基本的にオブジェクトの内部状態の表示により実行上のオブジェクトの状態変化を明らかにしてメッセージ実行の誤りを検査し，オブジェクトに対する意味上のメッセージ到着順の誤り，或いは到着先での誤りを発見するためのツールである．

## 1.2 デバッガのインストール

PASが/PASというディレクトリにインストールされているとき，デバッガのディレクトリ（Symmetry版では，/PAS/debugger/symmetry）でmakeを実行するとpas-dbというファイル名で実行ファイルが，生成される．

## 1.3 デバッガの起動方法

起動方法，オプションは，PAS準拠となっているため，詳細はPASの利用マニュアルを参照（ただしオプション-ptraceを指定すると，デバッガの表示が見にくくなるため，使用しない方が良い．）

# 2 デバッガのプロセス構成

PASデバッガでは，デバッガ処理部分を付加されたPASの処理系のプロセスを起動し，それらと同レベルで別にプロセスを余分に生成する．そのプロセスがデバッガのコマンド入力，PASプロセスからのテキスト情報の出力等の処理を行なうように実装を行なっている．

## 2.1 デバッガ実行時のプロセス構成

1CPU環境でのデバッガは，PASの中にコマンド処理を行い，情報を直接提供する形で実行される．分散環境での実行を行うように設定し，デバッガを起動すると，PASのプロセスの他にデバッグの為のプロセスも起動される．即ち，PASデバッガでNpeオプションをNに指定すると，実際にはN+1個のプロセスが，起動される事になる．デバッガのプロセス内では，標準入力からのコマンド入力を解析し，PASの各プロセスとの調停を行なうListener Shellが実行状態にある．ユーザーは，このListener Shellにコマンドを与え，デバッグの種々の操作を行なう．Listener Shell-PAS間の通信には，PASの通信系を用いている．PASのプロセスでは，Listener Shellからのメッセージを受け取ると，それに対する応答として，各種の情報をテキストデータで返送する．プログラムの実行開始を示すメッセージが送られてくると，schedulerに実行を移し，デバッグのために仕掛けられたブレークポイントまで実行を進め，そこでデバッガルーチンに実行を移す．

## 2.2 デバッガ使用上の制約

デバッガの実装上の現バージョンの制約を以下に述べる．

メッセージのデバッグに於いて，デバッグが可能なプロトコールＩＤは，引数を持つものに限られる．

分散環境でのトレース情報は，一定の文字列長に区切って各PEから送られるため，それらが一つの画面に混在する形で表示される．

## 2.3 プロセス構成の遷移とプロンプト

デバッガを起動すると，ユーザプログラムを実行する前の状態で，コマンド待ち状態となっているこのときのプロンプトはn:A'UmDB>出有りn:のnには，デバッガプロセスが割り当てられたプロセッサ番号を示している．次に，ユーザプログラム実行時にブレークポイントで停止，コマンド待ち状態になった場合のプロンプトは，n:A'UmDB-Exe>である．この場合のnは，現在デバッグの対象となっているのは，どのプロセッサのPASプロセスかを示している．

# 3 コマンドリファレンス

デバッガの Listener Shell のコマンド，及びその機能について

## 3.1 ユーザプログラムの実行

プロンプトの状態 n:A'UmDB> で使用可能のコマンド

コマンド名 **run**

書式 run

省略形 r

コマンド名 **execute**

書式 execute

現時点で読み込まれているユーザプログラムのファイルのクラス aum のインスタンスオブジェクトを PE0 に生成し，メッセージ:go(X) （X の内容は、オプションで指定可）を送信し，各 PAS のスケジューラへ制御を渡す.

## 3.2 静的情報の参照

プロンプトの状態 n:A'UmDB> n:A'UmDB-Exe> 共通に使用可能のコマンド

コマンド名 **info class**

書式 info class *classname*

クラス名で指定されたクラスのデバッグフラグの状態を表示する.

コマンド名 **info classes**

書式 info classes

読み込んだクラス名の一覧（他のクラスを継承している場合，そのクラス名を段下げ表示する）を表示する.

コマンド名 **info pidlist**

書式 info pidlist

全プロトコール ID を表示する.

## 3.3 デバッグ情報の設定，解除

break, unbreak, trace, untrace のコマンド使用に際して，入力データの構文を以下に示す.

**input_data :**

':'methodname[モード]
　　メッセージの指定
classname
　　クラスの指定
classname{'@','!'}slotname
　　スロットの指定
classname':'methodname[modes]
　　メソッドの指定
classname'_0x'オブジェクト番号
　　オブジェクトの指定
classname'_0x'オブジェクト番号{'@','!'}slotname
　　オブジェクトのスロットの指定
classname'_0x'オブジェクト番号':' methodname[modes]
　　オブジェクトのメソッドの指定

コマンド名 **break**
　　書式 break *input_data*

ブレークポイントを設定する.

コマンド名 **unbreak**
　　書式 unbreak *input_data*

ブレークポイントを解除する.

コマンド名 **trace**
　　書式 trace *input_data*

トレースモードを設定する.

コマンド名 **untrace**

書式 untrace *input_data*

トレースモードを解除する.

## 3.4 実行時でのみ使用可能なコマンド

プロンプトの状態 n:A'UmDB-Exe> でのみ使用可能のコマンド

コマンド名 **step,next**

書式 step,next

省略形 s,n

ブレークループ中のオブジェクトにたいして，メッセージ（或いは抽象命令）を一つだけ実行させる．ただし，PASのプロセスが複数の場合，動作不定となる場合がある.

コマンド名 **stepi**

書式 stepi

省略形 si

抽象命令モードが設定されていれば，抽象名命令単位でのステップ動作を行う．抽象命令モードが，設定されていなければ，step と同様な動作を行う．ただし，PAS のプロセスが複数の場合，動作不定となる場合がある.

コマンド名 **continue**

書式 continne

省略形 cont , c

次のブレークポイントまで実行を再開する.

コマンド名 **info active**

書式 info active

現在実行状態にあるオブジェクトを表示する.

コマンド名 **info waiting**

書式 info waiting

現在実行待機状態にあるオブジェクト名の一覧を表示する.

コマンド名 **info register**

書式 info register

省略形 info reg

全レジスタ内容の表示

コマンド名 **print**

書式 print *classname'_0x'*オブジェクト番号 {@,}*slotname*

オブジェクトのスロット名で指定されたスロットの内容を表示する.

書式 print *classname'_0x'*オブジェクト番号

オブジェクトの内部状態を表示する.

## 3.5 オンラインリファレンス

どのプロンプトでも実行可能

コマンド名 **info**

表記 info

キーワードに対する情報を表示する.

コマンド名 **help**

表記 help

コマンドリファレンスを表示する.

## 3.6 その他

コマンド名 **trace on**

書式 trace on

省略形 tron

抽象命令単位のトレースモードを各 PAS に設定する.

コマンド名 **trace off**

書式 trace off

省略形 troff

各 PAS の抽象命令単位のトレースモードを解除する.

(システム起動時デフォルトは Off)

コマンド名 **output**

書式 output *n(disenable trace pe number)*

指定した PE のトレース出力の表示, 非表示を切り替える.

## 3.7 終了

どのプロンプトでも実行可能

コマンド名 **quit,exit**

書式 quit

省略形 q

デバッガを終了する.

# Command Index