

Common Region *On Line*

Customer Services Enterprise Customer Support Knowledge Center 2

OpenVMS

1995年 2月

Vol.38

VAXからAXPへの移行



コンパックコンピュータ株式会社
カスタマーサービス統括本部

COMPAQ

Vol.38

VAXからAXPへの移行

目次

第1章 はじめに

1.1 OpenVMS とは?	1-2
1.2 VMS とOpenVMS の相違	1-3
1.3 VAX とAlpha AXP の OpenVMS の相違	1-4

第2章 オペレーション編

2.1 EDT エディタで日本語が文字化け	2-2
2.2 OpenVMS AXP には JTPU コマンドが無い	2-3
2.3 AXP を使っているか VAX を使っているか調べたい	2-4
2.4 Alpha AXP のブートストラップ	2-5
2.4.1 ブートコマンドのパラメータについて	2-5
2.4.2 典型的なブートコマンド	2-6
2.5 デフォルトブートデバイス、パラメータの設定	2-8
2.6 Ethernet ポートの切り替え	2-9
2.7 スタンドアロンバックアップキットを作成できない (1)	2-10
2.8 スタンドアロンバックアップキットを作成できない (2)	2-11

第3章 DEF Fortran プログラミング編

3.1 引数の数を調べる VAX MACRO のサブルーチン	3-2
3.2 キャラクタ定数をサブルーチンに渡す時の注意	3-6

3.3 コンパイル時に MISALIGN の警告メッセージ	3-8
3.4 共有コモンブロックの利用	3-13
3.4.1 リンクコマンドを変更する場合	3-14
3.4.2 プログラムにディレクティブを追加する場合	3-15

第4章 DEC Cプログラミング

4.1 VAX C で書かれたプログラムのコンパイル時にエラー	4-2
4.2 DEC C のリンクコマンド.....	4-4
4.3 SYS\$LIBRARY にヘッダファイルが無い	4-5
4.4 union のメンバがずれる.....	4-6
4.5 DEC C から共有コモンブロックへのアクセス.....	4-9

第5章 OpenVMSプログラミング編

5.1 SYS\$CRMPSC() を使用する際の注意点	5-2
5.2 コードやデータをページ境界に配置する方法	5-9
5.3 コンディションハンドラの設定方法	5-10
5.4 ランタイムライブラリの作成方法	5-15
5.5 浮動小数点データの変換方法	5-17
5.5.1 DEC C を使用する場合	5-17
5.5.2 DEC Fortran を使用する場合	5-19
5.6 SYS\$SYSTEM:SYS.STB がない	5-22

第 1 章 はじめに

1.1 OpenVMS とは?

質問:OpenVMS とは?その歴史は?

OpenVMS は V5.5 以前は VMS (Virtual Memory System)と呼ばれていました。VMS は 1976 年、DEC の新しい 32 ビット仮想アドレスコンピュータである VAX (Virtual Adress eXtension) のために考案されました。最初の VAX モデル VAX-11/780 の開発コードネームが "Star" であったため、VMS オペレーティングシステムは "Starlet" (Star の女性形) というコードネームで呼ばれていました。"Starlet" という名称は今でもシステムライブラリ STARLET.OLB 等にその名前を残しています。初めてカスタマにリリースされた VMS はバージョン X0.5 で、これは VAX-11/780 のハードウェアベータテストのためのものでした。その後 VAX/VMS バージョン V1.0 が 1978 年に VAX-11/780 とともに出荷されました。

OpenVMS は全て DEC 社によって設計されました。OpenVMS は PDP-11 上で稼動する RSX-11M を 32 ビットの仮想アドレス方式に拡張した、後継オペレーティングシステムとして考案されました。当初の設計者と OpenVMS のプログラマーの多数は既に RSX-11M の経験をもっていたため RSX-11M から多くの概念が OpenVMS に移行されました。OpenVMS は 32 ビット、マルチタスク、マルチプロセスの仮想メモリオペレーティングシステムです。

現在 VAX と Alpha AXP コンピュータシステムに実装されています。OpenVMS オペレーティングシステムの機能についての詳細は、OpenVMS ソフトウェア製品仕様書 (OpenVMS Software Product Descriptions) をお読みください。

1.2 VMS とOpenVMS の相違

質問:VMS とOpenVMS はどこが違うのですか?

VMS とOpenVMS は同じオペレーティングシステムを表す 2 つの名前です。最初、このオペレーティングシステムは VAX-11/VMS と呼ばれていましたが V2.0 において VAX/VMS に変更されました。VMS オペレーティングシステムが Alpha AXP プラットフォームに移植された時、VAX とAlpha AXP 両プラットフォームともOpenVMS と名称が変更されました。これは POSIX 等の工業規格を高度にサポートしていることを表しています。POSIX は UNIX システムの多くの機能を提供します。OpenVMS のライセンスには POSIX for OpenVMS の利用権が含まれています。ライセンスを新たに購入する必要はありません。他に必要なものは、ソフトウェアの配布メディアとマニュアル類が、これらは Consolidated Distribution CD-ROM, On-Line Document CD-ROM に含まれています。

OpenVMS という名称が初めて使用されたのが OpenVMS AXP V1.0 であったため、OpenVMS は Alpha AXP 専用で、"Open" が付かない VMS は VAX 専用なのだという誤った認識が広まってしまいました。DEC も公式には VAX のオペレーティングシステム V5.5 で名称を OpenVMS に変更していましたが、製品が OpenVMS という名称を使用するようになったのは V6.0 からです。現在、2 つのプラットフォームの上の OpenVMS を表す固有名詞は "OpenVMS VAX" と "OpenVMS AXP" です。

1.3 VAX と Alpha AXP の OpenVMS の相違

質問:VAX とAlpha AXP の OpenVMS はどこが違うのですか?

ほとんど相違はありません。OpenVMS V6.1 現在、VAX とAlpha AXP の両プラットフォームはほとんど機能的に同等です。たいいていのアプリケーションは再コンパイルするだけで移行可能です。ただし、両プラットフォームのアーキテクチャの違いに起因する相違点がいくつかあります。個々の問題についての詳細は、後のプログラミング編をお読みください。

- OpenVMS AXP では、デフォルトで使用される倍精度浮動小数点データ型は VAX G 浮動小数点型です。VAX ではデフォルトで D 浮動小数点数データ型が使用されていました。D 浮動小数点型を Alpha AXP で使用することも可能ですが、D 浮動小数点数型の変数は、G 浮動小数点型に変換されてから計算が行なわれ、そして結果がレジスタに格納される時、再度 D 浮動小数点型に変換されます。G 浮動小数点型は D 浮動小数点型に比べて小数部ビットが 3 ビット少ないため、異なった結果を得るアプリケーションプログラムがあるかも知れません。IEEE 浮動小数点データ型も OpenVMS AXP で利用可能です。
- データのアライメントを考えることは Alpha AXP では非常に重要です。Alpha AXP で最高の性能を得るためには、データをそれらの大きさの正確な倍数であるアドレスに配置する(自然境界に配置する)べきです。ワード境界にキャラクタ変数及び 32 ビットより小さい変数を配置すれば、最上のパフォーマンスを得ることが可能です。これらの作業は通常、コンパイラにより行なわれます。コンパイラがデータ配置を最適化できなかった場合には、警告メッセージを発行することがあります。
- DEC C は OpenVMS AXP プラットフォームに対して、DEC が提供する唯一の C コンパイラです。それは OpenVMS VAX の上の DEC C と互換性がありますが、多くの人々が精通している古い VAX C コンパイラといくつかの点で異なっています。マニュアルの /EXTERN_MODEL 修飾子、/STANDARD 修飾子に関する記述をお読みいただければ、多くの問題を解決することができます。
- Alpha AXP システムのページサイズは可変で、少なくとも 8K バイトです。このため DCL コマンドの \$ SHOW MEMORY コマンドや、\$CRMPSC システムサービスを使うアプリケーションに影響を与えます。ページサイズは \$GETSYI システムサービスで SYI\$ _PAGE_SIZE アイテムコードを使用することにより得ることができます。

VAX から AXP への移行については以下のマニュアルを参照してください。

- "A Comparison of System Management on OpenVMS AXP and OpenVMS VAX"
- "Migrating to an OpenVMS AXP System: Planning for Migration"
- "Migrating to an OpenVMS AXP System: Porting VAX MACRO Code"
- "Migrating to an OpenVMS AXP System: Recompiling and Relinking"

第 2 章 オペレーション編

2.1 EDT エディタで日本語が文字化け

質問:OpenVMS VAX V5.5-2 では、EDT エディタで日本語の表示ができました。ところが OpenVMS AXP では日本語が表示できません。なぜですか?

OpenVMS VAX の V5.5-2 まではカナサポートのために、日本語 VMS は共有ライブラリ JSY\$EDTSHR.EXEを提供していました。日本語 VMS のスタートアップ・プロシジャ SYS\$STARTUP:JSY\$STARTUP.COM は以下の論理名定義を行い英語版の提供する EDTSHR.EXE を置き換えていました。

```
$ DEFINE/SYSTEM/EXEC EDTSHR JSY$EDTSHR
```

この置き換えにより 従来は日本語表示が可能となっていたのです。しかし、OpenVMS AXP ではこの共有可能ライブラリは提供されていません。OpenVMS V6.1 以降の VAX でも JSY\$EDTSHR.EXE は提供されていません。将来提供される予定もありません。XTPU 等、他のエディタをご使用ください。

2.2 OpenVMS AXP には JTPU コマンドが無い

質問:OpenVMS VAX で JTPU エディタを使用していました。OpenVMS AXP には JTPU コマンドは無いようですが?

OpenVMS AXP では JTPU エディタのかわりに XTPU エディタが提供されています。XTPU エディタは JTPU エディタの上位互換エディタで、OpenVMS VAX の V6.0 以上でも使用可能です。以下のコマンドで起動してください。

```
$ EDIT/XTPU <filename>
```

また以下のシンボルを LOGIN.COM 等で定義しておけば、これまでと同じように \$ JTPU で XTPUエディタを起動することができます。

```
$ JTPU := EDIT/XTPU
```

ただし、参照するファイル名や論理名には若干の変更があります。ご注意ください。

表 2-1 JTPU とXTPU の違い

	JTPU	XTPU
セクションファイルを指定する論理名	JTPU\$SECTION	XTPU\$SECTION
初期化ファイルを指定する論理名	JEVE\$INIT	JEVE\$INIT_V3
コマンドファイルを指定する論理名	JTPU\$COMMAND	XTPU\$COMMAND

2.3 AXP を使っているか VAX を使っているか調べたい

質問:使っているCPU が VAX か Alpha AXP かを判別し、コマンドプロシジャの動きを変えたいのですが?

いくつか方法があります。

OpenVMS VAX V5.5以上、または OpenVMS AXP の全バージョンで、レキシカル関数 F\$GETSYIで新しいアイテム "ARCH_NAME" が使用可能となっています。

```
F$GETSYI("ARCH_NAME")
```

このレキシカル関数は、使用しているCPU が VAX なら "VAX", Alpha AXP なら "Alpha" という文字列を返します。これを利用した簡単なコマンドプロシジャの例を以下に示します。

```
$ CPUNAME = F$GETSYI("ARCH_NAME")
$ IF CPUNAME .EQS. "VAX"
$ THEN
$ ! VAX C, DEC C 混在環境の VAX で VAX C コンパイラを使用
$   CC/VAXC 'P1
$ ELSE
$ ! Alpha AXP で DEC C コンパイラを VAX C 互換モードでコンパイルする
$   CC/STANDARD=VAXC 'P1
$ ENDIF
$ EXIT
```

VAX/VMS V5.5 より前のバージョンでは "ARCH_NAME" アイテムを使用することはできません。代わりに F\$GETSYI のアイテム "CPU" か "HW_MODEL" を用います。それぞれの使い方は以下の通りです。

- F\$GETSYI("CPU")
戻り値が 128 より小さければ OpenVMS VAX システム
128 であれば OpenVMS AXP システム
- F\$GETSYI("HW_MODEL")
戻り値が 1024 より小さければ OpenVMS VAX システム
1024 より大きければ OpenVMS AXP システム

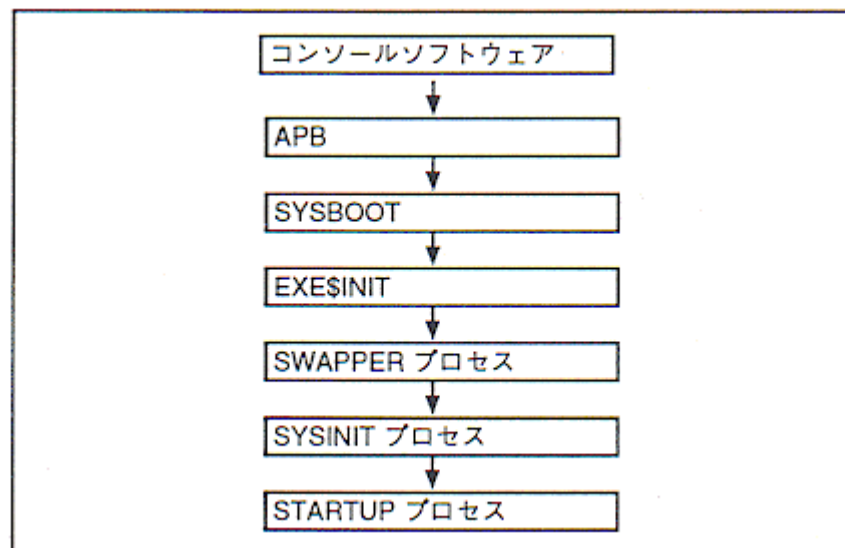
2.4 Alpha AXP のブートストラップ

Alpha AXPシステムの起動は、VAXの場合と同じようにコンソール・コマンド言語 (CCL) を使用して行います。コンソール・コマンドは、Alpha AXP システムでもマシンによって一部違いがあるため、管理するマシンごとに覚える必要があります。Alpha AXP システムの起動時にコンソール・ソフトウェアが行う作業は次の通りです。

1. プロセッサの初期化
2. APB.EXE (OpenVMS AXPプライマリ・ブートストラップ・プログラム) を置くためのメモリの確保
3. APBの初期環境の設定
4. PALコードのロード
5. APBのロード
6. APBに制御を移す

この後に行われるシステム初期化は、VAX/VMSと同じになります。以下に Alpha AXP のブートストラップ図を示します。

図 2-1 Alpha AXP のブートストラップ



2.4.1 ブートコマンドのパラメータについて

システムを起動するには、以下のようなブートコマンドを使用します。

```
>>> BOOT <-FLAG> <FILENAME> ブートデバイス
```

ブートコマンドのパラメータの意味は次のとおりです。

表 2-2 ブートコマンドのパラメータ

パラメータ	指定方法	説明
-FLAG	-FLAG 値	BOOT_OSFLAGS の指定を上書きする。
FILENAME	FILENAME ファイル名	ネットワーク・ブートする時に使用します。
ブートデバイス	デバイス名	ブート装置名を指定します。

DEC3000 M500 のブート例

```
>>> BOOT DKA400
      INIT-S-CPU...
      INIT-S-RESET_TC...
      INIT-S-ASIC...
      INIT-S-MEM...
      INIT-S-NVR...
      INIT-S-CXT...
      INIT-S-SCC...
      INIT-S-NI...
      INIT-S-SCSI...
      INIT-S-ISDN...
      AUDIT_BOOT_STARTS ...
      AUDIT_CHECKKKKSUM_GOOD
      AUDIT_LOAD_BEGINS
      AUDIT_LOAD_DONE
```

OpenVMS AXP (TM) Operating System, Version V1.0

⋮
⋮

2.4.2 典型的なブートコマンド

- 通常のブート


```
>>> B -FLAG 0,0 ブートデバイス名
      デフォルト設定がしてあれば
      >>> B
```
- 会話型ブート


```
>>> B -FLAG 0,1 ブートデバイス名
```
- スタンドアロンバックアップのブート方法 (OpenVMS AXP V1.5-1H1 まで)


```
>>> B -FLAG E,0 ブートデバイス名
```
- ルートディレクトリ[SYSn] からのブート

>>> B -FLAG n,0 ブートデバイス名

2.5 デフォルトブートデバイス、パラメータの設定

質問:毎回 >>> B -FLAG 0,0 ディスク装置名と入力するのは面倒です。デフォルト値を設定し>>> B だけでブートさせたいのですが。

デフォルトのブートデバイスを設定するには、次のコマンドを使用します。

```
>>> SET[ENV] BOOTDEF_DEV 装置名
```

デフォルトのブートデバイスは、次のコマンドで確認できます。

```
>>> SHOW BOOTDEF_DEV
```

デフォルトのブートデバイスの設定を解除する (DEC 3000 シリーズではデフォルトのブートデバイスの設定を変更することはできますが、解除することはできません。) には、次のコマンドを使用します。

```
>>> SET[ENV] BOOTDEF_DEV
```

ブートパラメータは、どのシステム・ルートからブートするのか、どういふブートなのかを指定します。

```
>>> SET[ENV] BOOT_OSFLAGS 値1, 値2
```

値1 は、システム・ルート・ディレクトリの値を 16 進数で指定します。値2 には、次のような値を指定することができます。

表 2-3 BOOT_OSFLAGS に指定する主な値と意味

マスク	意味
1	会話型ブート
2	XDELTA の動作中のシステムにマップ
4	最初のシステムブレークポイントでブートを停止する
8	診断ブートストラップを実行する
10	ブートストラップ・ブレークポイントでブートプロシジャを停止する
20	セカンダリ・ブートストラップイメージからヘッダを省略する
40	メモリテストを行わない
80	セカンダリ・ブートストラップファイルのファイル名を入力できる
100	セカンダリ・ブートストラップの前にシステムを HALT する
2000	リードエラーページを不良としてマークする
10000	ブート中に詳細なデバッグメッセージを表示する
20000	ブート中にユーザに関連したメッセージを選択して表示する

2.6 Ethernet ポートの切り替え

質問:DEC 3000 Model 800 を使っています。これまで Ethernet のポートを 10 Base 5 (Thick Wire) から 10 Base T (Twisted Pair) に変更しようとして、ケーブルを繋ぎ変えました。しかし以前の設定が残っているようで、システムを再起動し、他の DECnet ノードが unreachable になってしまいます。CCL で >>> SHOW DEVICE コマンドを発行しても Ethernet のハードウェア アドレスの後には "Thick" と表示されています。以前使用していた VAXstation には本体背面に切り替えスイッチがありました。DEC3000 Model 800 にはそれらしいスイッチが見当たりません。どうすればいいのでしょうか？

Alpha マシンの場合には、VAXstation のように Thick と Thin のポートの間に切り換えスイッチがあるわけではありません。ハードウェアでの設定はケーブルを付け替えるだけです。ただし、ソフトウェア的な設定が必要になります。コンソールコマンド言語 (CCL) で次のコマンドを入力してください。

- 10 Base T にするのであれば、

```
>>> SET ETHERNET TENBT
```

あるいは

```
>>> SET ETHERNET 1
```

- 10 Base 5 (thick) にするのであれば、

```
>>> SET ETHERNET THICK
```

あるいは

```
>>> SET ETHERNET 0
```

と設定して下さい。

ETHERNET = TENBT あるいは ETHRTNET = THICK というエコーが返ってきます。この後でマシンを立ち上げ、SET HOST コマンドを実行してください。今度はログインできるはずで

2.7 スタンドアロンバックアップキットを作成できない(1)

質問:OpenVMS AXP V1.5-1H1 を使っています。テープメディアにスタンドアロンバックアップのキットを作成しようとして STABACKIT.COM を実行したのですが、"Booting from the output kit device is not supported on this machine." というメッセージが表示されます。

OpenVMS AXP では、テープ及び 20M バイト以下のディスクにスタンドアロンバックアップキットを作成することはできません。以下は OpenVMS AXP V1.0 のリリースノートからの抜粋です。

4.4 Backup Utility - Restrictions

The following restrictions apply to the Backup utility on OpenVMS Alpha AXP:

...

You cannot build a standalone backup kit on the following media types:

- A magnetic tape

A magnetic tape OpenVMS Alpha AXP does not support booting standalone from magnetic tape.

- A disk smaller than 20Mb

This is the minimum disk size needed to build an OpenVMS Alpha AXP standalone backup kit.

2.8 スタンドアロンバックアップキットを作成できない (2)

質問:OpenVMS AXP V6.1 を使っています。ディスクにスタンドアロンバックアップのキットを作成しようとして STABACKIT.COM を実行したのですが、見慣れないメッセージが出力されます。

OpenVMS AXP V6.1 で STABACKIT.COM を実行すると以下のメッセージが表示されます。

```
$ @SYS$UPDTAE:STABACKIT
```

```
Standalone Backup is no longer part of the OpenVMS AXP operating
system. Please refer to the "Backing Up and Restoring the System
Disk" appendix in the "OpenVMS AXP Upgrade and Installation
Manual" for more information.
```

```
After reading the information in the manual, you may wish to use
the procedure SYS$SYSTEM:AXPVMS$PCSI_INSTALL_MIN.COM to install
OpenVMS AXP without any optional features on one or more of your
"data disks".
```

```
Do you wish to invoke SYS$SYSTEM:AXPVMS$PCSI_INSTALL_MIN.COM? [Yes/No]
```

スタンドアロンバックは OpenVMS V6.1 からは、オペレーティングシステムの一部として提供されなくなりました。かわりに SYS\$SYSTEM:AXPVMS\$PCSI_INSTALL_MIN.COM コマンドプロシジャが提供され、ユーザディスクに最小限の OpenVMS AXP をインストールすることができます。スタンドアロンバックアップでは BACKUP コマンドしか使用できませんでしたが、V6.1 からは他の DCL コマンドも使用できるようになり使い勝手が向上しています。システムディスクのバックアップを取りたい場合にはユーザディスクから、この最小限の OpenVMS AXP システムを起動し、イメージバックアップをとることができます。

上記の質問に YES で答えると、さらに以下のメッセージが表示されます。

```
...executing @SYS$SYSTEM:AXPVMS$PCSI_INSTALL_MIN.COM
```

```
This procedure will allow you to install OpenVMS AXP with no options
on a disk other than your system disk. You can then boot from that
disk to backup your system disk.
```

```
You must enter the device name for the target disk. The target disk
must be an OpenVMS system disk. (ここで言う OpenVMS system disk とは、
OpenVMS が認識できる Files-11 ディスクという意味です。)
```

```
Before this procedure is run, the
target disk be mounted privately to this process. This means that it
must not be mounted with /SYSTEM, /CLUSTER, /GROUP or /SHARE qualifiers.
It also must not be mounted with the /FOREIGN qualifiers.
```

```
If any of these qualifiers were used when the disk was mounted,
```

dismount it and re-mount it without them.

(If you need to MOUNT enter Ctrl-Y to exit.)

なお、この最小限のシステムは OpenVMS AXP V6.1 配布メディアの CD-ROM 上に既に作成されています。

CD-ROM からブートすると 以下のメッセージが表示されます。

You can install or upgrade the OpenVMS AXP operating system.
You can also execute DCL commands and procedure to perform
"standalone" tasks, such as backing up the system disk.

Please choose one of the following:

- 1) Install or Upgrade OpenVMS AXP Version V6.1
- 2) Execute DCL commands and procedures
- 3) Shut down this system

Enter CHOICE or ? to repeat menu: (1/2/3/?)

ここで 2 を選択すると \$\$\$ というプロンプトになります。ここでは標準的な OpenVMS のコマンドを実行できます。ここでディスクをマウントし、イメージバックアップをとってください。

第 3 章 DEC Fortran プログラミング編

3.1 引数の数を調べる VAX MACRO のサブルーチン

質問: VAX VMS で作成された Fortran のプログラムを Alpha AXP に移植中です。このプログラムの中には、可変個の引数を受け取るサブルーチンがあります。このサブルーチンは VAX MACRO で書かれた小さなプログラムを呼び出して、メインプログラムからいくつの引数が渡されたかを調べています。ところが Alpha AXP ではこの MACRO プログラムはうまくコンパイルできません。どのようにしたらよいのでしょうか？

MACRO プログラムは以下のようなものです。

```

; SUBROUTINE NOARG(NARG)
; RETURN NUMBER OF ARGUMENTS CALLERS WAS CALLED WITH
; --
      .IDENT   /01/
      .PSECT   $CODE, PIC, CON, REL, LCL, SHR, EXE, RD, NOWRT, LONG
      .ENTRY   NOARG, ^M<>

      MOVL     8(FP), R0
      MOVZBL   (R0), R1
      CMPL     R1, #1
      BNEQ     NOPROB
      TSTL     4(R0)
      BNEQ     NOPROB
      CLRL     R1

NOPROB: MOVL   R1, @4(AP)
      RET

```

Fortran プログラムでサブルーチンに渡す引数を省略するのは標準的な手法ではありません。しかし多くの処理系で動いてしまうのも事実です。

VAX では、多くのプログラマが上記のような VAX MACRO プログラムを使用してサブルーチン呼び出し側の AP レジスタ (Argument Pointer Register, 引数ポインタレジスタ。汎用レジスタ R12 が用いられます。) にアクセスし、引数リストのアドレスや引数の数を調べていました。しかし、この VAX MACRO プログラムは VAX アーキテクチャの標準呼び出し規約に依存しているため、Alpha AXP ではたとえコンパイルできたとしても期待された結果は得られません。上記の MACRO で書かれたサブルーチン NARG から見ると、メインプログラムは caller's caller になります。caller の引数の数は AI レジスタ (R25) に格納されていますが、Alpha AXP のアーキテクチャでは、caller's caller の AI レジスタを参照する共通の方法はありません。Alpha AXP システムでは各言語ごとに用意される手法に従わなければなりません。DEC Fortran では、同等の処理を行なう `iargcount()`、`iargptr()` という組み込み関数を提供しています。以下にこの使用例を示します。

C

C iargptr() の使用例

C

```
program test
```

```
call ByAddress( 10 )
```

```
call ByAddress( 10,20 )
```

```
call ByAddress( 10,20,30 )
```

```
call ByAddress( 1,2,3,4 )
```

```
call ByAddress( 4,3,2,1 )
```

```
call ByVal( %val(1) )
```

```
call ByVal( %val(2), %val(3) )
```

```
call ByVal( %val(4), %val(5), %val(6) )
```

```
end
```

C *****

C 少なくとも OpenVMS AXP V1.5, DEC Fortran V6.1 では

C iargptr() の戻り値が指しているアドレスはこのプログラムでは以下の構造になる。

C 引数が 64 bit align されているのはコールフレームに置かれるため。

C 4 バイト

```
C +-----+
```

```
C | arg_count | iargptr() の戻り値
```

```
C +-----+
```

```
C | 0 |
```

```
C +-----+
```

```
C | &arg1 | 第一引数のアドレス。
```

```
C +-----+ 呼び出しがアドレス渡しになっている。
```

```
C | 0 |
```

```
C +-----+
```

```
C | &arg2 | 第二引数のアドレス。
```

```
C +-----+ 呼び出しがアドレス渡しになっている。
```

```
C | 0 |
```

```
C +-----+
```

```
C .
```

```
C .
```

```
C .
```

```
subroutine ByAddress()
```

```
integer*2 count
```

```
integer*4 ptr
```

```
integer*4 arg_list( 10 ) ! (最大引数の数+1)* 8 バイト用意
```

```
integer*4 args(4)
```

```
integer*4 stat, lib$movc3
```

```
write(6,*) '=== subroutine ByAddress ==='
```

```

count = iargcount()
ptr = iargptr()
write(6,*) 'iargcount() の戻り値',count

```

C 引き数リストをもってくる

```

stat = lib$movc3( 40, %val(ptr), arg_list)

```

C arg_list(1) は引数の数

```

count = arg_list(1)
write(6,*) 'iargptr から求められた引数の数',count

```

C 引数の値を得る

```

do i = 1,count
    stat = lib$movc3( 4, %val(arg_list(i*2+1)), args(i))
end do
write(6,*) '引数',(args(i),i=1,count)
return
end

```

C *****

C 少なくとも OpenVMS AXP V1.5, DEC Fortran V6.1 では

C iargptr() の戻り値が指しているアドレスは以下の構造になっている。

C おなじく 64 bit align になっている。

```

C      4 バイト
C      +-----+
C      | arg_count |   iargptr() の戻り値
C      +-----+
C      |         0 |
C      +-----+
C      |  arg1    |   第一引数。
C      +-----+   呼び出しが値渡しになっている。
C      |         0 |
C      +-----+   第二引数。
C      |  arg2    |   呼び出しが値渡しになっている。
C      +-----+
C      |         0 |
C      +-----+
C      .
C      .
C      .

```

```

subroutine ByValue()
integer*2      count
integer*4      ptr
integer*4      arg_list(48) ! (最大引数の数+1)* 8 バイト用意

```

```

integer*4      args(4)
integer*4      stat, lib$movc3

write(6,*) '=== subroutine ByValue ==='
ptr = iargptr()
count = iargcount()
write(6,*) 'iargcount() の戻り値',count

```

C 引き数リストをもってくる

```
stat = lib$movc3( 40, %val(ptr), arg_list)
```

C arg_list(1) は引数の数

```
count = arg_list(1)
write(6,*) 'iargptr から求められた引数の数',count

```

C 引数の値を得る

```
do i = 1,count
  args(i) = arg_list(i*2+1)
end do
write(6,*) '引数', (args(i), i=1,count)
return
end

```

マシンコードつきでコンパイルリストを作成しコンパイラが作成するコードを見ると `iargcount()` は関数呼び出しではなくオブジェクトコード中で R25 を変数にコピーするようインライン展開されています。subroutine 内で `iargptr()` を使っていると、コンパイラはプロログコードに `JSB OTH$HOME_ARGS` というコードを埋め込みます。この `OTH$HOME_ARGS` はマニュアルには記載されていませんが、コールフレームに引数をホーミング (AXP の標準呼び出し規約では、基本的に第 1 引数から第 6 引数まではレジスタにセーブされ、第 7 引数以降はコールフレームにセーブされます。しかしレジスタにセーブするだけでなくコールフレームに第 1 引数から第 6 引数までを置くこともできます。これを `argument homing` と言います。) しているようです。実際の `ptr = iargptr()` の正体は

```

ADDL FP, xxx, R1
STL R1, PTR

```

です。これで ソースプログラム上の `caller's caller` の引数でも参照できるようになるのです。

3.2 キャラクタ定数をサブルーチンに渡す時の注意

OpenVMS VAX で作成された Fortran のプログラムを Alpha AXP に移植しています。このプログラムの中には、' ' で括られた文字列 をサブルーチンに渡しており サブルーチンではこの引数を整数型で受け取っています。

サブルーチンと、そのサブルーチンを呼び出すプログラムは別ファイルになっています。VAX ではうまく動いていたのですが、Alpha AXP ではうまくデータがサブルーチンに渡って来ません。なぜですか？

ご質問のプログラムは以下のようなものです。

```

CCC
C MAIN.FOR
      call sub001('ABCD')
      end

CCC
C SUB.FOR
      subroutine sub001( I )
      integer*4 I
      write(6,100) I
100    format(1H ,4Z)
      end

$ FORTRAN MAIN, SUB
$ LINK MAIN,SUB
$ RUN MAIN,SUB

```

このプログラムを VAX でコンパイル / リンクし実行すると "44434241" と表示されます。これがお客様の期待する結果です。しかし Alpha AXP でコンパイル / リンクし実行すると "10E0004" になります。なぜでしょうか？

VAX では、引数の渡し方の決定は、コンパイラとリンカーの共同作業で行なわれていました。上記のプログラムの場合、文字列 'ABCD' の渡し方はコンパイラではなくリンカーによって決定され、リファレンス渡しになっていました。しかし Alpha AXP では多くの作業をコンパイラが行いません。Alpha AXP では文字列 'ABCD' はコンパイラによって、ディスクリプタ渡しになります。結果として Alpha AXP で上記のプログラムを実行するとサブルーチン SUB001 の仮引数 I には、実際の文字データ部分ではなくディスクリプタの第 1 ロングワード部分が入ってきます。

VAX の時と同じ結果を得るためには、DEC Fortran V6.1 までは

1. ひとつのファイルにまとめて、コンパイラの最適化にまかせる
2. 呼び出すときに、明示的にリファレンスで渡す。すなわち CALL SUB(%REF('ABCD')) とする。

のどちらかしかありませんでした。

DEC Fortran V6.2 では、コンパイル時の修飾子 /BY_REF_CALL が追加されました。コンパイル時にこの修飾子を使うと、実引数として使われるキャラクタ定数をリファレンス渡しにすることができます。デフォルトでは、キャラクタ定数は全てデスクリプタで渡されます。コンパイル時に以下のように使います。

```
$ FORTRAN /BY_REF_CALL=(サブルーチン名, サブルーチン名...)
```

上記のプログラム例では、

```
$ FORTRAN /BY_REF_CALL=(SUB001) MAIN
```

となります。以下は、DEC Fortran V6.2 リリースノートからの抜粋です。

- /BY_REF_CALL causes character constants used as actual arguments to routines to be passed by reference {%REF} instead of by descriptor {%DESCR}. This may help applications that pass quoted literals to numeric dummy parameters {Hollerith}: this works in DFVV because the VAX/VMS linker fixes up the argument passing mechanism; this does not work without /BY_REF_CALL in DFAV. /BY_REF_CALL modifies all uses of character constants as actual arguments.

3.3 コンパイル時に MISALIGN の警告メッセージ

OpenVMS VAX で作成された Fortran のプログラムを Alpha AXP に移植しています。このプログラムでは COMMON ブロックを使用しているのですが、Alpha AXP の DEC Fortran ではコンパイル時に %FORT-W-MISALIGN, ... というメッセージが表示されます。VAX では、このようなメッセージは出力されませんでした。これはどういう意味ですか？

ご質問のプログラムは以下のようなものです。

```
PROGRAM COMMON_TEST
  integer*2 i2
  integer*4 i4
  character*5 chara

  common/com001/chara, i2, i4

  chara = 'ABCDE'
  i2 = 10
  i4 = 20

  write(6,*) chara, i2, i4

end
```

このプログラムを VAX 上の DEC Fortran コンパイラでコンパイルすると、警告メッセージは表示されません。一方、Alpha AXP 上の DEC Fortran コンパイラでコンパイルすると、以下の警告メッセージが表示されます。

```
integer*2 i2
.....^
%FORT-W-MISALIGN, Alignment of variable or array is inconsistent with its data
type at line number 2 in file DISK$USER:[USER]TEST.FOR;6

integer*4 i4
.....^
%FORT-W-MISALIGN, Alignment of variable or array is inconsistent with its data
type at line number 3 in file DISK$USER:[USER]TEST.FOR;6
```

これは両プラットフォームのアーキテクチャの違いによるものです。VAX は memory-to-memory アーキテクチャの 32 ビット CPU を使用します。VAX の命令セットは可変長であり、メモリアクセスもバイト境界で行なえます。VAX 上の DEC Fortran コンパイラは変数をバイト境界に配置します。一方 Alpha AXP は load-store アーキテクチャの 64 ビット CPU を使用します。命令セットは固定であり、メモリアクセスはロングワードまたはクォードワード境界で行なわれます。もし変数がロングワード境界に置かれていない場合にはトラップが発生し、十分なパフォーマンスを得ること

ができません。そこで Alpha AXP 上の DEC Fortran コンパイラは パフォーマンスを得るために、デフォルトで局所データをその変数の型の自然境界に配置しようとしています。しかし EQUIVALENCE, COMMON, RECORD, STRUCTURE のデータ宣言文に関してはプログラムに書かれた順にメモリ中にデータが詰め込まれます。

このプログラムの コモンブロック COM001 では 5 バイトの文字列、2 バイトの整数、4 バイトの整数を順に格納しています。コンパイル時の警告メッセージは、Alpha AXP が十分な性能を出せない形にデータが詰め込まれていることを示しています。この問題の対処方法として以下の 4 つが考えられます。

- 警告メッセージなので無視する。またはコンパイル時に /NOWARNING 修飾子を指定して警告メッセージそのものを表示させない。
- コンパイル時に、/ALIGNMENT=(COMMON=STANDARD) 修飾子を指定してコンパイルする。これによりコモンブロック内のデータは最大 32 ビット境界に配置される。
- プログラム内に CDEC\$ OPTIONS ディレクティブを追加し適切な境界にデータが置かれるようにする。
- プログラムのコモンブロックを書き換え、データ長の大きい順に並べ変える。

ただし 1 の方法では alignment trap が多発し十分な性能を得られないかもしれません。以下は、DEC Fortran for Alpha AXP V6.2 の HELP の抜粋です。

FORTRAN

```
/ALIGNMENT[=p]                D=/ALIGN=(COMM=(PACK,NOMULTI),RECO=NATU)
```

```
/[NO]ALIGNMENT
```

Controls whether the DEC Fortran compiler naturally aligns fields in records or data items in common blocks, or whether the compiler packs those fields and data items together on arbitrary byte boundaries.

For performance reasons, DEC Fortran always aligns local data items on natural boundaries. However, EQUIVALENCE, COMMON, RECORD, and STRUCTURE data declaration statements can force misaligned data. Use the /ALIGNMENT qualifier to control the alignment of fields associated with COMMON and RECORD statements.

NOTE

Misaligned data significantly increases the time it

takes to execute a program, depending on the number of misaligned fields encountered. Specifying /ALIGNMENT=ALL (same as /ALIGNMENT=NATURAL) minimizes misaligned data. For more information on alignment and data sizes, see your user manual.

To obtain compiler messages when misaligned data is encountered, use the /WARNING=ALIGNMENT qualifier (or within the VMS Debugger, use the command SET BREAK/UNALIGNED).

Parameter "p" is a specifier with one of the following forms:

```
[class =] rule  
(class = rule,...)  
ALL  
NONE
```

class

Is one of the following keywords:

```
COMMONS      (for common blocks)  
RECORDS      (for records)  
STRUCTURES   (in Fortran, a synonym for RECORDS)
```

rule

Is one of the following keywords:

PACKED

Packs fields in records or data items in common blocks on arbitrary byte boundaries.

NATURAL

Naturally aligns fields in records and data items in common blocks on up to 64-bit boundaries (inconsistent with the FORTRAN-77 standard).

If you specify NATURAL, the compiler will naturally align all data in a common block, including INTEGER*8, REAL*8, and all COMPLEX data.

STANDARD

Naturally aligns data items in common blocks on up to 32-bit boundaries (consistent with the FORTRAN-77 standard).

The compiler will not naturally align INTEGER*8, REAL*8, and COMPLEX data declarations. Such data declarations should be planned so they fall on natural boundaries. Specifying /ALIGNMENT=COMMONS=STANDARD alone means that the default for record structures is used.

Note that this keyword only applies to common blocks; therefore, you can specify /ALIGNMENT=COMMONS=STANDARD, but you cannot specify /ALIGNMENT=STANDARD.

[NO]MULTILANGUAGE

Controls whether the compiler pads the size of overlaid psects so as to ensure compatibility when the psect is shared by code created by other DEC compilers.

When a psect generated by a Fortran common block is overlaid with a psect consisting of a C structure, linker error messages can result. This is because the sizes of the psects are inconsistent; the C structure is padded and the Fortran common block is not.

Specifying /ALIGNMENT=COMMON=MULTILANGUAGE ensures that DEC Fortran follows a consistent psect size allocation scheme that works with DEC C psects that are shared across multiple images. Psects shared in a single image do not have a problem. The corollary DEC C qualifier is /PSECTMODEL=[NO]MULTILANGUAGE.

The default is /ALIGNMENT=COMMON=NOMULTILANGUAGE which is the behavior of previous releases of DEC Fortran and is sufficient for most applications.

Note that this keyword only applies to common blocks; therefore, you can specify /ALIGNMENT=COMMONS=[NO]MULTILANGUAGE, but you cannot specify /ALIGNMENT=[NO]MULTILANGUAGE.

ALL

Is equivalent to /ALIGNMENT, /ALIGNMENT=NATURAL, and /ALIGNMENT=(COMMONS=(NATURAL,NOMULTILANGUAGE),RECORDS=NATURAL).

NONE

Is equivalent to /NOALIGNMENT, /ALIGNMENT=PACKED, and /ALIGNMENT=(COMMONS=(PACKED,NOMULTILANGUAGE),RECORDS=PACKED).

If you do not specify /ALIGNMENT, the default is /ALIGNMENT=(COMMONS=(PACKED,NOMULTILANGUAGE),RECORDS=NATURAL).

If you specify `/ALIGNMENT=class=rule`, the rule only applies to that class, the other class gets aligned according to the default; for example:

1. `/ALIGNMENT=COMMONS=STANDARD`

In this case, `RECORDS=NATURAL` by default.

2. `/ALIGNMENT=RECORDS=NATURAL`

In this case, `COMMONS=PACKED` by default.

To request packed, unaligned data in a record structure, specify `/ALIGNMENT=RECORDS=PACKED`, or consider placing source data declarations for the record so that the data is naturally aligned.

To request aligned data in common blocks, specify `/ALIGNMENT=COMMONS=STANDARD` (for data items up to 32 bits in length) or `/ALIGNMENT=COMMONS=NATURAL` (for data items up to 64 bits in length), or place source data declarations within the common block carefully so that each data field is naturally aligned.

The `/ALIGNMENT` and `/WARN=ALIGNMENT` qualifiers can be used together in the same command line.

You can override the alignment specified on the command line by using a `CDEC$ OPTIONS` directive, as described in your language reference manual.

3.4 共有コモンブロックの利用

OpenVMS VAX で作成された Fortran のプログラムを Alpha AXP に移植しています。このプログラムでは共有コモンブロックを利用して、複数のプログラムからデータの参照 / 更新を行なうものです。VAX では問題なく動いていたのですが、Alpha AXP では期待どおりに動きません。どうしたらいいのでしょうか？

以下に OpenVMS VAX での共有コモンブロックを用いたサンプルプログラム及びコンパイル / リンクコマンドの例を示します。

```

$ SET VERIFY
$ FORTRAN/OBJECT=DATA.OBJ SYSS$INPUT
CCC DATA.FOR
CCC 共有コモンブロック宣言
      COMMON /GLOBAL/ IARRAY(4,50)
      END
$ FORTRAN/OBJECT=WRITE.OBJ SYSS$INPUT
CCC WRITE.FOR
CCC 共有コモンブロックにデータを書き込む
      IMPLICIT          INTEGER*4 (A-Z)
      COMMON /GLOBAL/          IARRAY(4,50)
      VOLATILE IARRAY

      DO I = 1,4
        DO J = 1,50
          IARRAY(I,J) = J+(I-1)*50
        END DO
      END DO
      WRITE (6,*) ' WRITES DATA TO SHARABLE COMMON . '

      END
$ FORTRAN/OBJECT=READ.OBJ SYSS$INPUT
CCC READ.FOR
CCC 共有コモンブロックのデータを参照する。
      IMPLICIT          INTEGER*4 (A-Z)
      COMMON /GLOBAL/          IARRAY(4,50)
      VOLATILE IARRAY

      WRITE (6,*) ' CONTENTS OF GLOBAL COMMON '
      WRITE (6,200) (( IARRAY(I,J), I=1,4), J=1,10)
200   FORMAT ( X, 318)

      END
$ LINK/SHAREABLE DATA

```

```

$ INSTALL ADD DISK$USER:[USER.FORTRAN]DATA.EXE/SHARE/WRITEABLE
$ DEFINE DATA DISK$USER:[USER.FORTRAN]DATA.EXE
$ LINK WRITE, SYS$INPUT/OPTION
DATA/SHAREABLE
$ LINK READ, SYS$INPUT/OPTION
DATA/SHAREABLE
$ RUN WRITE
$ RUN READ
$ INSTALL REMOVE DATA
$ DEASSIGN DATA

```

OpenVMS AXP で共有コモンブロックを使用したプログラムを作成する際には、リンク時に VAX とは異なる指定が必要です。

1. VAX では OVR/REL/GBL 属性を持つ全てのプログラムセクション (各プログラムセクションの属性は、\$ FORTRAN /LIST /MACHINE_CODE コマンドでマシンコード付きのコンパイルリストを作成することにより調べることができます。) は自動的に共有イメージのグローバルシンボルベクトルにエクスポートされていました。OpenVMS AXP では共有イメージを作成する際に SYMBOL_VECTOR = (<コモンブロック名> = PSECT) 指定をオプションファイル内で行い、共有コモンブロック名を他のモジュールから参照できるように指示しなければなりません。これにはいくつかの利点があります。
 - a. 新規に、上位互換の共有イメージを作成する場合、プログラムセクションのアドレスを気にせずにすむ。
 - b. 共有イメージ内のどのプログラムセクションを外部から参照可能にするかを細かく制御できる。
2. DEC Fortran for Alpha AXP では、コモンブロックのプログラムセクション属性がデフォルトで NOSHR になっているため、イメージを作成する際に SHR 属性に変更するか、CDEC\$ PSECT デイレクティブをプログラムに追加し、コモンブロックの属性を変えなければなりません。(CDEC\$ は 1 カラム目から書かれていなければなりません。)

詳細はリンカーのマニュアルを参照してください。

3.4.1 リンクコマンドを変更する場合

以上を考慮してプログラムをリンクする部分を以下のように変更してください。

```

$ LINK/SHAREABLE DATA, SYS$INPUT/OPTION ! 変更
SYMBOL_VECTOR=(GLOBAL=PSECT)           ! 追加
PSECT=GLOBAL, SHR                        ! 追加
$ INSTALL ADD DISK$USER:[USER.FORTRAN]DATA.EXE/SHARE/WRITEABLE
$ DEFINE DATA DISK$USER:[USER.FORTRAN]DATA.EXE
$ LINK WRITE, SYS$INPUT/OPTION
DATA/SHAREABLE
PSECT=GLOBAL, SHR                        ! 追加

```

```

$ LINK READ, SYS$INPUT/OPTION
DATA/SHAREABLE
PSECT=GLOBAL,SHR                ! 追加
$ RUN WRITE
$ RUN READ
$ INSTALL REMOVE DATA
$ DEASSIGN DATA

```

3.4.2 プログラムにディレクティブを追加する場合

Fortran プログラムに CDEC\$ PSECT ディレクティブを使用した場合には以下のよう to してください。

```

$ SET VERIFY
$ FORTRAN/OBJECT=DATA.OBJ SYS$INPUT
CCC DATA.FOR
CCC 共有コモンブロック宣言
CDEC$ PSECT /GLOBAL/ GBL,SHR,WRT    ! 追加
      COMMON /GLOBAL/ IARRAY(4,50)
      END
$ FORTRAN/OBJECT=WRITE.OBJ SYS$INPUT
CCC WRITE.FOR
CCC 共有コモンブロックにデータを書き込む
      IMPLICIT      INTEGER*4 (A-Z)
CDEC$ PSECT /GLOBAL/ GBL,SHR,WRT    ! 追加
      COMMON /GLOBAL/      IARRAY(4,50)
      VOLATILE IARRAY

      DO I = 1,4
        DO J = 1,50
          IARRAY(I,J) = J+(I-1)*50
        END DO
      END DO
      WRITE (6,*) ' WRITES DATA TO SHARABLE COMMON . '

      END
$ FORTRAN/OBJECT=READ.OBJ SYS$INPUT
CCC READ.FOR
CCC 共有コモンブロックのデータを参照する。
      IMPLICIT      INTEGER*4 (A-Z)
CDEC$ PSECT /GLOBAL/ GBL,SHR,WRT    ! 追加
      COMMON /GLOBAL/      IARRAY(4,50)
      VOLATILE IARRAY

      WRITE (6,*) 'CONTENTS OF GLOBAL COMMON '
      WRITE (6,200) (( IARRAY(I,J), I=1,4), J=1,50)

```

200 FORMAT (X, 318)

 END

\$ LINK/SHAREABLE DATA,SYSS\$INPUT/OPTION

SYMBOL_VECTOR=(GLOBAL=PSECT) ! 追加

\$ INSTALL ADD DISK\$USER:[USER.FORTRAN]DATA.EXE/SHARE/WRITABLE

\$ DEFINE DATA DISK\$USER:[USER.FORTRAN]DATA.EXE

\$ LINK WRITE, SYSS\$INPUT/OPTION

DATA/SHAREABLE

\$ LINK READ, SYSS\$INPUT/OPTION

DATA/SHAREABLE

\$ RUN WRITE

\$ RUN READ

\$ INSTALL REMOVE DATA

\$ DEASSIGN DATA

第 4 章 DEC C プログラミング編

4.1 VAX C で書かれたプログラムのコンパイル時にエラー

VAX 上の VAX C で書かれたプログラムを Alpha AXP に移植しようとしています。VAX ではコンパイルできたプログラムなのですが、Alpha AXP 上の DEC C でコンパイルしようとするとエラーがたくさんでます。なぜですか？

DEC C は American National Standard for Information (ANSI) System-Programming Language C (ドキュメント番号 X.3159-1989)、通称 ANSI C Standard に準拠しています。一方、VAX C はこの規約に準拠していません。一般に、DEC C は VAX C に比べて、文法が厳しくなっています。今後のプログラムの移植性を考えるとDEC C の文法に従った形に書き直したほうがよいでしょう

しかし時間の制約等から書き直しが困難な場合には、/STANDRAD=VAXC 修飾子を使用してコンパイルしてみてください。エラーが減るはずです。この修飾子は、DEC C コンパイラに VAX C の文法でコンパイルするよう指示します。

DEC C のデフォルトは /NOSTANDARD になっています。これは /STANDARD=RELAXED_ANSI89 と同じ意味になります。以下は DEC C V4.0 の HELP からの抜粋です。

CC

```
/STANDARD[=(option[,...])]
```

```
    /NOSTANDARD (D)
```

Defines the compilation mode. You can select the following options:

ANSI89	Places the compiler in strict ANSI C Standard mode. This mode compiles the C language as defined by the American National Standard for C, along with any extensions not prohibited by that standard.
--------	--

RELAXED_ANSI89	Places the compiler in relaxed ANSI C Standard mode. the compiler follows the ANSI C standard, but also accepts additional Digital keywords and predefined macros that do not begin with an underscore. This is the default mode of the compiler, and is equivalent to /NOSTANDARD.
----------------	---

COMMON	Places the compiler in common C mode; that is, compatibility with the pcc compiler on ULTRIX
--------	--

systems. This mode is close to a subset of /STANDARD=VAXC mode.

VAXC Places the compiler in VAX C mode. There are differences in the C language as implemented in previous versions of VAX C and the C language as defined by ANSI (the differences are primarily concerned with how the preprocessor works). This mode provides compatibility for programs that depend on old VAX C behavior.

PORTABLE Places the compiler in VAX C mode, and enables the issuance of diagnostics that warn of any nonportable usages encountered.

The default is /NOSTANDARD, which is equivalent to /STANDARD=RELAXED_ANSI89.

You can use the COMMON option in combination with either the VAXC option or the PORTABLE option. No other combinations are allowed.

DEC C modules compiled in different modes can be linked and executed together.

4.2 DEC C のリンクコマンド

VAX 上の VAX C で記述されたプログラムをリンクする時には、論理名 LNK\$LIBRARY を定義したり オプションファイルを作成したりしなければなりません。DEC C でのリンク方法を教えてください。

リンカーは SYS\$SHARE:IMAGELIB.OLB, SYS\$SHARE:STARLET.OLB に登録されているライブラリファイルを検索して未定義シンボルを解決しようとします。VAX C のライブラリファイル VAXCRTL.EXE, VAXCRTLG.EXE は様々な理由から IMAGELIB.OLB に登録されていませんでした。

このため VAX C でコンパイルされたオブジェクトモジュールをリンクするには SYS\$SHARE:VAXCRTL.EXE または SYS\$LIBRARY:VAXCRTL.OLB を明示的に指定してリンクしなければなりません。

DEC C では特にオプションファイルを作成しなくてもリンカーが C のライブラリファイルである DECC\$SHR.EXE を検索するようになりました。このため、単に \$ LINK コマンドだけでリンクすることができます。

ただしレイヤードソフトウェアのライブラリを使用する場合には、まだオプションファイルを作成しなければならないものもあります。例えば、DECwindows Motif V1.2 のライブラリを使用したプログラムをリンクするためには、以下のようにオプションファイルを指定してリンクする必要があります。

```
$ LINK xxxx, SYS$INPUT/OPTION
  SYS$SHARE:DECW$XLIBSHR/SHAREABLE
  SYS$SHARE:DECW$XTLIBSHRR5/SHAREABLE
  SYS$SHARE:DECW$XMLIBSHR12/SHAREABLE
  SYS$SHARE:DECW$DXMLIBSHR12/SHAREABLE
  SYS$SHARE:DECW$MRMLIBSHR12/SHAREABLE
```

^Z

4.3 SYS\$LIBRARY にヘッダファイルが無い

OpenVMS AXP の DEC C の環境では、stdio.h 等のヘッダファイルはどこに置かれているのでしょうか? \$ DIRECTORY コマンドで SYS\$LIBRARY を探してみたのですが見つかりません。

VAX C ではインストレーション時の指定で SYS\$LIBRARY にヘッダファイルを展開することができました。DEC C のインストレーション時にはこのような質問はありません。ご質問の stdio.h はテキストライブラリSYS\$SHARE:DECC\$RTLDEF.TLB に収められています。

DEC C コンパイラはソースコード中に #include を見つけると 以下の順に検索を行いません。

1. SYS\$LIBRARY:XXX.H
2. DECC\$TEXT_LIBRARY (通常は未定義です。) 論理名が定義されていれば、そのテキストライブラリ
3. ALPHA\$LIBRARY:DECC\$RTLDEF.TLB, ALPHA\$LIBRARY:DECC\$STARTLET_C.TLB (ALPHA\$LIBRARY は SYS\$LIBRARY を指しています。)

以下のコマンドで、テキストライブラリからヘッダファイルを取り出すことも可能です。

```
$ LIBRARY/EXTRACT=STDIO ALPHA$LIBRARY:DECC$RTLDEF.TLB -
  /OUTPUT=SYS$COMMON:[SYSLIB]STDIO.H
```

4.4 union のメンバがずれる

以下のプログラムを Alpha AXP 上の DEC C でコンパイルし 実行するとtableok とtable で領域が 4 バイトずれてしまいます。VAX 上の VAX C では期待どおりtableok とtable は同じ領域になります。なぜでしょうか？

プログラムは以下のものです。

```
#include <stdio.h>
struct p_def{
    double pp;
};

struct p2_def{
    float pp;
};

struct b_def{
    int b1;
    char *b2;
    int b3;
};

struct table{
    int a1, a2, a3, a4, a5, a6, a7, a8, a9;
    union{
        struct p_def p;
        struct b_def a;
    }u;
};

struct tableok{
    int a1, a2, a3, a4, a5, a6, a7, a8, a9;
    union{
        struct p2_def p;
        struct b_def a;
    }u;
};

#define NIL (0)
#define eql 55

struct a_t_def{
    int a1, a2, a3, a4, a5, a6, a7, a8, a9;
```

```

    int  b1;
    char *b2;
    int  b3;
};

struct  a_t_def at0[]={
  { &at0[ 1], eql, "$wwwwww", 0, NIL,NIL,NIL,NIL,NIL, NIL, "$xxxxxxxx", 0 },
  { &at0[ 2], eql, "$wwwwww", 0, NIL,NIL,NIL,NIL,NIL, NIL, "$xxxxxxxxxxxxxxxx"},

  { &at0[ 3], eql, "$wwwwww", 0, NIL,NIL,NIL,NIL,NIL, NIL, "$xxxxxxx", 0 }
};

main()
{
    struct table      *st;
    struct tableok    *stok;

    stok=st=at0;
    printf("%d¥n", st->u.a.b1 );
    printf("%d¥n", st->u.a.b2 );
    printf("%d¥n", st->u.a.b3 );

    printf("%d¥n", stok->u.a.b1 );
    printf("%d¥n", stok->u.a.b2 );
    printf("%d¥n", stok->u.a.b3 );
}

```

これは VAX と Alpha AXP 両プラットフォームのアーキテクチャの違いによるものです。Alpha AXP は load-store アーキテクチャの 64 ビット CPU です。命令セットは固定であり、メモリアクセスはロングワードまたはクォードワード境界で行なわれます。もし変数がロングワード境界に置かれていない場合にはトラップが発生し、十分なパフォーマンスを得ることができません。そこで DEC C コンパイラはパフォーマンスを得るために、変数を、その変数の型の自然境界に配置しようとしています。一方 VAX は memory-to-memory の 32 ビット CPU です。VAX の命令セットは可変長であり、メモリアクセスもバイト境界で行なえます。VAX C は変数をバイト境界に配置します。

このプログラムの table は int 型データを 9 つ (a1 から a9) の後に double 型データを置くようになっています。この場合、double 型データは クォードワード(64 ビット)境界に配置されるため 4 バイトずれてしまいます。

一方 tableok は int 型データ 9 つ (a1 から a9) の後に float 型データを置いています。float 型データはロングワード(32 ビット)境界に置かれるため、結果として連続して配置されます。sizeof 演算子を使用して、構造体の大きさを調べてみるのもよいでしょう。

このため tableok と table へのポインタを stok=st=at0 と同じ値で初期化してもメンバの値は異なってしまいます。この問題の対処方法として以下の 3 つが考えられます。

1. コンパイル時に /NOMEMBER_ALIGNMENT の修飾子を付ける
2. #pragma nomember_alignment をソースコード中に書き込む
3. データ型に注意して pad を入れるようにソースコードを直す

ただし 1, 2, の方法では alignment trap が多発し、十分なパフォーマンスを得ることは難しくなります。Fortran の COMMON や C の struct などは、型の大きいものから順に並べるように心掛けると安全です。

以下は DEC C V4.0 の HELP からの抜粋です。

CC

/MEMBER_ALIGNMENT (D)

/NOMEMBER_ALIGNMENT

Directs the compiler to naturally align data structure members. This means that data structure members are aligned on the next boundary appropriate to the type of the member, rather than on the next byte. For instance, a long variable member is aligned on the next longword boundary; a short variable member is aligned on the next word boundary.

Any use of the #pragma member_alignment or #pragma nomember_alignment directives within the source code overrides the setting established by this qualifier. Specifying /NOMEMBER_ALIGNMENT causes data structure members to be byte-aligned (with the exception of bit-field members).

For OpenVMS AXP systems, the default is /MEMBER_ALIGNMENT.

For OpenVMS VAX systems, the default is /NOMEMBER_ALIGNMENT.

4.5 DEC C から共有コモンブロックへのアクセス

Fortran で作成した共有コモンブロックを C で書かれたプログラムからも参照更新するようなプログラムが VAX で稼動していました。このプログラムを OpenVMS AXP 上でコンパイル / リンクしなおしたのですが、リンク時に UNDEFINED SYMBOL の警告メッセージが出力されます。どうしたらいいのでしょうか？

プログラムは以下のようなものです。

```
CC 共有コモンブロック作成プログラム COMMON.FOR
COMMON /GLOBAL/ IARRAY(4,50)
END
```

```
CC 共有コモンブロックにデータを書き込むプログラム WRITE.FOR
IMPLICIT INTEGER*4 (A-Z)
COMMON /GLOBAL/ IARRAY(4,50)
VOLATILE IARRAY
DO I = 1,4
  DO J = 1,50
    IARRAY(I,J) = J+(I-1)*50
  END DO
END DO

WRITE (6,*) ' WRITES DATA TO SHARABLE COMMON . '

END
```

```
/*
 * 共有コモンからデータを読むプログラム READ.C
 */
#include <stdio.h>
main()
{
  extern int global[50][4];
  int i,j;

  for(j=0;j<4;j++){
    for(i=0;i<50;i++){
      printf("%d\n", global[i][j]);
    }
  }
}

$!
```

```

$!コンパイルリンク用プロシジャ
$!
$ FORTRAN COMMON, WRITE
$ CC READ
$ LINK/SHAREABLE COMMON,SYSS$INPUT/OPTION
SYMBOL_VECTOR=(GLOBAL=PSECT)

PSECT=GLOBAL, SHR
$ INSTALL ADD ddcu:[directory]COMMON.EXE/SHARED/WRITABLE
$ DEFINE COMMON ddcu:[directory]COMMON.EXE
$ LINK WRITE,SYSS$INPUT/OPTION
COMMON/SHAREABLE
PSECT=GLOBAL, SHR
$ LINK READ,SYSS$INPUT/OPTION
COMMON/SHAREABLE
$ EXIT

```

DEC C と VAX C では、extern で宣言された変数の解釈方法が異なります。VAX C と同じように解釈させるには、コンパイル時に /EXTERN_MODEL=COMMON_BLOCK 修飾子を使用するか、ソースプログラム中に #pragma extern_model common_block shr を追加してください。

```

/*
 * 修正された共有コモンからデータを読むプログラム READ.C
 */
#include <stdio.h>
#pragma extern_model common_block shr /* 追加 */

main()
{
    extern int global[50][4];
    int i,j;

    for(j=0;j<4;j++){
        for(i=0;i<50;i++){
            printf("%d¥n", global[i][j]);
        }
    }
}

```

以下は DEC C V4.0 の HELP からの抜粋です。

CC

```
/EXTERN_MODEL=option (D=RELAXED_REFDEF)
```

In conjunction with the /SHARE_GLOBALS qualifier, controls the initial extern model of the compiler. Conceptually, the compiler behaves as if the first line of the program being compiled was a #pragma extern_model directive with the model and psect name, if any, specified by the /EXTERN_MODEL qualifier and with the shr or noshr keyword specified by the /SHARE_GLOBALS qualifier.

For example, assume the command line contains the following qualifier:

```
/EXTERN_MODEL=STRICT_REFDEF="MYDATA"/NOSHARE
```

The compiler will act as if the program began with the following line:

```
#pragma extern_model strict_refdef "MYDATA" noshr
```

See also #pragma extern_model.

The /EXTERN_MODEL qualifier takes the following options, which have the same meaning as for the #pragma extern_model directive:

COMMON_BLOCK

RELAXED_REFDEF

STRICT_REFDEF

STRICT_REFDEF="NAME"

GLOBALVALUE

The default model on DEC C is relaxed/refdef with the noshare attribute. This is different from the model used by VAX C, which is common block, share.

第 5 章 OpenVMS プログラミング編

5.1 SYS\$CRMPSC() を使用する際の注意点

グローバルセクションを使ったプロセス間通信のプログラムを VAX から Alpha AXP に移植しようとしています。VAX では問題なく動いていたのですが、Alpha AXP では INVARG エラーになってしまいます。どこを直したらいいのでしょうか？

VAX では SYS\$CRMPSC() サービスルーチンがセクションをページ境界に配置していました。Alpha AXP では `inadr` 引数は、必ずページ境界を指定しなければなりません。また Alpha AXP は 1 ページの大きさがシステムごとに可変であるため、セクションを作成する前に、そのシステムの 1 ページの大きさを入手しておく必要があります。このため Alpha AXP では、SYS\$GETSYI(), SYS\$GETSYIW() システムサービスに新たなアイテムコード SYI\$_PAGE_SIZE が追加されています。以下にサンプルプログラムを掲載します。

```
/*
 * SYS$CRMPSC() の使用例
 *
 * GLOBAL1.C
 * このプログラムはグローバルページフレームセクションを作成、マップします。
 * セクション中のデータは配列 iarray を通して行なわれます。
 * 作成されるセクションは GLOBAL2.EXE と共有されます。
 *
 * このプログラムは
 * OpenVMS VAX V5.5-2  VAX C V3.2
 * OpenVMS AXP V1.0   DEC C V1.3
 * で動作確認を行ないました。
 */
```

```
#include <stdio.h>
#include <descrip.h>
#include <secdef.h>
#define ARRAY_COUNT 5000
#define PAGELET 512

#ifdef __ALPHA
# include <syidef.h>
  typedef struct{
    unsigned short length;
    unsigned short code;
    void          *buffer;
    void          *return_length;
  }ItemList;
#endif

/*
```

* セクションはページ境界に置かれなければならない。

*/

```
#pragma nostandard
    _align(page) int iarray[ARRAY_COUNT];
#pragma standard

main()
{
    $DESCRIPTOR(name,"GSEC");
    $DESCRIPTOR(cluster,"MYCLUS");
    int i,flags,status,pagelet_count,page_size;
    int inaddr[2],sysaddr[2];
    int SYS$ASCEFC(),SYS$CRMPSC(),SYS$SETEF(),SYS$WAITFR(),LIB$STOP();

#ifdef __ALPHA
    int SYS$GETSYIW();
    int pages_of_array;
    ItemList syilist[2];
#endif

    /*
     * GLOBAL2.EXE との同期に用いるイベントフラグ・クラスタの確保
     */
    status = SYS$ASCEFC(64,&cluster,0,0);
    if(!(status & 1)) LIB$STOP(status);

    /*
     * OpenVMS AXP では $CRMPSC の inaddr 引数はページ境界でなければならない。
     * また Alpha AXP では CPU によってページサイズが異なる。
     * したがって SYS$GETSYIW() でページサイズを調べなければならない。
     */

#ifdef __ALPHA
    syilist[0].length = sizeof( int );
    syilist[0].code   = SYI$_PAGE_SIZE;
    syilist[0].buffer = &page_size;
    syilist[0].return_length = NULL;
    syilist[1].length = 0;
    syilist[1].code   = 0;

    status = SYS$GETSYIW(0,0,0,syilist,0,0,0);
    if(!(status & 1)) LIB$STOP(status);
#else
    page_size = 512;
#endif
}
```

```

#endif

/*
 * グローバルセクション( テンポラリ ) の作成とマップ
 */

/*
 * SYS$CRMPSC() の pagecnt 引数の単位は pagelet (512 bytes)
 * マップするページレット数を計算する。
 * OpenVMS VAX では page size = pagelet size
 * OpenVMS AXP では page size = pagelet size * N
 */
if( (sizeof(iarray) % PAGELET) != 0 ){
    pagelet_count = sizeof(iarray) / PAGELET + 1;
}else{
    pagelet_count = sizeof(iarray) / PAGELET;
}

#ifdef __ALPHA
/*
 * OpenVMS AXP では $CRMPSC の inaddr 引数はページ境界でなければならない。
 */
if( (sizeof( iarray ) % page_size) != 0 ){
    pages_of_array = sizeof( iarray ) / page_size + 1;
}else{
    pages_of_array = sizeof( iarray ) / page_size;
}
inaddr[0] = (int)&iarray[0];
inaddr[1] = (int)&iarray[0] + pages_of_array * page_size - 1;
#else
inaddr[0] = (int)&iarray[0];
inaddr[1] = (int)&iarray[ARRAY_COUNT-1];
#endif

flags = SEC$m_PAGFIL|SEC$m_GBL|SEC$m_WRT|SEC$m_DZRO;
status = SYS$CRMPSC(inaddr,sysaddr,0,flags,&name,0,0,0,pagelet_count,0,0,0);
if(!(status & 1)) LIB$STOP(status);

printf("¥nGLOBAL1: iarray[] address %x %x¥n",
    (int)&iarray[0], (int)&iarray[ARRAY_COUNT-1]);
printf("GLOBAL1: Input address %x %x ¥n", inaddr[0], inaddr[1] );
printf("GLOBAL1: Return address %x %x ¥n", sysaddr[0], sysaddr[1] );

/*
 * セクション中のデータを操作する

```

```

*/
printf("GLOBAL1: 下記のデータをセットしました。¥n");
for(i=0; i<ARRAY_COUNT; i++){
    iarray[i] = i+1;
    printf("%5d",iarray[i]);
    if((i+1) % 16 == 0) printf("¥n");
}

/*
 * GLOBAL2 にデータ操作が終了したことを通知する。
 * 通知はイベントフラグのセットによって行なっている。
 */
status = SYS$SETEF(72);
if(!(status & 1)) LIB$STOP(status);

/*
 * GLOBAL2 がデータ操作を終了するまで待つ。
 * 通知はイベントフラグのセットによって行なわれる。
 */
printf("¥nGLOBAL1: 待機中...¥n");
status = SYS$WAITFR(73);
if(!(status & 1)) LIB$STOP(status);

printf("¥nGLOBAL1: データ再表示。値は変更されているはず。¥n");
for(i=0; i<ARRAY_COUNT; i++){
    printf("%5d",iarray[i]);
    if((i+1) % 16 == 0) printf("¥n");
}
}

/*
 * GLOBAL2.C
 * このプログラムは GLOBAL1.EXE が作成したセクションをマップし
 * セクション中のデータを変更します。
 * 同期はイベントフラグを使用。
 *
 * このプログラムは
 * OpenVMS VAX V5.5-2 VAX C V3.2
 * OpenVMS AXP V1.0 DEC C V1.3
 * で動作確認を行ないました。
 */

#include <stdio.h>
#include <descrip.h>
#include <secdef.h>
#define ARRAY_COUNT 5000

```

```

#define PAGELET      512

#ifdef __ALPHA
#  include <syidef.h>
  typedef struct{
    unsigned short length;
    unsigned short code;
    void          *buffer;
    void          *return_length;
  }ItemList;
#endif

/*
 * セクションはページ境界に置かれなければならない。
 */
#pragma nostandard
  _align(page) int iarray[ARRAY_COUNT];
#pragma standard

main()
{
  int inaddr[2],sysaddr[2],flags,i,status,page_size;
  int SYS$ASCEFC(),SYS$WAITFR(),SYS$MGBLSC(),SYS$SETEF(),LIB$STOP();

#ifdef __ALPHA
  int SYS$GETSYIW();
  int pages_of_array;
  ItemList syilist[2];
#endif

  $DESCRIPTOR(name,"GSEC");
  $DESCRIPTOR(cluster,"MYCLUS");

  /*
   * GLOBAL1.EXE との同期に用いるイベントフラグ・クラスタの確保
   */
  status = SYS$ASCEFC(64,&cluster,0,0);
  if(!(status & 1)) LIB$STOP(status);

  /*
   * GLOBAL1.EXE がフラグをセットするまで待機する。
   */
  status = SYS$WAITFR(72);
  if(!(status & 1)) LIB$STOP(status);

```

```

/*
 * OpenVMS AXP では $CRMPSC の inaddr 引数はページ境界でなければならない。
 * また Alpha AXP では CPU によってページサイズが異なる。
 * したがって SYS$GETSYIW() でページサイズを調べなければならない。
 */

#ifdef __ALPHA
    syilist[0].length = sizeof( int );
    syilist[0].code   = SYI$_PAGE_SIZE;
    syilist[0].buffer = &page_size;
    syilist[0].return_length = NULL;
    syilist[1].length = 0;
    syilist[1].code   = 0;

    status = SYS$GETSYIW(0,0,0,syilist,0,0,0);
    if(!(status & 1)) LIB$STOP(status);
#else
    page_size = 512;
#endif

/*
 * グローバルセクションのマップ
 */

#ifdef __ALPHA
/*
 * OpenVMS AXP では $CRMPSC の inaddr 引数はページ境界でなければならない。
 */
    if( (sizeof( iarray ) % page_size) != 0 ){
        pages_of_array = sizeof( iarray ) / page_size + 1;
    }else{
        pages_of_array = sizeof( iarray ) / page_size;
    }
    inaddr[0] = (int)&iarray[0];
    inaddr[1] = (int)&iarray[0] + pages_of_array * page_size - 1;
#else
    inaddr[0] = (int)&iarray[0];
    inaddr[1] = (int)&iarray[ARRAY_COUNT-1];
#endif

    flags = SEC$_WRT;
    status = SYS$_MGBLSC(inaddr,sysaddr,0,flags,&name,0,0);
    if(!(status & 1)) LIB$STOP(status);

    printf("¥nGLOBAL2: iarray[] address %x %x¥n",
        (int)&iarray[0], (int)&iarray[ARRAY_COUNT-1]);

```

```
printf("GLOBAL2: Input address   %x %x %n", inaddr[0], inaddr[1] );  
printf("GLOBAL2: Return address  %x %x %n", sysaddr[0], sysaddr[1] );
```

```
/*
```

```
 * セクション中のデータを変更
```

```
 */
```

```
for(i=0; i<ARRAY_COUNT; i++){
```

```
    iarray[i] = iarray[i]*2;
```

```
}
```

```
/*
```

```
 * データ変更の終了を GLOBAL1 に通知
```

```
 */
```

```
status = SYS$SETEF(73);
```

```
if(!(status & 1)) LIB$STOP(status);
```

```
}
```

5.2 コードやデータをページ境界に配置する方法

前の質問に関連して、プログラムのコードやデータをページ境界に配置する方法を教えてください。

OpenVMS AXP での SYS\$CRMPSC() システムサービスのように、引数に渡されるデータ領域がページ境界に置かれなければならない場合の指定方法はいくつかあります。例としてグローバル変数 `int iarray[128]` を取り上げます。

1. DEC C の `_align` を使用し、ソースコードで指定を行なう `_align` は ANSI Standard には無い、DEC C の言語拡張なので、`#pragma nostandard` と組み合わせて使用します。

```
#pragama nostandard
    _align( page ) int iarray[128];
#pragma standard
```

2. リンク時にプログラムセクションの属性に "PAGE" 指定をつける。

```
$ LINK/MAP xxx, SYS$INPUT/OPTION
PSECT_ATTR = IARRAY,PAGE
```

3. リンク時にプログラムセクションの属性に "SOLITARY" 指定をつける。

```
$ LINK/MAP xxx, SYS$INPUT/OPTION
PSECT_ATTR = IARRAY,SOLITARY
```

いずれの場合もリンクマップ (リンクマップはプログラムのリンク時に /MAP 修飾子を指定することで作成できます。) から IARRAY がページ境界に置かれていることを確認できます。

5.3 コンディションハンドラの設定方法

コンディションハンドラをセットしたプログラムを VAX から Alpha に持ってきました。とくあえずコンパイル、リンクはできたのですが動きが変です。VAX と Alpha ではコンディションハンドラの使い方が違うのでしょうか？

OpenVMS VAX では、プログラム実行時にコンディションハンドラが設定されます。VAX ではプロシジャ呼び出し命令である CALLG, CALLS によってレジスタ内容が保存され、フレームポインタ FP が指し示すコールフレームの先頭に、コンディションハンドラのアドレスが設定されます。

しかし Alpha AXP には CALLx 命令はありません。OpenVMS AXP の標準呼び出し規則 (Calling Standard) では、コールフレームの作成、レジスタの保存、コンディションハンドラの設定、引数リストの作成、リンケージセクションポインタの正規化といった処理は、OpenVMS AXP では、アプリケーションプログラムの責任で行なわなければなりません。

通常この処理を行なうコードはコンパイラによって自動生成されます。この自動生成されるコード部分は、プロローグ・コードと呼ばれます。つまり ユーザ定義のコンディションハンドラを設定するには、コンパイラにそういうコードを生成するように指示しなければなりません。

この目的のために LIB\$ESTABLISH() は OpenVMS AXP でも使用可能です。LIB\$ESTABLISH() は OpenVMS VAX では実行時ライブラリ関数として実装されていますが、OpenVMS AXP では関数ではありません。OpenVMS AXP 上で稼動するコンパイラは、ソースプログラム中に LIB\$ESTABLISH() を発見すると、コンディションハンドラを設定するためのコードをプロローグコード中に作成します。(コンパイル時に /LIST/MACHINE_CODE 修飾子を付加することで、確認可能です。)

また OpenVMS AXP では、メカニズムアレイのレイアウトも大幅に変更されています。OpenVMS AXP でのメカニズムアレイのレイアウトは以下のプロフラムを参照してください。さらに、OpenVMS VAX では浮動小数点演算関連のエラーの多くはフォルトでした。これらのエラーが発生した場合、ランタイムライブラリ LIB\$SIM_TRAP(), LIB\$FIXUP_FAULT() を使用してフォルトをトラップに変換して処理の続行を行なうのが一般的でした。OpenVMS AXP での浮動小数点演算関連のエラーは SS\$HPARITH トラップにまとめられました。このため、ランタイムライブラリ LIB\$SIM_TRAP(), LIB\$FIXUP_FAULT() はサポートされなくなりました。これらのエラーがフォルトではなくなったため、不必要になったためです。

以下に OpenVMS AXP システムでのコンディションハンドラの設定例を示します。

```
/*
* OpenVMS AXP システムでのコンディションハンドラの設定例
*
* LIB$ESTABLISH() は RTL ではなくなりました。コンパイラが LIB$ESTABLISH() を
* 発見すると、該当するハンドラ設定コードがオブジェクトコード中に作成されます。
* これは Calling Standard の変更によるためです。OpenVMS VAX ではプログラム実行時
* にコールフレームの先頭 (FP) にコンディションハンドラのアドレスを設定すると
* いう手法が用いられていましたが、OpenVMS AXP ではこの手法は使用されません。
```

```

*/

#include <stdio.h>
#include <ssdef.h>
#include <math.h>
#include <ints.h>

/*
 * Open VMS AXP のメカニズムアレイのレイアウト
 * シグナルアレイのレイアウトは OpenVMS VAX と同じ。
 */
typedef struct{
    uint32 chf$is_mch_args;
    uint32 chf$is_flags;
    uint64 chf$ph_mch_frame;
    uint32 chf$is_mch_depth;
    uint32 chf$is_mch_resvd1;
    uint64 chf$ph_mch_daddr;
    uint64 chf$ph_mch_esf_addr;
    uint64 chf$ph_mch_sig_addr;
    uint64 chf$ih_mch_savr0;    /* 整数レジスタ */
    uint64 chf$ih_mch_savr1;
    uint64 chf$ih_mch_savr16;
    uint64 chf$ih_mch_savr17;
    uint64 chf$ih_mch_savr18;
    uint64 chf$ih_mch_savr19;
    uint64 chf$ih_mch_savr20;
    uint64 chf$ih_mch_savr21;
    uint64 chf$ih_mch_savr22;
    uint64 chf$ih_mch_savr23;
    uint64 chf$ih_mch_savr24;
    uint64 chf$ih_mch_savr25;
    uint64 chf$ih_mch_savr26;
    uint64 chf$ih_mch_savr27;
    uint64 chf$ih_mch_savr28;
    uint64 chf$fh_mch_savef0;
    uint64 chf$fh_mch_savef1; /* 浮動小数点レジスタ */
    uint64 chf$fh_mch_savef10;
    uint64 chf$fh_mch_savef11;
    uint64 chf$fh_mch_savef12;
    uint64 chf$fh_mch_savef13;
    uint64 chf$fh_mch_savef14;
    uint64 chf$fh_mch_savef15;
    uint64 chf$fh_mch_savef16;
    uint64 chf$fh_mch_savef17;
    uint64 chf$fh_mch_savef18;

```

```

uint64 chf$fh_mch_savef19;
uint64 chf$fh_mch_savef20;
uint64 chf$fh_mch_savef21;

uint64 chf$fh_mch_savef22;
uint64 chf$fh_mch_savef23;
uint64 chf$fh_mch_savef24;
uint64 chf$fh_mch_savef25;
uint64 chf$fh_mch_savef26;
uint64 chf$fh_mch_savef27;
uint64 chf$fh_mch_savef28;
uint64 chf$fh_mch_savef29;
uint64 chf$fh_mch_savef30;
}MechanismArray;

# define SWCOMPLETE 1
# define INVFLT (1<<1) /* IEEE only */
# define FLTDIV (1<<2) /* floating divide by 0 */
# define FLTOVF (1<<3) /* floating overflow */
# define FLTUND (1<<4) /* floating underflow */
# define FLTCNV (1<<5) /* invalid floating operation or conversion */
# define INTCNV (1<<6) /* invalid integer operation or conversion */

main()
{
    extern int handler();
    int LIB$ESTABLISH();

    int i, ia = 10, ic;
    float a = 10.0, c;

    LIB$ESTABLISH( handler );

    for( i= -10; i<10; i++){
        c = a/(float)i;
        printf("%f/%d = %f¥n", a,i,c);
    }

    for( i= -10; i<10; i++){
        ic = ia/i;
        printf("%d/%d = %d¥n", ia,i,ic);
    }
}

int handler( unsigned long *signal, MechanismArray *mech )
{

```

```
unsigned long status, SYS$UNWIND(), LIB$STOP();
```

```
switch( signal[1] ){
  case SS$_HPARITH :
```

```
/*
```

```
SS$_HPARITH のシグナルアレイ
```

```
+-----+
| arg_count(6) | 引数の数
+-----+
| status      | SS$_HPARITH が入っている。
+-----+
| lmask       | セーブされた I レジスタ
+-----+
| fmask       | セーブされた F レジスタマスク
+-----+
| summary     | どんな種類の SS$_HPARITH か?
+-----+
| pc          | セーブされた PC ( trap shadow 内 )
+-----+
| ps          | セーブされた PS ( 下位 32 ビット)
+-----+
```

```
*/
```

```
if((signal[4] & FLTDIV) == FLTDIV ){
  printf("浮動小数点 0 除算発生!");
}else if((signal[4] & FLTOVF) == FLTOVF ){
  printf("浮動小数点オーバーフロー発生!");
}else if((signal[4] & FLTUND) == FLTUND ){
  printf("浮動小数点アンダーフロー発生!");
}
printf("この結果は無視してください ");
status = SYS$UNWIND( &(mech->chf$is_mch_depth ), 0 );
if( ~status & 1 ) LIB$STOP( status );
break;
```

```
case SS$_INTDIV :
```

```
printf("整数 0 除算発生。この結果は無視してください ");
status = SYS$UNWIND( &(mech->chf$is_mch_depth ), 0 );
if( ~status & 1 ) LIB$STOP( status );
break;
```

```
case SS$_INTOVF :
```

```
printf("整数オーバーフロー発生。この結果は無視してください ");
status = SYS$UNWIND( &(mech->chf$is_mch_depth ), 0 );
if( ~status & 1 ) LIB$STOP( status );
break;
```

```
default :
```

```
printf("Cannot handle this condition! %d¥n", signal[1]);
return SS$_RESIGNAL;
```

```
}  
}
```

5.4 ランタイムライブラリの作成方法

OpenVMS VAX で使用していた共有可能実行時ライブラリ(ランタイムライブラリ Run-Time Library) を OpenVMS AXP にポーティングしようとしています。転送ベクトルの記述方法を教えてください。

OpenVMS VAX では 転送ベクトル (transfer vector) を用いて共有可能イメージの上位互換を保証していました。Alpha AXP では転送ベクトルは必要ありません。リンク時の SYMBOL_VECTOROR 指定をするだけで共有可能実行時ライブラリを作成できるようになりました。

例えば、共有イメージに 2 つの関数 FOO と BAR がある場合、リンク時のオプションファイルに、

```
SYMBOL_VECTOR=(FOO=PROCEDURE, BAR=PROCEDURE)
```

と記述します。詳細はリンカーのマニュアルを参照してください。

以下に簡単な実行時ライブラリの作成例を示します。

```
$! OpenVMS AXP でのRTL の作成方法
$! RTL_TEST.COM
$!
$!
$ set verify
$ cc/object=test.obj sys$input
#include <stdio.h>
main()
{
    int int_add(), int_sub(), int_mul(), int_div();
    int a, b;

    printf("a (int) : ");
    scanf("%d", &a );
    printf("b (int) : ");
    scanf("%d", &b );

    printf("add(a,b) = %d\n", int_add(a,b));
    printf("sub(a,b) = %d\n", int_sub(a,b));
    printf("mul(a,b) = %d\n", int_mul(a,b));
    printf("div(a,b) = %d\n", int_div(a,b));
}
$ cc/object=myshr.obj sys$input
int int_add(int ia, int ib)
{
```

```
    return( ia + ib );
}
int int_sub(int ia, int ib)
{
    return( ia - ib );
}
int int_mul(int ia, int ib)
{
    return( ia * ib );
}
int int_div(int ia, int ib)
{
    return( ia / ib );
}
$ link/shareable myshr,sys$input/option
GSM=LEQUAL,1,10000
SYMBOL_VECTOR=( SPARE,-
int_add = procedure,-
int_sub = procedure,-
int_mul = procedure,-
int_div = procedure )
$ link test,sys$input/option
sys$disk:[ ]myshr/shareable
$ define/user myshr sys$disk:[ ]myshr
$ define/user sys$input sys$command
$ run test
$ exit
```

5.5 浮動小数点データの変換方法

OpenVMS AXP では IEEE 浮動小数点データ型が扱えるそうですが Big Endian の IEEE 浮動小数点データ型も扱えますか？ また、これまで VAX で作成した F 浮動小数点データを IEEE 浮動小数点型に変換できますか？

OpenVMS は全てのメモリ内データを Little Endian で持っています。内部でバイト順序を変更することで Big Endian のバイナリファイルを扱うことも可能ですが、内部表現はあくまでも Little Endian 型式になります。

5.5.1 DEC C を使用する場合

DEC C でデータ変換を行なうためにはランタイムライブラリ関数 `CVT$CONVERT_FLOAT()` を使用します。この関数は浮動小数の内部フォーマットを様々な型式に変換することができます。変換可能なデータ型は以下のとおりです。

`CVT$CONVERT_FLOAT()` の詳細は、OpenVMS RTL Library (`LIB$`) Manual を参照してください。

表 5-1 `CVT$CONVERT_FLOAT()` でサポートされるデータ型

サポートされるデータ型	データ長 (バイト数)
VAX F 浮動小数点データ型	4
VAX D 浮動小数点データ型	8
VAX G 浮動小数点データ型	8
VAX H 浮動小数点データ型	16
IEEE S 浮動小数点データ型	4
IEEE T 浮動小数点データ型	8
IBM SHORT 浮動小数点データ型	4
IBM LONG 浮動小数点データ型	8
CRAY 浮動小数点データ型	8

以下に `CVT$CONVERT_FLOAT()` 関数の使用例を示します。

```

/*
 * CVT$CONVERT_FLOAT() を用いて VAX_D_Floating を IEEE_T_Floating に
 * 変換するプログラムです。
 *
 * globalvalue を用いているため /STANDARD=VAXC を指定して下さい。
 * また、このプログラムの場合、/FLOATING=D_FLOAT も忘れないでください。
 * $ CC/STANDRAD=VAXC/FLOATING=D_FLOAT でコンパイルしてください。
 */

```

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

globalvalue CVT$K_VAX_D;
globalvalue CVT$K_IEEE_T;
globalvalue CVT$M_ROUND_TO_NEAREST;
globalvalue CVT$_NORMAL;

unsigned long main(int argc, char** argv)
{
    extern unsigned long LIB$STOP();
    extern unsigned long CVT$CONVERT_FLOAT();

    long double VAX_D_Floating;
    unsigned long IEEE_T_Floating[2];
    unsigned long status;
    int i;

    for (i = 0; i < 5; i++){
        VAX_D_Floating = (long double)i;
        memset(IEEE_T_Floating, 0, 8);

        if (status = CVT$CONVERT_FLOAT(&VAX_D_Floating, CVT$K_VAX_D,
            &IEEE_T_Floating, CVT$K_IEEE_T, NULL) != CVT$_NORMAL)
            (void)LIB$STOP(status);
        fprintf(stdout, "VAX_F_Floating = %08x\t \tIEEE_T_Floating = %08x\n",
            VAX_D_Floating, IEEE_T_Floating);

        VAX_D_Floating = (long double)0;

        if (status = CVT$CONVERT_FLOAT(&IEEE_T_Floating, CVT$K_IEEE_T,
            &VAX_D_Floating, CVT$K_VAX_D, NULL) != CVT$_NORMAL)
            (void)LIB$STOP(status);
        fprintf(stdout, "IEEE_T_Floating = %08x\t \tVAX_F_Floating = %08x\n\n",
            IEEE_T_Floating, VAX_D_Floating);
    }
    exit(EXIT_SUCCESS);
}

```

【実行結果】

VAX F_Floating = 00000000

IEEE_T_Floating = 7fe6fa18

IEEE_T_Floating = 7fe6fa18	VAX F_Floating = 00000000
VAX F_Floating = 00004080	IEEE_T_Floating = 7fe6fa18
IEEE_T_Floating = 7fe6fa18	VAX F_Floating = 00004080
VAX F_Floating = 00004100	IEEE_T_Floating = 7fe6fa18
IEEE_T_Floating = 7fe6fa18	VAX F_Floating = 00004100
VAX F_Floating = 00004140	IEEE_T_Floating = 7fe6fa18
IEEE_T_Floating = 7fe6fa18	VAX F_Floating = 00004140
VAX F_Floating = 00004180	IEEE_T_Floating = 7fe6fa18
IEEE_T_Floating = 7fe6fa18	VAX F_Floating = 00004180

5.5.2 DEC Fortran を使用する場合

一方 DEC Fortran では OPEN 文に CONVERT= キーワードを指定することにより、ファイル読み込み時に自動的にデータ変換を行なうことができます。
以下に簡単な使用例を示します。

```
$ FORTRAN/OBJECT=MAKE_VAXD.OBJ/FLOAT=D_FLOAT SYSS$INPUT
C VAX D 浮動小数点データを作成する。
```

```
REAL*8 float_data(100)

OPEN(UNIT=1, FILE='VAX_D.DAT',
1    FORM='UNFORMATTED',STATUS='NEW')

DO I=1,100
  float_data(I) = float( I )
END DO

WRITE(1)(float_data(I),I=1,100)
CLOSE(1)
END
```

```
$ FORTRAN/OBJECT=CONVERT.OBJ SYSS$INPUT
```

```
C VAX D 浮動小数点データファイルを IEEEES 浮動小数点データファイルに変換する
```

```
PROGRAM CONVERT_TEST
REAL*8 float_data(100)
```

```
C VAX D 浮動小数点データファイルをオープン
```

```
OPEN(UNIT=1, FILE='VAX_D.DAT', CONVERT='VAXD',
```

```
1   FORM='UNFORMATTED', STATUS='OLD')
```

```
C Little Endian の IEEE T 浮動小数点データファイルをオープン
  OPEN(UNIT=2, FILE='IEEE_T.DAT', CONVERT='LITTLE_ENDIAN',
  1   FORM='UNFORMATTED', STATUS='NEW')

  READ(1) (FLOAT_DATA(1), I=1,100)
  WRITE(2) (FLOAT_DATA(1), I=1,100)

  CLOSE(1)
  CLOSE(2)

  END
```

```
$ FORTRAN/OBJECT=CONFIRM.OBJ/FLOAT=IEEE_FLOAT SYS$INPUT
```

C 変換したデータが正しい事を確認する。

```
C Little Endian の IEEE T 浮動小数点データファイルをオープン

  REAL*8  FLOAT_DATA(100)

  OPEN(UNIT=2, FILE='IEEE_T.DAT',
  1   FORM='UNFORMATTED', STATUS='OLD')

  READ(2) (FLOAT_DATA(1), I=1,100)
  WRITE(6, *) (FLOAT_DATA(1), I=1,100)

  CLOSE(2)

  END
```

```
$ LINK MAKE_VAXD
$ LINK CONVERT
$ LINK CONFIRM
$ RUN MAKE_VAXD
$ RUN CONVERT
$ RUN CONFIRM
```

1.0000000000000000	2.0000000000000000	3.0000000000000000
4.0000000000000000	5.0000000000000000	6.0000000000000000
7.0000000000000000	8.0000000000000000	9.0000000000000000
10.0000000000000000	11.0000000000000000	12.0000000000000000
13.0000000000000000	14.0000000000000000	15.0000000000000000
16.0000000000000000	17.0000000000000000	18.0000000000000000
19.0000000000000000	20.0000000000000000	21.0000000000000000

22.00000000000000	23.00000000000000	24.00000000000000
25.00000000000000	26.00000000000000	27.00000000000000
28.00000000000000	29.00000000000000	30.00000000000000
31.00000000000000	32.00000000000000	33.00000000000000
34.00000000000000	35.00000000000000	36.00000000000000
37.00000000000000	38.00000000000000	39.00000000000000
40.00000000000000	41.00000000000000	42.00000000000000
43.00000000000000	44.00000000000000	45.00000000000000
46.00000000000000	47.00000000000000	48.00000000000000
49.00000000000000	50.00000000000000	51.00000000000000
52.00000000000000	53.00000000000000	54.00000000000000
55.00000000000000	56.00000000000000	57.00000000000000
58.00000000000000	59.00000000000000	60.00000000000000
61.00000000000000	62.00000000000000	63.00000000000000
64.00000000000000	65.00000000000000	66.00000000000000
67.00000000000000	68.00000000000000	69.00000000000000
70.00000000000000	71.00000000000000	72.00000000000000
73.00000000000000	74.00000000000000	75.00000000000000
76.00000000000000	77.00000000000000	78.00000000000000
79.00000000000000	80.00000000000000	81.00000000000000
82.00000000000000	83.00000000000000	84.00000000000000
85.00000000000000	86.00000000000000	87.00000000000000
88.00000000000000	89.00000000000000	90.00000000000000
91.00000000000000	92.00000000000000	93.00000000000000
94.00000000000000	95.00000000000000	96.00000000000000
97.00000000000000	98.00000000000000	99.00000000000000
100.00000000000000		

5.6 SYS\$SYSTEM:SYS.STB がない

OpenVMS VAX V6.1 で稼動しているプログラムがあります。このプログラムには OS が使用しているグローバルシンボルを参照している部分があり VAX では SYS\$SYSTEM:SYS.STB とリンクしていました。このプログラムを Alpha AXP にポータリング中なのですが、Alpha AXP には SYS.STB が見当たりません。どうしたらいいのでしょうか？

ご質問のプログラムは以下のようなものです。

```

/*
 * SHOW MEMORY/PHYSICAL_PAGES とほぼ同じ動きをします。
 */

#include <stdio.h>

#pragma nostandard

globalref unsigned long SCH$GL_FREECNT;
globalref unsigned long SCH$GL_MFVCNT;

#ifdef VMSV5
globalref unsigned long MMG$GL_MEMSIZE;
#endif

globalref unsigned long MMG$GL_PHYPGCNT;

main()
{
    unsigned long total_pages, free_pages, modified_pages;

    total_pages    = ( MMG$GL_PHYPGCNT < MMG$GL_MEMSIZE ) ?
                    MMG$GL_PHYPGCNT : MMG$GL_MEMSIZE;
    free_pages     = SCH$GL_FREECNT;
    modified_pages = SCH$GL_MFVCNT;

    printf("Total : %d\nFree : %d\nIn Use : %d\nModified : %d\n",
           total_pages, free_pages, total_pages - free_pages,
           modified_pages);
}

```

このプログラムは VAX では、以下のコマンドでリンクされていました。

```
$ LINK program, SYS$SYSTEM:SYS.STB/SELECTIVE
```

Alpha AXP では、SYS.STB とリンクするかわりに LINK コマンドの修飾子 /SYSEXE を使用してください。

```
$ LINK/SYSEXE program
```

このコマンドにより、プログラムはシステム共有可能イメージのひとつ、ALPHA\$LOADABLE_IMAGES:SYS\$BASE_IMAGES.EXE とリンクされます。以下は HELP からの抜粋です。

LINK

Qualifier_Descriptions

/SYSEXE

On AXP systems, directs the linker to process the system shareable image, SYS\$BASE_IMAGE.EXE, in a link operation. The linker looks for SYS\$BASE_IMAGE.EXE in the directory pointed to by the logical name ALPHA\$LOADABLE_IMAGES.

Format

```
/SYSEXE[=[NO]SELECTIVE]
```

```
/NOSYSEXE (default)
```

Additional information available:

Qualifier_Values

SELECTIVE

When you specify the SELECTIVE keyword, the linker processes the SYS\$BASE_IMAGE.EXE file selectively, including only those symbols from the global symbol table of the SYS\$BASE_IMAGE.EXE file that were referenced by input files previously processed in the link operation.

NOSELECTIVE

When you specify the NOSELECTIVE keyword, the linker includes all the symbols from the SYS\$BASE_IMAGE.EXE global symbol table in the link operation.

When the /SYSEXE qualifier is specified without a keyword, the linker processes the executive image selectively.