OpenVMS AXP
オペレーティング・システム

OpenVMS AXP オペレーティング・システムへの移行：システム移行の手引き

AA-PU8KC-TE

ソフトウェア・バージョン: OpenVMS AXP 6.1

日本ディジタル イクイップメント株式会社

1994 年 7 月

このマニュアルは， CD-ROM で提供されます。

本書は，日本語 VAX DOCUMENT V 2.1 を用いて作成しています。

目次

用語集

索引

図

表

まえがき

本書の目的

本書は，VAX システムから AXP システムに既存のアプリケーションを移行する処理の概要をまとめた解説書であり，実際の移行作業の計画を立てるのに役立つ情報が記載されています。本書では，移行を計画するときに判断しなければならない事柄と，それらの判断を下すのに必要な情報の入手方法について説明します。実際の移行プロセスの技術的な説明についの詳細は，このまえがきの "参考文献 "という節に示した解説書を参照してください。

対象読者

本書は，OpenVMS VAX システムから OpenVMS AXP システムに移行する際に，アプリケーションを評価し，移行の計画を立てる責任者を対象にしています。

本書の構成

本書は，6 つの章と 2 つの付録，用語集で構成されています。

第 1 章　以下のように，OpenVMS と VAX アーキテクチャおよび Alpha AXP アーキテクチャの関係をまとめます。

- OpenVMS AXP と OpenVMS VAX との間で互換性が高く維持されている部分
- Alpha AXP アーキテクチャの機能を他の RISC アーキテクチャの機能，および VAX アーキテクチャの機能と比較した，AXP システムの特徴の概観

第 2 章　AXP システムに移行するプロセスの概要を示します。特に次の情報を示します。

- 移行プロセスの各段階の概要
- 2 つの主な移行の手段，つまりソース・コードの再コンパイルと VAX イメージのトランスレーション
- DEC が提供する移行のサポート

第 3 章以降では，移行プロセスの実際の操作を説明し， VAX アプリケーションを移行するための作業の各段階と，効果的な移行計画の作成方法について説明します。

第 3 章　アプリケーション全体を分析し，正確に何を移行するのかを判断する方法について説明します。

第 4 章　2 つの主な移行の手段の相違点を考慮し，アプリケーションを移行するときに，その手段の選択について判断しなければならない問題点を説明します。また，アプリケーションの各部分を分析して，アーキテクチャの違いが，移行にどのような影響を与えるかを判断する方法と，これらの違いを解決するために，何が必要であるかを評価する方法について説明します。

第 5 章　VAX から AXP への移行計画の作成方法について説明します。

第 6 章　実際の移行の実施について，移行環境の設定をはじめ，移行したアプリケーションを新しい環境に統合する方法まで，詳しく説明します。

付録 A　OpenVMS VAX から OpenVMS AXP に移行するアプリケーションを評価するときに使用できるチェック・リストを示します。

付録 B　移行計画の例を示します。

用語集　本書で紹介した用語の定義を示します。

参考文献

OpenVMS AXP への移行に関する参考文献として，次のマニュアルがあります。

- 『OpenVMS AXP オペレーティング・システムへの移行：再コンパイルと再リンク』

  このマニュアルでは， VAX アプリケーションを再コンパイルし，再リンクすることにより，そのアプリケーションの AXP バージョンを構築する方法について説明します。既存のアプリケーションが VAX アーキテクチャ固有の機能に依存している場合には (たとえば，ページ・サイズや同期，条件処理などに関する仮

定など), ネイティブな AXP バージョンを作成するために，これらの部分を変更しなければなりません。さらに，このマニュアルでは，ネイティブな AXP のコンポーネントが，トランスレートされた VAX のコンポーネントと相互にやりとりできるようなアプリケーションの作成方法についても説明します。

- 『DECmigrate for OpenVMS AXP Systems Translating Images』

  このマニュアルでは， VAX Environment Software Translator (VEST) ユーティリティについて説明します。 VEST は， DECmigrate for OpenVMS AXP というオプションのレイヤード・プロダクトに付属しており， VAX アプリケーションを OpenVMS AXP に移行する処理をサポートします。このマニュアルでは，大部分のユーザ・モード VAX イメージを， AXP システムで実行可能なトランスレートされたイメージに変換するために VEST を使用する方法，トランスレートされたイメージの実行時性能を向上する方法， VAX イメージの中で AXP と互換性のない部分を元のソース・ファイルでトレースするために VEST を使用する方法，ネイティブなランタイム・ライブラリとトランスレートされたランタイム・ライブラリの間で互換性を維持するために VEST を使用する方法について説明します。このマニュアルではまた，完全な VEST コマンド・リファレンスも示します。

- 『Migrating to an OpenVMS AXP System: Porting VAX MACRO Code』

  このマニュアルでは，VAX MACRO-32 Compiler for OpenVMS AXP を使用してVAX MACROコードを AXP システムに移植する方法およびコンパイラの機能，移植不可能なコーディング様式の識別法，このようなコーディング様式のかわりとなる適切な方法などについて説明します。このマニュアルではまた，コンパイラの修飾子，ディレクティブ，および組み込み機能と， OpenVMS AXP に移植するために作成されたシステム・マクロの詳細な説明を示したリファレンスも記載されています。

この他にも，次に示すような移行プロセスで役立つ解説書があります。

- 『OpenVMS Linker Utility Manual』

  このマニュアルには， AXP イメージを作成するために OpenVMS リンカ・ユーティリティを使用する方法が詳しく説明されています。

- 『A Comparison of System Management on OpenVMS AXP and OpenVMS VAX』

  このマニュアルには，システム全体を移行するにあたって，システム管理に関して考慮しなければならない情報がまとめられています。

- 『OpenVMS Compatibility Between VAX and AXP』

  このマニュアルには， OpenVMS VAX と OpenVMS AXP の相違点に関する詳しい情報がまとめられています。

- 『Alpha Architecture Reference Manual』

  このマニュアルには， Alpha AXP アーキテクチャの定義が示されています。機械語プログラムから見た Alpha AXP CPU の動作の，詳しい説明が記載されています。

- 『VMS for Alpha Platforms: Internals and Data Structures』

  このマニュアルでは， OpenVMS AXP オペレーティング・システムの内部構造について説明されています。

- 『OpenVMS Calling Standard』

  このマニュアルでは，別のプロシージャを呼び出すときにプロシージャが使用する規則を規定し，これらの規則をサポートするための引数の受け渡し方法とその構造を定義します。

注意

本書は， 『Migrating to an OpenVMS Alpha System: Planning for Migration』の日本語翻訳版です。日本 DEC の提供するサービスについては記述と異なる場合がありますので，詳細については日本 DEC にお問い合わせください。

表記法

注意

本書では，以下の表記がそれぞれ，

OpenVMS AXP	→	OpenVMS AXP オペレーティング・システム
OpenVMS VAX	→	OpenVMS VAX オペレーティング・システム
OpenVMS	→	OpenVMS AXP オペレーティング・システム および OpenVMS VAX オペレーティング・システム の両方

を意味しています。

本書では次の表記法を使用します。

表記法	意味
Return	四角形で囲まれたこの記号は，キーボードのキーを押すことを示します。たとえば，Return は Return キーを押すことを示します。
Ctrl/x	Ctrl/x の記号は， Ctrl キーを押しながら，同時にあるキーを押すことを示します。たとえば，Ctrl/c は Ctrl キーと c 文字キーを同時に押します。
PF1/X	PF1/X の記号は，最初に PF1 キーを押し，離した後，別のキーまたはポインティング・デバイス・ボタンを押すことを示します。
. . .	例の中で水平反復記号は，次のいずれかを示します。 • 文中のオプショナル引数が省略されていること • 前の項目 (1 つ以上) を 1 回以上繰り返すことができること • 追加パラメータ，値，あるいは他の情報を入力でること
. . .	垂直反復記号は，コード例やコマンド形式から項目が省略されていることを示します。このように項目が省略されるのは，その項目が説明している内容にとって重要でないからです。
( )	括弧は，複数のオプションを選択するときに，選択項目を括弧で囲まなければならないことを示す。

表記法	意味
[ ]	大括弧は，項目が省略可能であることを示します (しかし，VMS ファイル指定のディレクトリ名の構文や，代入文の部分文字列指定の構文では，大括弧は省略可能ではありません)。
{ }	中括弧は，必ず 1 つを選択しなければならない項目を囲むために使用します。
太字	太字のテキストは，用語集で説明されている新しい用語を導入する場合や，引数，属性，条件の名前を示すために使用します。 また，マニュアルのオンライン・バージョンでユーザ入力を示す場合も，太字のテキストを使用します。
イタリック体	イタリック体のテキストは，システム・メッセージやコマンド・ラインの中で，変化する可能性のある情報を表現します。
英大文字	英大文字は，コマンド，修飾子，パラメータ，ルーチン名，ファイル名，ファイル保護コード名，システム特権の短縮形を示します。
—	コード例で使用されているハイフンは，要求に対する追加引数が後続の行に指定されることを示します。
数字	特に示した場合を除き，説明文の内部で使用している数字はすべて 10 進数です。数値が 10 進数以外 (2 進数， 8 進数， 16 進数) の場合には，そのことが明記されます。
「 」	かぎ括弧は，この製品のドキュメント構成に含まれるマニュアル名を示します。
『 』	二重かぎ括弧は，この製品のドキュメント構成に含まれないマニュアル名または，別の製品のマニュアル名を示します。

1

VAX，Alpha AXP，および OpenVMS

多くのアプリケーションにとって，OpenVMS VAX から OpenVMS AXP への移行 (migration) は簡単です。アプリケーションがユーザ・モードでのみ実行され，標準的な高級言語で作成されている場合には，ほとんどの場合，AXP コンパイラを使用してそのアプリケーションを再コンパイルし，再リンクすることにより，AXP システムで実行可能なバージョンを作成できます。本書では，移行するアプリケーションを評価する方法，およびもっと複雑で特殊な場合の対処方法について説明します。

1.1 VAX システムと AXP システムの互換性

OpenVMS AXPオペレーティング・システムは，OpenVMS VAX のユーザ，システム管理，およびプログラミングの環境とできるだけ互換性 (compatibility) を維持するように設計されています。一般的なユーザとシステム管理者にとって，OpenVMS AXP は OpenVMS VAX と同じインターフェイスを備えています。プログラマにとっての目標は，「再コンパイル，再リンク，実行」という移行のモデルにできるだけ近づけることです。

OpenVMS VAX システムで動作しているアプリケーションの場合，ほとんどの部分は OpenVMS AXP システムでも変更されません。

ユーザ・インターフェイス

- DIGITAL コマンド言語 (DCL)

  DIGITAL コマンド言語 (DCL) は OpenVMS に対する標準的なユーザ・インターフェイスであり，OpenVMS AXP でも変更されません。OpenVMS VAX で使用できるすべてのコマンド，修飾子，およびレキシカル関数は OpenVMS AXP でも使用できます。

- **コマンド・プロシージャ**

  OpenVMS VAX の以前のバージョンを対象に作成されたコマンド・プロシージャは， OpenVMS AXP システムでもまったく変更せずに動作します。しかし，ビルド・プロシージャなどのある特定のコマンド・プロシージャは，新しいコンパイラ修飾子やリンカ・スイッチに対応できるように変更しなければならないことがあります。リンカ・オプション・ファイルも変更が必要な場合があり，特に共有可能イメージ (shareable image) の場合は変更が必要となります。

- DECwindows

  ウィンドウ・インターフェイスである DECwindows Motif は変更されません。

- DECforms

  DECforms インターフェイスは変更されません。

- **エディタ**

  2 つの標準的な OpenVMS エディタである EVE と EDT は変更されません。

システム管理インターフェイス

システム管理ユーティリティはほとんど変更されません。ただし，おもな例外が 1 つあります。それはデバイス構成管理機能で， OpenVMS VAX システムでは System Generation utility (SYSGEN) で提供される機能ですが，OpenVMS AXP では System Generation utility (SYSMAN) で提供されます。詳しくは『A Comparison of System Management on OpenVMS AXP and OpenVMS VAX』を参照してください。

プログラミング・インターフェイス

概して，システム・サービスおよびランタイム・ライブラリ (RTL) 呼び出しインターフェイスは変更されません。引数の定義を変更する必要はありません。相違点がいくつかありますが，これらの相違点は，次の 2 種類に分類されます。

- 一部のシステム・サービスとランタイム・ライブラリ・ルーチン (メモリ管理システムと例外処理サービス) は， VAX システムと AXP システムとでは，少し異なる方法で動作します。詳しくは『OpenVMS System Services

Reference Manual』と『OpenVMS RTL Library (LIB$) Manual』を参照してください。

- 一部のランタイム・ライブラリ・ルーチンは VAX アーキテクチャに密接に関係しており，AXP システムでは意味がありません。これらのルーチンは次のとおりです。

ルーチン名	制約事項
LIB$DECODE_FAULT	VAX 命令をデコードする
LIB$DEC_OVER	VAX プロセッサ・ステータス・ロングワード (PSL) のみに適用される
LIB$ESTABLISH	AXP システムでは，類似する機能をコンパイラがサポートする
LIB$FIXUP_FLT	VAX PSL のみに適用される
LIB$FLT_UNDER	VAX PSL のみに適用される
LIB$INT_OVER	VAX PSL のみに適用される
LIB$REVERT	AXP システムではコンパイラがサポートする
LIB$SIM_TRAP	VAX コードに適用される
LIB$TPARSE	動作ルーチンのインターフェイスの変更が必要である。LIB$TABLE_PARSE に置換されている

これらのサービスとルーチンを呼び出す VAX イメージの大部分は，VEST (VAX Environment Software Translator) を使ってトランスレートし，OpenVMS AXP の TIE (Translated Image Environment) のもとで実行すれば，正しく動作します。TIE についての詳しい説明は，第 6.2.2.1 項と『DECmigrate for OpenVMS AXP Systems Translating Images』を参照してください。

データ

ODS-2 データ・ファイルのディスク上でのフォーマットは，VAX システムと AXP システムとで同じです。しかし，ODS-1 ファイルは OpenVMS AXP でサポートされません。

レコード管理サービス (RMS) とファイル管理インターフェイスは変更されていません。

IEEE リトル・エンディアン・データ型である S 浮動小数点 (S_floating) と T 浮動小数点 (T_floating) が追加されました。

大部分の VAX データ型 (data type) は Alpha AXP アーキテクチャでもそのまま使用できます。しかし，システム全体の性能を向上するために， H 浮動小数点 (H_floating) と完全な精度の D 浮動小数点 (G_floating) のハードウェアによるサポートはなくなりました。

AXP ハードウェアは D 浮動小数点データを処理のために G 浮動小数点に変換します。 VAX システムでは， D 浮動小数点は 56 ビット (D56) であり， 16 桁の精度です。 AXP システムでは， D 浮動小数点は 53 ビット (D53) であり，15 桁の精度です。

H 浮動小数点データ型と D 浮動小数点データ型は通常， G 浮動小数点または IEEE フォーマットのいずれかに変換されます。しかし， H 浮動小数点が必要な場合や， D56 (56 ビットの D 浮動小数点) の精度が必要な場合には，アプリケーションの一部をトランスレートしなければなりません。

データベース

標準的な DEC のデータベース (Rdb/VMS など) は， VAX システムと AXP システムで同様に機能します。

ネットワーク・インターフェイス

VAX システムと AXP システムはどちらも次のインターフェイスをサポートします。

- インターコネクト
  - Ethernet
  - X.25
  - FDDI
- プロトコル
  - DECnet (バージョン 6.1 のフェーズ IV; フェーズ V 以上)
  - TCP/IP
  - OSI
  - LAD/LAST

  - LAT (ローカル・エリア・トランスポート)
- 周辺装置接続
  - TURBOchannel
  - SCSI
  - Ethernet
  - CI
  - DSSI
  - XMI
  - Futurebus+
  - VME

1.2 VAX アーキテクチャと Alpha AXP アーキテクチャの相違点

VAX アーキテクチャは強力で柔軟な命令を備えた，複雑命令セット・コンピュータ (CISC)・アーキテクチャであり， VAX システム・ファミリ全体で使用されています。統一アーキテクチャの VAX ファミリを， OpenVMS オペレーティング・システムと組み合わせて使用すれば，アプリケーションを VAXstation で開発し，小型 VAX システムでプロトタイプを作成し，大型 VAX プロセッサのプロダクション環境で使用したり，フォールト・トレラントな VAXft プロセッサで実行することができます。 VAX システムのアプローチの利点は，個別のソルーションを変更し，大規模な問題全体のソルーションとして容易に統合できるということです。VAX プロセッサのハードウェア設計は特に，可用性の高いアプリケーションに適しています。たとえば，重要な使命を担うビジネス業務を実行するための，信頼性の高いアプリケーションや，様々な分散クライアント/サーバ環境でのサーバ・アプリケーションを実行するのに適しています。

DEC が実現した Alpha AXP アーキテクチャは，高性能の縮小命令セット・コンピュータ (RISC)・アーキテクチャであり， 1 つのチップで 64 ビット処理を実現できます。 64 ビットの仮想アドレスと物理アドレス， 64 ビットの整数， 64 ビットの浮動小数点数値を処理します。 64 ビット処理は特に，高い性能ときわめて大きいアドレッシング空間を必要とするアプリケーションにとって役立ちます。たとえ

ば，AXP プロセッサは，グラフィック処理アプリケーションや，膨大な数値を処理する経済予測や気象予報などのソフトウェア・アプリケーションに適しています。これらは，イメージ処理，マルチメディア，ビジュアライゼーション，シミュレーション，モデリングなどの処理を必要とします。

Alpha AXP アーキテクチャはスケーラブルでオープンなアーキテクチャとして設計されています。シングル・チップによる手のひらサイズのパームトップ・システムから，数千個のチップからなる超並列スーパーコンピュータまで，様々なシステムでの実現が可能です。このアーキテクチャはまた，OpenVMS AXP も含めて，複数のオペレーティング・システムをサポートします。

表 1–1 は，Alpha AXP アーキテクチャと VAX アーキテクチャの主な違いを示しています。

表 1–1 Alpha AXP アーキテクチャと VAX アーキテクチャの比較

Alpha AXP	VAX
• 64 ビット・アドレス	• 32 ビット・アドレス
• 64 ビット処理	• 32 ビット処理
• 複数のオペレーティング・システム: OpenVMS, OSF/1	• 1 つのオペレーティング・システム: OpenVMS
• 命令	• 命令
– 単純	– やや複雑
– すべて同じ長さ (32 ビット)	– 可変長
• ロード/ストア・メモリ・アクセス	• 操作とメモリ・アクセスを 1 つの命令に組み合わせることが可能
• アラインされていないデータに対しては重大な性能低下	• アラインされていないデータに対して中程度の性能低下
• 多くのレジスタ	• 比較的数の少ないレジスタ
• 命令は要求の順序と無関係に終了	• 命令は要求された順に終了
• 多段のパイプラインと分岐予測	• パイプラインは限定的に使用
• 大きいページ・サイズ (ハードウェアに応じて 8KB ～ 64KB)	• 小さいページ・サイズ (512 バイト)

RISC の一般的な特性

Alpha AXP アーキテクチャの特徴の一部は，新しい RISC アーキテクチャの典型的な特徴です。次の特徴は特に重要です。

- 単純な命令セット

  Alpha AXP アーキテクチャではかなり単純な命令を使用しており，これらはすべて 32 ビットの長さです。これらの共通の命令は 1 クロック・サイクルだけを必要とします。このようにサイズが統一された単純な命令の採用により，**複数命令発行** (mullti-instruction issue)や最適化された命令スケジューリングなどの方式を採用でき，その結果，きわめて高い性能を実現できます。

- 複数命令発行

  大部分の AXP プラットフォームは，1 クロック・サイクルで 2 つの命令を出します。将来のシステムでは，1 クロック・サイクルに 4 つの命令を出すことも可能になります。

- ロード/ストア操作モデル

  Alpha AXP アーキテクチャでは，32 個の 64 ビット整数レジスタと，32 個の 64 ビット浮動小数点レジスタを定義しています。大部分のデータ操作は，レジスタ間で実行されます。操作の前に，オペランドがメモリからレジスタにロードされます。操作が終了した後，結果はレジスタからメモリに格納 (ストア) されます。

  このように操作をレジスタ・オペランドに制限すれば，単純で統一された命令セットを使用できます。さらに，メモリ・アクセスを算術演算から分離することにより，完全なパイプライン，命令スケジューリング，および並列操作ユニットを実現できるシステムとして，大幅に性能を向上できます。

- マイクロコードの排除

  Alpha AXP アーキテクチャではマイクロコードを使用しないため，AXP プロセッサはマシン命令を実行するために，ランダム・アクセス・メモリ (RAM) からマイクロコード命令をフェッチするのに必要な時間を削減できます。

- 命令の並列実行と，命令のランダムな終了が可能なプロセッサ

  Alpha AXP アーキテクチャでは，命令が発行された順番に完了することは保証されません。その結果として，算術演算例外や浮動小数点例外は，最適化された命令列を乱さないように，少し時間をおいて報告されます (あいまいな例外報告，第 4.2.3.3 項を参照)。Alpha AXP アーキテクチャでは，このようにして性能を最大限に発揮できるように設計されています。

Alpha AXP 固有の特性

これまで説明した RISC の一般的な特性の他に，Alpha AXP アーキテクチャは，移行した VAX アプリケーションを AXP システムで実行するのに使用される多くの機能を備えています。Alpha AXP アーキテクチャでは次の機能が提供されます。

- H 浮動小数点と D 浮動小数点を除き，他のすべての VAX データ型に対するハードウェア・サポート (アプリケーションで H 浮動小数点，または D 浮動小数

点データを使用しているときに，どのような処理を実行しなければならないかについては，第 4.2.3.1.2 項を参照してください)。

- 4 種類のプロセッサ・モード (ユーザ，スーパーバイザ，エグゼクティブ，およびカーネル) などの特定の特権付きアーキテクチャ機能， 32 段階の割り込み優先順位レベル (IPL, Interrupt Priority Level)，および非同期システム・トラップ (AST)。
- 特権付きアーキテクチャ・ライブラリ (PAL) は PALcode と呼ばれる環境の一部であり， Change Mode (CHM*x*)， Probe (PROBE*x*)，キュー命令， REI など，特定の VAX 命令に対応する不可分な (atomic) 実行をサポートします。

2

移行プロセスの概要

VAX プログラムを AXP システムで実行するために変換するプロセスは，次の段階に分類されます。

1. 移行するコードを評価します。
   - アプリケーションのモジュールとその環境を確認します。他のプログラムに依存する部分があるかどうかも確認します。
   - 各モジュールのコードを調べ，移行にとって障害となる部分があるかどうかを確認します。
   - アプリケーションの各部分を AXP システムに移行するための最適な方法を判断します。
2. 移行計画を作成します。
3. 移行環境を設定します。
4. アプリケーションを移行します。
5. 移行したアプリケーションをデバッグし，テストします。
6. 移行したソフトウェアをソフトウェア・システムに統合します。

アプリケーションを OpenVMS AXP に移行するのに役立つように，多くのツールと DEC によるサービスが提供されます。これらのツールについては，本書で実際のプロセスを説明するときに示します。移行サービスについては，第 2.2 節にまとめられています。

2.1 移行の手段

AXP システムで実行するためにプログラムを変換する方法としては，次の 2 種類の方法があります。

- **再コンパイルと再リンク**

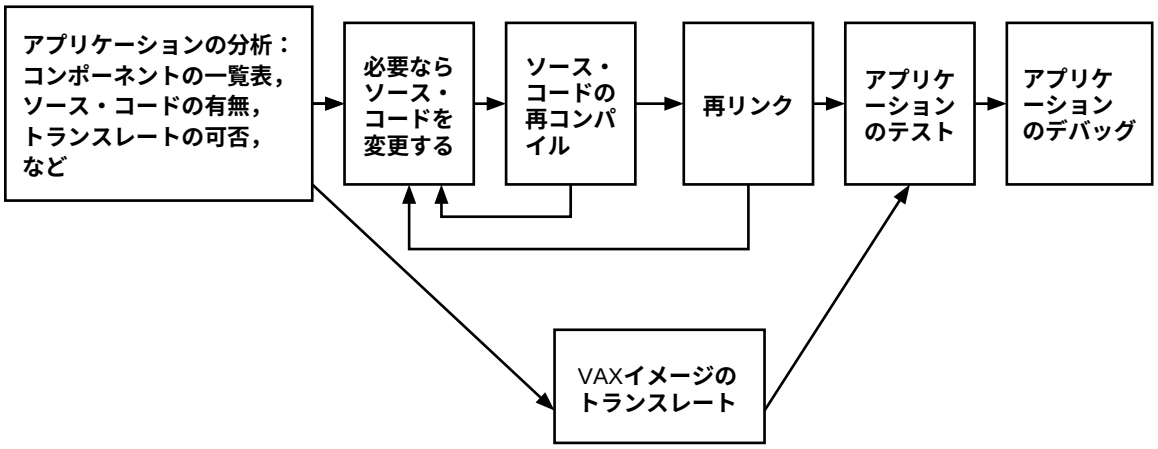
  この方法ではネイティブな AXP イメージが作成されます。

- **トランスレーション**

  この方法でも AXP イメージが作成されますが，一部のルーチンは TIE のもとでエミュレートされます。

これらの 2 種類の方法は，図 2–1 に示すとおりです。第 4.4 節では，移行方式を選択するときに考慮しなければならない事柄を説明しています。

図 2–1 VAX アプリケーションを AXP システムに移行する方法



JRD-4988A

再コンパイル

プログラムを OpenVMS VAX から OpenVMS AXP に変換するための，もっとも効果的な方法は， AXP コンパイラ (DEC C や DEC Fortran など) を使用してソース・コードを再コンパイルし，その後， OpenVMS リンカで適切なスイッチとオプション・ファイルを使用して，作成されたオブジェクト・ファイルと必要な他の共有可能なイメージを再リンクする方法です。この方法では， AXP システムのスピードを完全に活用できる，ネイティブな AXP イメージが作成されます。

トランスレーション

VAX システムと AXP システムにはいくつかの相違点がありますが， VAX Environment Software Translator (VEST) を使用すれば，大部分のユーザ・モードの VAX イメージを， AXP システムでエラーなしに実行することができます。 VEST は DECmigrate for OpenVMS AXP の一部です。例外については，第 4.4 節を参照してください。

この方法では，ソースを再コンパイルする場合より高いレベルで VAX との互換性を維持できますが，トランスレートされたイメージは再コンパイルされたイメージほど高い性能を実現できないため，トランスレーションは，再コンパイルが不可能な場合や実用的でない場合に，代わりとなる安全な手段として使用してください。たとえば，次の状況ではトランスレーションが適しています。

- OpenVMS AXP 用のコンパイラを使用できない場合
- ソース・ファイルを入手できない場合

VEST は VAX バイナリ・イメージ・ファイルをネイティブな AXP イメージにトランスレートします。このイメージは， AXP システム上の Translated Image Environment (TIE) のもとで実行されます (TIE は， OpenVMS AXP でバンドル・ソフトウェアとして提供される共有可能イメージです)。トランスレートされた VAX イメージは，エミュレータやインタプリタのもとで実行されるわけではなく (ただし，一部の例外があります)，元の VAX イメージで命令が実行する操作と同じ操作を実行する Alpha AXP 命令が，新しい AXP イメージに盛り込まれるのです。

トランスレートされたイメージを AXP システムで実行する速度は，元のイメージを VAX システムで実行する速度と同じくらいになります。しかし，トランスレートされたイメージは， Alpha AXP アーキテクチャを完全に活用するための最適化

コンパイラの恩恵を受けないため，通常，ネイティブな AXP イメージの速度と比較すると，約 25 ～ 40%の速度でしか動作しません。このように性能が低下する主な理由は，データがアラインされていない (unaligned) ためと，複雑な VAX 命令が広範囲にわたって使用されているためです。

イメージのトランスレーションと VEST についての詳しい説明は，第 6.2.2.1 項と『DECmigrate for OpenVMS AXP Systems Translating Images』を参照してください。

ネイティブな AXP イメージとトランスレートされたイメージの混在

1 つのアプリケーションを構成するイメージで，2 種類の移行方式を混在させて使用できます。また，1 つのアプリケーションを，移行の 1 段階として部分的にトランスレートすることもできます。このようにすれば，完全に再コンパイルする前に，アプリケーションを Alpha AXP ハードウェアで実行し，テストできます。1 つのアプリケーション内のネイティブな AXP イメージと，トランスレートされた VAX イメージの相互操作性 (interoperability) についての詳しい説明は，第 4.4.2 項を参照してください。

2.2 アプリケーションの移行に対する DEC のサポート

DEC はお客様がアプリケーションを OpenVMS AXP に移行されるのに役立つように，多彩なサービスを提供しています。

DEC では，お客様のニーズにあったレベルのサービスを各種ご用意しています。

- Orientation Service(Order number: QS-ALPAA-CA)
- Detailed Analysis Service
- Migration Support Service
- Project Planning Service
- Custom Project Service

Orientation Service は， DECdirect (1-800-DIGITAL) でご利用いただけます (このサービスのための注文番号のリストは，米国国内でのみ有効です)。その他のサービスは，お客様のシステムの構成に応じて最適な形でご利用いただけます。

最適なサービスを判断するには， Digital account representative または authorized reseller にご連絡いただくか， 1-800-832-6277(米国国内の場合) または 1-603-884-8990(米国国外の場合) へお電話ください。

2.2.1 Orientarion Service

Orientation Service は， OpenVMS VAX システムから OpenVMS AXP システムへのアプリケーションの移行に含まれる問題を理解するのに役立ちます。このパッケージは， Alpha AXP へ移行されるすべてのお客様にお勧めします。

2.2.2 Detailed Analysis Service

Detailed Analysis Service は， OpenVMS VAX システムから OpenVMS AXP システムへアプリケーションを移行するために，よりお客様にあった計画をたてる方法を提供します。 DEC のコンサルタントがアプリケーションの詳しい調査を行い，移行の実行やスケジュールに問題になりそうな事柄をあげ，詳細な移行評価計画を作成します。このサービスは，移行に援助を必要とするすべてのお客様にお勧めします。

2.2.3 Migration Support Service

Migration Support Service は，移行プロジェクトをサポートする DEC の移行専門家が行います。このサービスは，必要なレベルのオンサイト・テクニカル・コンサルティングを含みます。

2.2.4 Project Planning Service

Project Planning Service は， DEC のコンサルタントが行います。以下のサービスを提供します。

- 移行プロジェクトの大きさや範囲を理解するのに必要な情報
- 必要に応じた詳細な移行方法を開発するために必要な情報
- 詳細な移行プロジェクトの計画

2.2.5 Custom Project Service

Custom Project Service は，トータルな移行を提供します。このサービスは，お持ちのアプリケーションすべての移行を， DEC へ依頼されるお客様にお勧めします。

2.2.6 Business Partner Development Assistance Centers

Business Partner Development Assistance (BPDA) センタは，表 2–1 に示すように世界的な移行サービスを提供します。移行の専門家がセンタに置かれ，その移行サービスで DEC のビジネス・パートナーを支援します。

移行サービスの詳細については， Digital account representative または DEC 認定の販売店にご連絡いただくか， 1-800-832-6277 または 1-603-884-8990 にお問い合わせください。

表 2–1 BPDA センタの所在地

Europe	United States	Asia
Basingstoke	Dtroit	Hong Kong
Munich	Marlboro	Tokyo
Paris	Palo Alto	

2.3 移行トレーニング

Digital Customer Training は，サードパーティのアプリケーション開発者やエンド・ユーザの移行トレーニングのために，いくつかのセミナーやコースを設けています。以下のリストの最初にあるコースは技術または MIS の管理者向けです。それ以外のコースはある程度経験のある OpenVMS VAX プログラマ向けに設定されています。

- Alpha AXP Planning Seminar—2 日間， EY-L570E-SO
- Migrating HLL Applications to OpenVMS Alpha AXP—3 日間， EY-L577E-LO
- Migrating MACRO-32 Applications to OpenVMS AXP—2 日間， EY-L578E-LO

米国内におけるスケジュールや登録情報については， 1-800-332-5656 へお問い合わせください。その他の地域は，最寄りの Digital account representative または DEC 認定の販売店へお問い合わせください。

3

アプリケーションの評価： アプリケーション・モジュールの調査

アプリケーションの評価を行えば，どのような作業が必要であるかを判断でき，移行計画を作成できます。つまり，次の質問に対する解答が得られます。

- どのようにアプリケーションを移行するか
- 移行のために必要な作業量，時間，およびコストはどの程度か
- DEC サービス (Alpha AXP Resource Center) を使用するかどうか

評価のプロセスは，次の 3 つの段階に分けることができます。

1. 一般的なモジュールの確認と，他のソフトウェアに依存する部分の確認
2. 移行に影響するコーディングの様式を判断するためのソース分析
3. 移行方式の選択，つまり，ソース・コードから再構築するのか，トランスレートするのか

これらの各段階の評価を終了すれば，効果的な移行計画を作成するのに必要な情報を入手できます。

移行のためのアプリケーションの評価の第 1 段階は，何を移行するかを正確に判断することです。この段階では，アプリケーション自体を評価するだけでなく，アプリケーションを正しく実行するために，何が必要であるかも判断しなければなりません。アプリケーションの評価を開始するには，次のことを確認してください。

- アプリケーションの各モジュール
  - メイン・プログラムのソース・モジュール
  - 共有可能イメージ
  - オブジェクト・モジュール

  - ライブラリ (オブジェクト・モジュール，共有可能イメージ，テキスト，またはマクロ)
  - データ・ファイルとデータベース
  - メッセージ・ファイル
- アプリケーションが依存する他のソフトウェア
  - ランタイム・ライブラリ
  - DEC レイヤード・プロダクト
  - サード・パーティ・プロダクト

  他のコードへの依存関係を調べる場合には， VEST と/DEPENDENCY 修飾子を使用してください。 VEST/DEPENDENCY は，アプリケーションが依存している実行可能イメージや共有可能イメージを示します。たとえば，ランタイム・ライブラリやシステム・サービス，その他のアプリケーションを識別します。 VEST/DEPENDENCY の使い方についての詳しい説明は，『DECmigrate for OpenVMS AXP Systems Translating Images』を参照してください。
- 必要な操作環境
  - システム属性

    アプリケーションを実行および管理するために，どのような種類のシステムが必要か。たとえば，必要なメモリ・サイズ，必要なディスク空間などです。
  - ビルド・プロシージャ

    このプロシージャは， Code Management System (CMS) や Module Management System (MMS) などの Digital tools を必要とします。
  - テスト・スーツ

    移行したアプリケーションが正しく動作するかどうかを確認し，その性能を評価するために，テストが必要です。

これらの多くは，すでに OpenVMS AXP に移行されています。たとえば，次のソフトウェアやライブラリはすでに移行されています。

- OpenVMS とともに提供される DEC ソフトウェア
  - RTL(ランタイム・ライブラリ)
  - 他の共有可能ライブラリ (shareable library)。たとえば，呼び出し可能なユーティリティ・ルーチンやアプリケーション・ライブラリ・ルーチンを提供するライブラリなどです。
- DEC レイヤード・プロダクト
  - コンパイラとコンパイラ RTL
  - データベース・マネージャ
  - ネットワーク環境
- サード・パーティ・プロダクト

  現在多くのサード・パーティ・アプリケーションが， OpenVMS AXP で実行可能です。各アプリケーションが移行されているかどうかかについては，各アプリケーションの提供業者にお問い合わせください。

ビルド・プロシージャとテストも含めて，アプリケーションと開発環境を移行する作業は，お客様が実行しなければなりません。

アプリケーションの各モジュールを評価した後，各モジュールとイメージの詳しい評価が必要になります。第 4 章を参照してください。

4

移行方法の選択

アプリケーションのモジュールを調査した後，アプリケーションの各部分を移行する方法を決定しなければなりません。つまり，再コンパイルと再リンクを実行するのか，トランスレートするのかを判断しなければなりません。大部分のアプリケーションは再コンパイルし，再リンクするだけで移行できます。アプリケーションがユーザ・モードだけで実行され，標準的な高級言語で作成されている場合には，おそらく再コンパイルと再リンクだけで十分です。主な例外については，第 4.2 節を参照してください。

この章では，移行のために追加作業が必要となる一部のアプリケーションで移行方法をどのように選択すればよいかについて説明します。この判断を下すには，アプリケーションの各部分でどの方法が可能であるかということと，各方法でどれだけの作業量が必要となるかということを，知っておかなければなりません。

注意

この章では，アプリケーションを移行するにあたって，可能な限り再コンパイル，再リンクを行い，イメージのトランスレーションについては，再コンパイル，再リンクできない部分に用いるか，移行の過程での一時的な処理のみに用いると仮定しています。

この後の節では，移行方法を選択するプロセスについて，その概要を説明します。このプロセスでは，次のステップを踏んでください。

1. 2 種類の移行方法のどちらを使用できるかを判断する

   ほとんどの場合，プログラムを再コンパイルおよび再リンクするか， VAX イメージをトランスレートすることができます。どちらか一方の移行方法だけしか使用できないケースについては，第 4.1 節を参照してください。

2. 再コンパイルに影響を与える可能性のあるアーキテクチャへの依存部分を識別する

   アプリケーションが全般的には再コンパイルに適している場合でも，Alpha AXP アーキテクチャと互換性のない VAX アーキテクチャの機能に依存するコードが含まれている可能性があります。

   第 4.2 節では，これらの依存性について説明し，このような固有の機能に依存する部分を識別し，その問題を解決するのに必要な作業の種類と作業量を見積もるのに必要な情報を示します。

   第 4.3 節では，アプリケーションの評価で発生した疑問点に答えるのに役立つツールと方法を説明します。

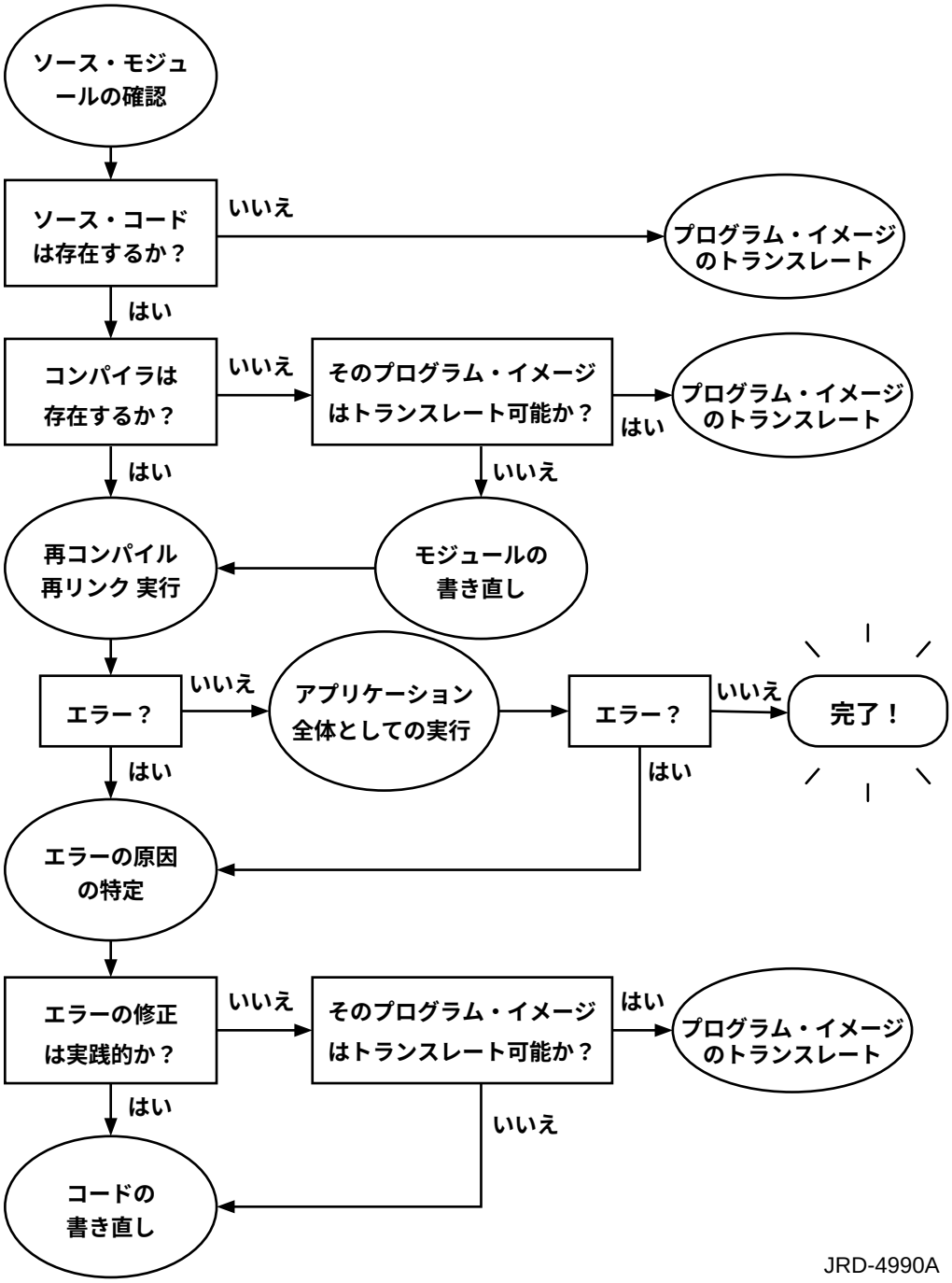
3. 再コンパイルするのか，トランスレートするのかを決定する

   アプリケーションを評価した後，どの移行方法を使用するのかを決定しなければなりません。第 4.4 節では，各方法の利点と欠点のバランスをとることにより，どちらの方法を使用するのかを判断する処理について説明します。

   プログラムを再コンパイルおよび再リンクできない場合や，VAX イメージが VAX アーキテクチャ固有の機能を使用している場合には，そのイメージをトランスレートしなければなりません。第 4.4.1 項では，トランスレートされたイメージの互換性と性能を向上するための方法を説明しています。

アプリケーションの評価のプロセスは，図 4–1 のように表わすことができます。DEC では，この図に示されたアプリケーションの評価に役立つツールを提供しています。これらのツールは，移行プロセスの各段階の説明で，必要に応じてふれます。

図 4–1 プログラム・イメージの移行



JRD-4990A

4.1 どの移行方法が可能か?

ほとんどのアプリケーションは，ソース・コードの再コンパイルおよび再リンクと，イメージのトランスレートのどちらの方法を用いても移行が可能です。しかし，アプリケーションの設計に応じて，次に示すように， 2 種類の移行方法のどちらか一方だけしか使用できないことがあります。

- 再コンパイルできないプログラム

  次のイメージはトランスレートしなければなりません。

  – OpenVMS AXP コンパイラがまだ提供されていないプログラミング言語で作成されたソフトウェア

    Ada， BASIC， C， C++， COBOL， FORTRAN， Pascal， PL/I，およびVAX MACROのネイティブ・コンパイラは，OpenVMS AXPバージョン 6.1 で提供されます。 LISP を含む他の言語は，将来のリリースで提供されます。

  – ソース・コードを入手できない実行可能イメージと共有可能イメージ

  – H 浮動小数点または 56 ビットの D 浮動小数点データを必要とするプログラム

- トランスレートできないイメージ

  次のイメージのソース・コードは再コンパイルおよび再リンクしなければなりません (変更も必要となる可能性があります)。

  – OpenVMS VAX バージョン 4.0 以前に作成されたイメージ

  – PDP–11互換モードを使用するイメージ

  – リンカ・オプションでベースを指定したイメージ

  – 現在， Alpha AXP アーキテクチャでサポートされていないコーディング方式を使用しているイメージ。このようなコードとしては，次のコードがあります。

    a. 内部アクセス・モードまたは高い IPL で実行されるコード
       (たとえば，デバイス・ドライバなど)

    b. システム空間のアドレスを直接参照するコード

c. 解説書に説明されていないシステム・サービスを直接参照するコード

d. スレッド・コードを使用するコード，たとえば，スタックを切り換えるコード

e. VAX ベクタ命令を使用するコード

f. 特権付き VAX 命令を使用するコード

g. リターン・アドレスを調べたり，変更するコードや，プログラム・カウンタ (PC) をもとに，他の判断を下すコード

h. 512 バイトのサイズのメモリ・ページに依存するため，正確なアクセス違反動作に依存するコード

i. ページ境界以外の境界に，グローバル・セクションをアラインするコード (たとえば， 512 バイトのメモリ・ページ・サイズに依存するコード)

j. 大部分の VAX P0 空間または P1 空間を使用するコード，トランスレートされたイメージの実行時サポート・ルーチンが使用する空間に敏感に反応するコード

VEST は次のようなイメージもトランスレートできますが，トランスレートされたイメージの実行時の性能は， TIE が解釈しなければならない VAX コードの量が多いために低下します。

– TIE によって実行時に作成されるコードを除き，それ自体を変更する VAX コードまたは実行時に作成される VAX コードを含むイメージ。

– 命令ストリームを調べるコードを含むイメージ。ただし， TIE が実行時にこのようなコードを解釈する場合を除きます。

どのイメージをトランスレートできるかについての詳しい説明は，『DECmigrate for OpenVMS AXP Systems Translating Images』を参照してください。

4.2 再コンパイルに影響を与えるコーディングの様式

多くのアプリケーション，特に標準のコーディング様式のみを使用しているアプリケーションや，移植性 (portability) を念頭において作成されているアプリケーションはほとんど問題なく， OpenVMS VAX から OpenVMS AXP に移行できます。しかし， VAX 固有の機能に依存し，その機能が Alpha AXP アーキテクチャと互換性のないようなアプリケーションを再コンパイルする場合には，ソース・コードを変更しなければなりません。次の例は，典型的な互換性のない機能を示しています。

- VAX システムで高い性能を実現したり， VAX アーキテクチャ固有の機能を利用するために使用されているVAX MACROアセンブリ言語
- 特権付きコード
- VAX アーキテクチャ固有の機能

これらの互換性のない機能がアプリケーションでまったく使用されていない場合には，この章のこの後の部分を読む必要はありません。

4.2.1 VAX MACROアセンブリ言語

AXP システムでは，VAX MACROはアセンブリ言語ではなく，コンパイラの 1 つでしかありません。しかし，高級言語のための Alpha AXP コンパイラと異なり，VAX MACRO–32 Compiler for OpenVMS AXP は常に高度に最適化されたコードを生成するわけではありません。したがって，VAX MACRO–32 Compiler for OpenVMS AXP は移行の補助手段としてのみ使用するようにし，新しいコードを作成する場合は使用しないでください。

VAX システムでアセンブリ言語を使用しなければならなかった多くの理由は，次のように， AXP システムでは解消されました。

- RISC プロセッサでは，アセンブリ言語を使用しても性能が向上するわけではありません。 Alpha AXP コンパイラ・セットに含まれているコンパイラなどの RISC コンパイラは，プログラマが手作業で最適化するより効率よく，もっと容易にアーキテクチャやインプリメントの特徴を利用して，最適化されたコードを生成できます。

- 新しいシステム・サービスは，これまでアセンブリ言語を必要としていた一部の機能を実行できます。

MACRO コードの移行についての詳しい説明は，『Migrating to an OpenVMS AXP System: Porting VAX MACRO Code』を参照してください。

4.2.2 特権付きコード

内部アクセス・モード (カーネル，エグゼクティブ，またはスーパーバイザ・モード) で実行されたり，システム空間を参照する VAX コードは， VAX アーキテクチャに依存したコーディング様式を使用している可能性が高く，また， OpenVMS AXP には存在しない VAX データ呼び出しを参照している可能性があります。このようなコードは，変更しなければ AXP システムに移行できません。これらのプログラムは再コーディング，再コンパイル，および再リンクが必要です。

この種類に分類されるコードは次のとおりです。

- ユーザ作成システム・サービスや他の特権付き共有可能イメージ (privileged shareable image)

  詳しくは，『OpenVMS Programming Concepts Manual』と『OpenVMS Linker Utility Manual』を参照してください。

- DEC 以外から提供されたデバイス・ドライバとパフォーマンス・モニタ
- 特殊な特権を使用するコード。たとえば，$CMEXEC または$CMKRNL システム・サービスを使用するコードや， PFNMAP オプションを選択して$CRMPSC システム・サービスを使用するコード

  詳しくは，『OpenVMS AXP オペレーティング・システムへの移行：再コンパイルと再リンク』を参照してください。

- 次のように， OpenVMS の内部ルーチンまたはデータを使用するコード
  - システム・アドレス空間をアクセスするためにシステム・シンボル・テーブル (SYS.STB) に対してリンクするコード
  - SYS$LIBRARY:LIB を用いてコンパイルするコード

OpenVMS エグゼクティブを参照する内部モード・コードを移行する場合には，DEC サービス (Alpha AXP Resource Center) にご連絡ください。

4.2.3 VAX アーキテクチャ固有の特徴

高い性能を実現するために，Alpha AXP アーキテクチャは VAX アーキテクチャと大きく異なっています。したがって，VAX アーキテクチャ固有の特徴を利用してコードを作成することに慣れているソフトウェア開発者は，AXP システムに正しく移行するために，コードで使用しているアーキテクチャ固有の特徴を理解しておかなければなりません。

この後の節では，一般的なアーキテクチャ固有の特徴と，それらの特徴を識別する方法および対処方法について簡単に説明します。これらのアーキテクチャ固有の特徴の識別方法と対処方法についての詳しい説明は，『OpenVMS AXP オペレーティング・システムへの移行：再コンパイルと再リンク』を参照してください。

4.2.3.1 性能に関する問題

VAX アーキテクチャと Alpha AXP アーキテクチャの相違点のうち，次の 2 つは VAX アプリケーションを OpenVMS AXP で実行不可能にするものではありませんが，性能に大きな影響を与えます。

- データ・アラインメント (alignment)
- データ型の選択

4.2.3.1.1 データ・アラインメント

データ・アドレスがデータ・サイズ (バイト数) の整数倍である場合には，データは**自然なアラインメント**になります。たとえば，ロングワードは 4 の倍数であるアドレスに自然なアラインメントになり，クォドワードは 8 の倍数であるアドレスに自然なアラインメントになります。構造体の場合も，すべてのメンバが自然なアラインメントになっているときは，その構造体も自然なアラインメントになります。

メモリ内で自然なアラインメントでないデータをアクセスすると，VAX システムでも AXP システムでも性能が大幅に低下します。VAX システムでは，大部分の言語は省略時の設定により，データを次の使用可能なバイト境界にアラインするた

め，VAX アーキテクチャでは，アラインされていないデータを参照したときに，性能の低下を最低限に抑えるためのハードウェア・サポートが準備されています。

しかし，AXP システムでは，各データを自然なアラインメントにすることが省略時の設定であるため，Alpha AXP は他の典型的な RISC アーキテクチャと同様に，アラインされていないデータを使用することによって発生する性能の低下を最低限に抑えるためのハードウェア・サポートを準備していません。この結果，AXP システムで自然なアラインメントになっているデータを参照する操作は，アラインされていないデータを参照する操作より 10 ～ 100 倍も速くなります。

AXP コンパイラは，アラインメントに関する大部分の問題を自動的に修正し，修正できない問題には警告を発します。

問題の検出

アラインされていないデータを検出するには，次の方法が有効です。

- 大部分の AXP コンパイラが提供するスイッチを使用する方法。このスイッチを使用すれば，コンパイラはアラインされていないデータのコンパイル時参照を報告できます。たとえば，DEC C および DEC Fortran プログラムの場合には，/WARNING=ALIGNMENT 修飾子を使用してコンパイルします。
- 実行時にアラインされていないデータを検出するために，OpenVMS デバッガまたは DEC PCA (Performance and Coverage Analyzer) を使用する方法。

問題への対処方法

アラインされていないデータに対処するには，次に示す方法を用います。

- データをクォドワード境界にアラインすることにより，性能を最大限に向上する方法。これは，AXP システムが一般にクォドワード**粒度** (granularity)のみをサポートするからです (第 4.2.3.2.2 項を参照)。
- 自然なアラインメントでコンパイルするか，または言語がこの機能を備えていない場合には，自然なアラインメントになるようにデータを移動する方法。データが確実にアラインされるように間隔をあけると，メモリ・サイズが拡大するという問題があります。メモリを節約すると同時に，確実にデータを自然にアラインする (naturally aligned) ための方法として，サイズの大きい変数を最初に宣言する方法があります。

- データ構造内で強制的に自然なアラインメントが実現されるように，高級言語命令を使用する方法。たとえばDEC Cでは，自然なアラインメントが省略時のオプションです。VAX Cの省略時のアラインメントと一致しなければならないデータ構造 (たとえばディスク上のデータ構造など) を定義するには，#PRAGMA NO_MEMBER_ALIGNMENT文を使用します。 DEC Fortran の場合には，省略時の設定により，ローカル変数は自然なアラインメントになります。レコード構造とコモン・ブロックのアラインメントを制御するには，/ALIGN 修飾子を使用します。
- VAX と互換性のある，アラインされていない#PRAGMA NO_MEMBER_ALIGNMENT のようなコンパイラ・スイッチを使用する方法。これらのスイッチを使用すると，機能的には正しいものの，実行速度が遅くなる可能性のある AXP プログラムが作成されます。

注意

自然なアラインメントに変換されたソフトウェアは，同じ VMS クラスタ環境内の VAX システム，またはネットワークによって接続された VAX システムでトランスレートされた他のソフトウェアと互換性がなくなる可能性があります。次の例を参照してください。

- **既存のファイル・フォーマットは，アラインされていないデータを含むレコードを指定する可能性があります。**
- **トランスレートされたイメージは，アラインされていないデータをネイティブ・イメージに渡したり，ネイティブ・イメージからそのようなデータを渡されることを要求する可能性があります。**

このような場合には，アプリケーションのすべての部分が同じデータ型，つまり，アラインされたデータ型またはアラインされていないデータ型を要求するように変更しなければなりません。

データのアラインメントについての詳しい説明は，『OpenVMS AXP オペレーティング・システムへの移行：再コンパイルと再リンク』を参照してください。

4.2.3.1.2 データ型

AXP プロセッサは性能の向上のために，パック 10 進数 (packed decimal) データ型， H 浮動小数点データ型，および完全な精度の D 浮動小数点データ型をソフトウェアによって実現します。

- パック 10 進数

  18 桁のパック 10 進数データは 64 ビットの 2 進数に内部的に変換されます。この結果， COBOL できわめて高い性能を実現できます。

- H 浮動小数点

  AXP コンパイラは H 浮動小数点データをサポートしません。しかし，Translated Image Environment (TIE) はトランスレートされたイメージで H 浮動小数点データをエミュレートによってサポートします。

- D 浮動小数点

  AXP プラットフォームでは， D 浮動小数点データは次の方法で実現されます。

  – G 浮動小数点ハードウェア (D53) を使用する方法。 AXP ハードウェアは D 浮動小数点データ (D53) を処理のために G 浮動小数点に変換します。この結果， D 浮動小数点データを格納した既存のバイナリ・ファイルとの間で，速度およびデータ型の互換性を維持できますが，現在の VAX システムの D 浮動小数点算術演算と比較すると， 3 ビットが失われてしまいます。つまり， VAX システムの D56 では 16 桁の精度で処理されるのに対し， D 浮動小数点データは 15 桁の精度で処理されます。

  – トランスレートされたイメージのためにソフトウェア・エミュレーション (D56) を使用する方法。この方法では，正確な D56 フォーマットの結果が得られますが， D53 や G 浮動小数点より処理速度が遅くなります。

問題への対処方法

データ型に関する問題に対処するには，次の方法を用います。

- 可能な場合には， H 浮動小数点のかわりに G 浮動小数点または IEEE T 浮動小数点を使用してください。これは次の理由によります。

  – どちらのデータ型も10^{-308}～10^{308}の範囲のデータをサポートするため

  – 約 15 桁の精度であるため

- 可能な場合には，パック 10 進数データ型のかわりに整数データ型を使用してください。

Alpha AXP のデータ型についての詳しい説明は，『OpenVMS AXP オペレーティング・システムへの移行：再コンパイルと再リンク』を参照してください。

4.2.3.2 共有データの保護

VAX アーキテクチャと Alpha AXP アーキテクチャの間にはいくつかの相違点があり，これらの相違点は共有データ (shared data) の整合性に影響を与える可能性があります。

4.2.3.2.1 メモリ内のデータの変更

不可分な操作とは，次のような操作です。

- 中間結果または部分的な結果を他のプロセッサや装置から確認できない
- 操作を中断できない (つまり，起動した後，操作は完全に終了するまで継続されます)

OpenVMS AXP では，データをメモリから読み込む操作，メモリ内のデータを変更する操作，およびデータをメモリに格納する操作は，複数の命令に分割され，これらの命令の間で割り込みをかけることができます。この結果，アプリケーションで共有メモリ内のデータを不可分な操作によって変更したい場合には，操作の不可分性 (atomicity) を保証するための作業が必要になります。

次の条件が満足される場合，アプリケーションは操作が不可分に実行されることに依存している可能性があります。

- プロセス内の AST ルーチンがメインライン・コードとデータを共有すること
- プロセスが同じ CPU(つまり，ユニプロセッサ・システム) で実行される別のプロセスと，**書き込み可能なグローバル・セクション**のデータを共有すること
- プロセスが別の CPU(つまり，マルチプロセッサ・システム) で並列に実行される別のプロセスとの間で，書き込み可能なグローバル・セクションのデータを共有すること

問題の検出
不可分性への依存を検出するには，共有変数の使用を再確認し，不可分性を暗黙にまたは明示的に仮定している部分がないかどうかを調べなければなりません。

問題への対処方法
命令の不可分性に関する一般的な問題を解決するには，次の方法を用います。

- 可能な場合には，共有変数を保護するために不可分性を保証する言語構造を使用してください。たとえば， C では VOLATILE 宣言を使用します。
- 不可分性を仮定するのでなく，明示的に**同期**を使用します。
- OpenVMS のロック・サービス (たとえば$ENQ と$DEQ)，並列処理ランタイム・ライブラリ (Parallel Processing Run-time Library, PPL$)・ルーチン，またはライブラリ (LIB$)・ルーチンを使用します。
- AST スレッドとの同期をとるために，メインライン・コードで$SETAST システム・サービスを使用して AST をブロックし，命令が終了した後で AST を再度有効に設定します。

同期についての詳しい説明は，『OpenVMS AXP オペレーティング・システムへの移行：再コンパイルと再リンク』を参照してください。

4.2.3.2.2 クォドワードより小さいデータの読み込みまたは書き込み

粒度という用語は，隣接するメモリ位置に格納されているデータを妨害せずに，不可分な操作として，メモリとの間で読み込みまたは書き込みを実行できるデータ・サイズを示します。 VAX のようにバイト・レベルの粒度を提供するマシンを**バイト粒度マシン**と呼びます。 AXP システムは**クォドワード粒度**のシステムです。[1]

OpenVMS AXP はクォドワード粒度のシステムであるため，共有されるバイト，ワード，またはロングワードに書き込むと，共有データと同じクォドワードに格納されている他のデータを破壊する可能性があります。このような状況は次の場合に発生します。

- プログラムでバイト，ワード，またはロングワードを変更する場合

[1] Alpha AXP アーキテクチャはロングワード粒度もサポートしますが，ロングワード粒度を仮定することは望ましくありません。省略時の設定で，コンパイラはソース・コードがクォドワード未満の粒度に依存しないものと仮定しています。

- 任意のサイズのアラインされていないフィールドが，アラインされたクォドワード境界と交差し，個別に書き込まなければならないバイト，ワード，またはロングワードを作成する場合

注意

不可分性に関する説明 (第 4.2.3.2.1 **項) で説明したデータ共有の種類はすべて，共有データを格納しているクォドワードの残りの部分で，粒度に関する問題を発生させる可能性があります。**

さらに，結果をプロセス空間に戻すような，非同期的に終了するライブラリ関数，または非同期システム・サービスをプロセスが起動した場合には，戻されたデータを格納するクォドワードで，粒度に関する問題が発生する可能性があります。たとえば，次の操作では，粒度に関する問題が発生する可能性があります。

- **状態ブロックに書き込む非同期システム・サービス**
- **プロセス・バッファに書き込む入出力操作**
- **ダイレクト・メモリ・アクセス** (DMA)**・コントローラがプロセス・バッファに書き込みを実行する入出力操作**

問題の検出

バイト粒度，ワード粒度，またはロングワード粒度の使用を検出するには，次の方法を用います。

- 意識的に共有されるデータ (AST とメイン・スレッドの間またはプロセス間で) を検出します。共有データが，書き込まれる可能性のある他のデータと同じクォドワードを使用するかどうかを確認します。
- 非同期システム・サービス，または非同期的に終了するライブラリ呼び出しから戻されたデータを確認します。そのデータが，他のプロセスによって書き込まれた他のデータと同じクォドワードを使用するかどうかを確認します。
- 非同期的に戻されたデータを格納する装置からデータを受信する入出力バッファを調べます。バッファの先頭と末尾が，他のプロセスによって書き込まれたデータと同じクォドワードを使用するかどうかを確認します。

問題への対処方法
クォドワード未満の粒度の使用に対処するには，次の方法を用います。

- 共有データを固有のクォドワードに格納します。
- 入出力バッファの先頭をクォドワード境界にそろえ，バッファの後に続くデータを次のクォドワードに移動します。
- 問題の原因がシステムと共有されるデータでない場合には，高いレベルの同期メカニズムを使用して，意識的に共有されるデータとバックグラウンド・データの両方を同じクォドワードでインターロックします。

AXP コンパイラは，省略時の設定ではバイト粒度，ワード粒度，またはロングワード粒度をサポートしませんが，現在のコードとの互換性を維持するために，バイト粒度，ワード粒度，アラインされていないロングワード粒度，およびアラインされていないクォドワード粒度を指定できます。詳しくは，各コンパイラの解説書を参照してください。

読み込み/書き込みの粒度についての詳しい説明は，『OpenVMS AXP オペレーティング・システムへの移行：再コンパイルと再リンク』を参照してください。

4.2.3.2.3 ページ・サイズに関する検討

ページ・サイズは，メモリ管理ルーチンとシステム・サービスが割り当てる仮想メモリのサイズを管理します。また，メモリ内のコードとデータに保護を割り当てる基礎にもなります。

OpenVMS VAX オペレーティング・システムは 512 バイトの倍数でメモリを割り当てます。しかし，全体的なシステム性能を向上するために， AXP システムでは，各ハードウェア・プラットフォームに応じて， 8KB ～ 64KB の大きなページ・サイズを採用しています。

ページ・サイズは，ワーキング・セット・クォータなど，システム資源を管理するときの基礎的な要素です。さらに， VAX システムでのメモリ保護は， 512 バイトの各メモリ領域ごとに設定できます。 AXP システムでは，メモリ保護の最小単位

はシステムのページ・サイズに応じて，VAX システムの場合よりはるかに大きくなります。

注意

ページ・サイズを大きく変更した結果，影響を受けるのは，512 バイトのページ・サイズに明示的に依存しているアプリケーションだけです。たとえば，次のアプリケーションが考えられます。

- **次の目的で** "512"**を使用しているアプリケーション**
  - **メモリの使用状況を計算するため**
  - **必要なページ・テーブルのサイズを計算するため**
- 512 **バイト単位で保護を変更するアプリケーション**
- **システム・サービス** Create and Map Section ($CRMPSC) **を使用して，ファイルをプロセス空間内の特定の位置にマッピングするアプリケーション (たとえば，使用可能なメモリが制限されているときにメモリを再利用するため)**

問題の検出

VAX ページ・サイズを使用している箇所を検出するには，512 バイトひとかたまりで仮想メモリを操作するコードを識別するか，またはメモリ保護の最小単位として 512 バイトを使用しているコードを識別します。コードから 10 進数の 511，512，または 513，16 進数の 1FF，200，または 201 などの数値を検索してください。

問題への対処方法

VAX と AXP のページ・サイズの相違によって発生する問題に対処するには，次のような方法を使用できます。

- ハードコードされたページ・サイズの参照をシンボリック値に変更します (実行時に$GETSYI を呼び出すことにより割り当てられます)。
- ページ・サイズとディスク (ファイル)・ブロック・サイズが等しいと仮定しているコードを再評価します。AXP システムでは，この仮定は正しくありません。

- メモリ管理関連システム・サービス (たとえば，$CRMPSC，$MGBLSC) を使用して，ファイルを固定のページ・サイズ依存アドレス空間 (グローバル・セクション) にマッピングできるものと仮定しないでください。このような場合には，$EXPREG システム・サービスを使用してください。

ページ・サイズについての詳しい説明は，『OpenVMS AXP オペレーティング・システムへの移行：再コンパイルと再リンク』を参照してください。

4.2.3.2.4 マルチプロセッサ・システムでの読み込み/書き込み操作の順序

VAX アーキテクチャでは，マルチプロセシング・システムのプロセッサが複数のデータをメモリに書き込む場合，これらは指定した順にメモリに書き込まれます。このように書き込みの順序が定義されているため，書き込み操作はプログラムと入出力装置で指定した順に他の CPU からも確認することができます。

このように，指定した順に書き込み結果を他の CPU からも確認できることを保証することは，システムが実現できる性能の最適化を制限してしまいます。また，キャッシュがより複雑になり，キャッシュ性能の最適化も制限されます。

全体のシステム性能を向上するために， AXP システムをはじめ， RISC システムでは，メモリへの読み込みと書き込みの順序を変更できます。したがって，メモリへの書き込み結果をシステム内の他の CPU から確認すると，その順序は実際に書き込まれた順序と異なる可能性があります。

注意

この節の説明はマルチプロセッサ・システムを対象にしています。ユニプロセッサ・システムでは，メモリ・アクセスはすべて，プログラムで要求した順に終了します。

問題の検出

マルチプロセッサ・システムで実行される可能性のあるアプリケーションで，読み込み/書き込みの順序に依存している部分を検出するには，データが書き込まれる順序に依存するアルゴリズムを識別しなければなりません。たとえば，同期をとるためにフラグ受け渡しプロトコルを使用している箇所を調べなければなりません。

問題への対処方法

読み込み/書き込み操作の順序に関する問題に対処するには，次の方法を用います。

- フラグ受け渡しプロトコルのかわりに，同期をとるためにシステムで提供されるルーチンを使用します。たとえば，並列処理ランタイム・ライブラリ (PPL$)・ルーチンや OpenVMS ロック・システム・サービス ($ENQ，$DEQ) を使用します。
- Alpha AXP アーキテクチャでは，メモリ・バリア命令が準備されており，この命令を使用すると，ハードウェアはバリアの前のすべてのメモリ読み込みと書き込みを終了した後で，バリアの後の読み込みと書き込みを実行します。OpenVMS AXP の一部の言語では，この命令を挿入する方法が準備されていますが，この命令を使用すると性能が低下します。

同期についての詳しい説明は，『OpenVMS AXP オペレーティング・システムへの移行：再コンパイルと再リンク』を参照してください。

4.2.3.3 算術演算例外の報告の即時性

AXP (およびベクタ VAX) システムでは，スカラ VAX システムと異なる方法で例外を取り扱います。スカラ VAX システムでは，「正確な例外報告 (precise exception reporting)」を使用します。つまり，命令を実行した結果，例外が発生した場合には，報告される**プログラム・カウンタ** (PC)は，その例外の原因となった命令のアドレスです。後続の命令は実行されておらず，プロセスのコンテキストに影響を与えないため，条件ハンドラは例外の原因を修正し，障害が発生した命令から，またはその次の命令からプログラムの実行を再開できます。

パイプライン環境で可能な最高の性能を実現するために，ベクタ VAX システムと AXP システムでは「あいまいな例外報告 (imprecise exception reporting)」を採用しています。つまり，例外ハンドラが報告する PC は，算術演算例外の原因となった命令の PC であるとは限りません。さらに，例外が報告される前に後続の命令が終了している可能性があります。

実際には，算術演算の例外となった特定の命令を知らなければならないプログラムはほとんどありません。通常，算術演算例外が発生した場合には，プログラムは次のいずれかの操作を実行します。

- 例外を記録し，処理を継続します。
- エラー・メッセージを印刷し，サブルーチンまたはプログラムを強制終了します。
- サブルーチン全体を再起動し，オーバーフローまたはアンダーフローを防止するために，異なるアルゴリズムを使用してデータを再計算します。

VAX プログラムが算術演算例外を検出したときに，これらのいずれかの動作を実行する場合には，そのプログラムは，あいまいな例外報告を採用している RISC システムに移行しても，まったく影響を受けません。

注意

あいまいな例外報告は算術演算例外にだけ適用されます。フォルトやトラップなどの他の例外の場合には，OpenVMS AXP は正確に例外を報告し，例外の原因となった特定の命令を報告します。

問題の検出

正確な例外報告に依存している部分を検出するには，算術演算例外ハンドラが存在するかどうかを確認しなければなりません。その後，ハンドラが正確なプログラム・カウンタ (PC) を必要としているのか，または例外の原因となったプロシージャを知るだけでよいのかを確認します。

問題への対処方法

正確な例外処理への依存に対処するには，算術演算例外の原因となった正確な命令を知ることが必要な算術演算条件ハンドラを作成しないようにします。

例外処理についての詳しい説明は，『OpenVMS AXP オペレーティング・システムへの移行：再コンパイルと再リンク』を参照してください。

互換性を維持するために，大部分の AXP コンパイラは，プログラマがコンパイル時に正確な例外報告が必要であるかどうかを指定できるように，コンパイラ・オプションを準備しています。しかし，正確な例外報告はアプリケーションの性能を著しく低下させます。アプリケーションで特定の操作だけが正確な例外報告を必要とする場合には，アプリケーションの中でこれらの操作を含む部分をコンパイルする場合にだけ，このオプションを使用します。詳しくは，各コンパイラの解説書を参照してください。

4.2.3.4 VAX プロシージャ呼び出し規則への明示的な依存

OpenVMS の呼び出し規則 (calling standard) では，VAX プログラムと AXP プログラムとで，呼び出し規則が大きく異なります。VAX プロシージャ呼び出し規則の詳細な部分に明示的に依存するアプリケーション・プログラムを，AXP システムでネイティブ・コードとして実行する場合は，変更が必要です。たとえば，次のようなコードは変更する必要があります。

- 引数ポインタ (AP) を基準にして引数の位置を判断するコード

  しかし，多くの場合，VAX MACRO–32 Compiler for OpenVMS AXP はこの問題を自動的に補正します。

- スタックのリターン・アドレスを変更するコード
- コール・フレームの内容を解釈するコード

VAX コンパイラも AXP コンパイラも，プロシージャ引数をアクセスするための方法を準備しています。コードでこれらの標準メカニズムを使用している場合には，単に再コンパイルするだけで，AXP システムで正しく実行できるようになります。コードでこれらのメカニズムを使用していない場合には，これらのメカニズムを使用するように変更しなければなりません。これらの標準メカニズムについての説明は，『OpenVMS AXP オペレーティング・システムへの移行：再コンパイルと再リンク』を参照してください。

トランスレートされたコードは，VAX プロシージャ呼び出しの動作をエミュレートします。ここに示したコードを含むイメージは，トランスレートすることにより，AXP システムで正しく実行できるようになります。

4.2.3.5 VAX データ処理メカニズムへの明示的な依存

例外処理の方法は，VAX システムと AXP システムとで異なります。算術演算例外が，VAX システムと AXP システムでディスパッチされる方法の違いについては，『OpenVMS AXP オペレーティング・システムへの移行：再コンパイルと再リンク』を参照してください。この節では主に，コードが動的に条件ハンドラを設定するメカニズムと，条件ハンドラが例外状態をアクセスするメカニズムについて説明します。

4.2.3.5.1 動的な条件ハンドラの設定

VAX システムには，アプリケーションが実行時に動的に条件ハンドラを設定できる方法が数多く準備されています。 OpenVMS の呼び出し規則では，条件ハンドラのアドレスをプロシージャのコール・フレームの先頭に書き込むことによって，そのプロシージャの中で起きた例外に対処する条件ハンドラとして設定できます。これにより， VAX 上のプログラムが条件ハンドラの設定を容易に行えるわけですが， AXP プロシージャのプロシージャ・ディスクリプタには，これに対応する場所は確保されていません。

たとえば，VAX MACROプログラムは次の一連の命令を使用して，条件ハンドラのアドレスをコール・フレームに移動できます。

```
MOVAB    HANDLER,(FP)
```

VAX MACRO–32 Compiler for OpenVMS AXP はこの文を解析し，条件ハンドラを設定するための適切な AXP アセンブリ言語コードを作成します。詳しくは『Migrating to an OpenVMS AXP System: Porting VAX MACRO Code』を参照してください。

注意

VAX システムでは，ランタイム・ライブラリ・ルーチン LIB$ESTABLISH **と，その逆の操作をする** LIB$REVERT **を使用すれば，アプリケーションは現在のフレームに対して条件ハンドラを設定したり，解除することができます。これらのルーチンは，** AXP **システムには存在しません。しかし，コンパイラはこれらの呼び出しを正しく取り扱うことができます (たとえば，** FORTRAN **の組み込み機能によって)。詳しくはアプリケーションに関連するコンパイラの解説書を参照してください。**

トランスレートされたコードは，条件ハンドラを動的に設定するための VAX メカニズムをエミュレートします。

特定の AXP コンパイラ (およびクロス・コンパイラ) は，動的な条件ハンドラを設定するためのメカニズムを準備しています。

条件ハンドラについての詳しい説明は，『OpenVMS AXP オペレーティング・システムへの移行：再コンパイルと再リンク』を参照してください。

4.2.3.5.2 シグナル・アレイとメカニズム・アレイ内のデータのアクセス

VAX システムと AXP システムはどちらも，例外処理で例外時のプロセッサ・ステータス，例外時のプログラム・カウンタ (PC)，シグナル・アレイ，およびメカニズム・アレイをスタックに格納します。

シグナル・アレイとメカニズム・アレイの内容およびフィールドの長さは， VAX システムと AXP システムとで異なります。シグナル・アレイ，またはメカニズム・アレイ内のデータをアクセスする条件ハンドラが，両方のシステムで正しく動作するためには，オフセットをハードコードするのではなく，適切な CHF$シンボルを使用しなければなりません。適切な CHF$シンボルについての説明は，『OpenVMS AXP オペレーティング・システムへの移行：再コンパイルと再リンク』を参照してください。

注意

条件ハンドラは，ハードコードされた引数ポインタ (AP) からのオフセットを使用して，メカニズム・アレイ内の情報を正しく検索することができません。

4.2.3.6 VAX AST パラメータ・リストへの明示的な依存

OpenVMS VAX は， 5 つのロングワード引数を AST サービス・ルーチンに渡します。VAX MACROで作成された AST サービス・ルーチンは，引数ポインタ (AP) からのオフセットを使用して，この情報をアクセスします。 OpenVMS VAX では， AST サービス・ルーチンは，保存されているレジスタやリターン・プログラム・カウンタ (PC) も含めて，これらの引数を変更できます。これらの変更は，AST ルーチンが終了した後，割り込まれたプログラムに影響を与える可能性があります。

AXP システムの AST パラメータ・リストも 5 つのパラメータで構成されますが，AST プロシージャを対象にした引数は AST パラメータだけです。他の引数も存在しますが，これらの引数は， AST プロシージャが終了した後は使用されません。したがって，これらの引数を変更しても， AST 終了時に再開される操作のスレッ

ドに影響を与えないため，このような影響に依存するプログラムを AXP システムで実行するには，従来の引数受け渡しメカニズムを使用するようにプログラムを変更しなければなりません。 AST サービス・ルーチンの移行についての詳しい説明は，『OpenVMS AXP オペレーティング・システムへの移行：再コンパイルと再リンク』を参照してください。

4.2.3.7 VAX 命令の形式と動作への明示的な依存

VAX MACRO命令の実行動作や VAX 命令のバイナリ・エンコーディングに特に依存するプログラムは， AXP システムでネイティブ・コードとして実行するために再コンパイルまたは再リンクする前に，変更しなければなりません。

たとえば，次のコーディング方式は AXP システムでは機能しません。

- VAX MACROで VAX 命令ブロックをプログラム・データ領域に組み込み， PC を変更してこのコード・ブロックに制御を渡すようなコーディング様式
- プロセッサ・ステータス・ワード (PSW) の条件コードや他の情報を調べるようなコーディング様式

VAX MACROコードの移行についての詳しい説明は，『Migrating to an OpenVMS AXP System: Porting VAX MACRO Code』を参照してください。

4.2.3.8 VAX 命令の実行時作成

実行時に VAX 命令を作成し，実行する従来の方法は，ネイティブ AXP モードでは機能しません。

実行時に作成される VAX 命令は，トランスレートされたイメージでエミュレーションによって実行されます。

特定の VAX 命令を実行時に作成するコードについての詳しい説明は，『Migrating to an OpenVMS AXP System: Porting VAX MACRO Code』を参照してください。

4.3 VAX システムと AXP システムの間で互換性が維持されない部分の識別

アプリケーションの各モジュールで，アーキテクチャ間の互換性が維持されない部分を識別するには，まず AXP コンパイラを使用して，モジュールのテスト・コンパイルを実行します。このときに役立つコンパイラ・スイッチについての説明は，各コンパイラの解説書を参照してください。

多くのモジュールはエラーなしにコンパイルし，実行できます。しかし，エラーが発生した場合には，モジュールを変更しなければなりません。

注意

モジュールを単独でエラーなしに実行できる場合でも，アプリケーションの他の部分と組み合わせて実行したときに，同期に関する問題が発生する可能性があります。

再コンパイルおよび再リンクした後，モジュールを正しく実行できない場合には，次の方法を使用して， AXP システムでプログラムを正しく実行するために，どの部分を変更しなければならないかを評価します。

- ソース・コードの確認

  移行プロセスのこの段階でコードを再確認すれば，多くの問題を前もって解決でき，この後の移行段階で多くの時間と作業を節約できます。コードを確認するには，付録 A に示したチェックリストを使用し，さらに『OpenVMS AXP オペレーティング・システムへの移行：再コンパイルと再リンク』に示したガイドラインに従ってください。移行に関するこれらの問題点は，第 4.2 節にまとめられています。

  アプリケーション全体のコードを直接確認することが実際に不可能な場合には，自動的な検索処理を使用すると便利です。たとえば， DCL の SEARCH コマンドとエディタを組み合わせて使用して，アーキテクチャ上の問題点を検出することができます。

- 初期のテスト・コンパイルでコンパイラが出したメッセージの確認

コンパイラは次の情報を提供します。

– VAX コンパイラと AXP コンパイラの違い

– データ・アライメント

特定のコンパイラは，VAX アーキテクチャと Alpha AXP アーキテクチャのその他の違いも検出します。

- VEST の使用によるイメージの分析

  プログラムを再コンパイルおよび再リンクする場合でも，分析ツールとして VEST を使用できます。この結果，AXP システムでプログラムをもっとも効率よく実行するのに必要な変更操作に関して，役立つ多くの情報が得られます。たとえば，VEST は次の問題を検出するのに役立ちます。

  – 静的にアラインされていないデータ (データ構造内のアラインされていないフィールドも含めたデータ宣言) とアラインされていないスタック操作

  – 浮動小数点データ型の参照 (H 浮動小数点と D 浮動小数点)

  – パック 10 進数の参照

  – 特権付きコード

  – 標準的でないコーディング様式

    a. システム・サービスを使用しない，OpenVMS データまたはコードの参照

    b. 初期化されていない変数

  – 同期に関するある種の問題，たとえば，インターロック命令のマルチプロセス使用

  ただし，VEST は一部の問題を検出できません。次の例を参照してください。

  – アラインされていない変数 (動的に作成されたデータ構造内の変数)

  – 同期に関する問題の大部分

- PCA(Performance and Coverage Analyzer) の使用によるイメージの実行

  PCA は次の問題を検出できます。

  – 実行時アラインメント・フォルト

– アプリケーションのどの部分がもっとも頻繁に実行され，性能に大きく影響するか

アプリケーションのすべてのイメージを AXP システムでエラーなしに実行できた場合には，イメージを組み合わせて実行し，イメージ間の同期に関する問題が発生しないかどうかをテストしなければなりません。テストについての詳しい説明は，第 6.3.2 項を参照してください。

4.4 再コンパイルするか，トランスレートするかの判断

イメージに対して 2 つの方法のどちらも使用できる場合には， AXP システム上でイメージのネイティブ・バージョンとトランスレートされたバージョンを実行した場合の性能を予測し，イメージのトランスレートに必要な作業とネイティブな AXP イメージに変換するのに必要な作業を判断し，性能と作業量のバランスを考えなければなりません。

一般に，アプリケーションを構成する各イメージは異なるモードで実行できます。たとえば，ネイティブな AXP イメージからトランスレートされた共有可能イメージ (translated shareable image) を呼び出したり，その逆に呼び出すことが可能です。 2 つのアーキテクチャが混在したアプリケーションについての詳しい説明は，第 4.4.2 項を参照してください。

表 4–1 では， 2 種類の移行方法を比較しています。

表 4–1 移行方法の比較

要素	再コンパイル/再リンク	トランスレート
性能	完全な AXP の機能	通常，ネイティブ AXP の 25 〜 40%の性能
必要な作業	容易な場合も困難な場合もある	容易

(次ページに続く)

表 4–1 (続き) 移行方法の比較

要素	再コンパイル/再リンク	トランスレート
スケジュールの制約	ネイティブ・コンパイラが提供されるかどうかに応じて異なる	なし，ただちに可能
サポートされるプログラム		
– バージョン	VAX VMS バージョン 4.0 以前のソースのコンパイラ	OpenVMS VAX バージョン 4.0 またはそれ以降のイメージのみがサポートされる
– 制約事項	特権付きコードがサポートされる	ユーザ・モードのコードだけがサポートされる
VAX との互換性	高い。ほとんどのコードは問題なく再コンパイル/再リンクできる	エミュレーションによって実現される
今後のサポートと保守	通常のソース・コードの保守	ソースは管理されない

アプリケーションをどのような方法で移行するかを決定するには，次の事項を考慮してください。

- アプリケーションをソース・コードから完全に再構築しますか，それとも一部の機能に対してはバイナリ・イメージを使用しますか?

  バイナリ・イメージを使用する場合には，それらをトランスレートしなければなりません。

- アプリケーションを構成する，すべてのイメージのソース・コードを入手できますか?

  ソース・コードを入手できないイメージは，トランスレートしなければなりません。

- どのイメージがアプリケーションの性能に大きな影響を与えますか?

  AXP システムの速度を有効に使用したいイメージは，再コンパイルしなければなりません。

  – 性能に大きな影響を与えるイメージを識別するには， Performance and Coverage Analyzer(PCA) を使用します。

  - AXP コンパイラが作成するイメージだけが AXP の処理機能を効率よく使用し，最適な性能を実現できます。トランスレートされた VAX イメージはネイティブ AXP コードの 1/3 程度の速度，またはそれ以下の速度で実行されます。実際の速度は，使用するトランスレーション・オプションに応じて異なります。

- 2 つの方法を使用した場合，各イメージを変換するのに必要な作業量はどの程度ですか?
  - アプリケーションの複雑さによりますが，通常ソフトウェア・トランスレーションのほうが，再コンパイルおよび再リンクよりも少ない作業量と作業時間で済みます。

    ネイティブ・バージョンに移行する場合， OpenVMS AXP で実行するために，アプリケーションの一部だけをトランスレートすることができます。
  - VAX アーキテクチャ固有の機能や， VAX 呼び出し規則に依存しているコードは直接再コンパイルできません。このようなコードはトランスレーションして実行するか，またはコードを変更して再コンパイルおよび再リンクしなければなりません。

アーキテクチャ間で互換性が維持されていない部分については，次の方法で対処できます。

- アーキテクチャに依存するコード・シーケンスは，プラットフォームから独立した方法で，同じ操作をサポートする高級言語のレキシカル要素に置き換えます。
- プロセッサ・アーキテクチャにとって適切な方法で作業を実行するために，OpenVMS システム・サービスに対する呼び出しを使用します。
- ソース・コードの変更箇所をできるだけ少なくし，正しくプログラムが動作するように，高級言語のコンパイラ・スイッチを使用します。

表 4–2 は，各プログラムのアーキテクチャに依存する部分が，プログラムを AXP システムに移行するための 2 つの方法に，どのような影響を与えるかを示しています。詳しくは，『OpenVMS AXP オペレーティング・システムへの移行：再コンパイルと再リンク』を参照してください。

表 4–2 移行方式の選択: アーキテクチャに依存する部分の取り扱い

再コンパイル/再リンクした VAX ソース	トランスレートされた VAX イメージ
データ・アラインメント[1]	
省略時の設定では，大部分のコンパイラはデータを自然なアラインメントにする。VAX アラインメントを保存するための修飾子についての説明は，『OpenVMS AXP オペレーティング・システムへの移行：再コンパイルと再リンク』を参照。	アラインされていないデータもサポートされるが，/OPTIMIZE=ALIGNMENT 修飾子を使用すれば，データがロングワードにアラインされていることを仮定することにより，実行速度を向上できる。
データ型	
H 浮動小数点は G 浮動小数点または IEEE T 浮動小数点に変更する。 D 浮動小数点の場合， D53 フォーマットの 15 桁の精度で十分なときは， D 浮動小数点を G 浮動小数点に変更する。アプリケーションで 16 桁の精度 (D56 フォーマット) が必要な場合には，トランスレートしなければならない。 COBOL のパック 10 進数は操作のために自動的にバイナリ形式に変換される。	D 浮動小数点の 16 桁の精度が必要な場合には，/FLOAT=D56_FLOAT 修飾子を使用する。この修飾子を使用した場合，性能は，省略時の設定である/FLOAT=D53_FLOAT を使用した場合より低下する。
読み込み/変更/書き込み操作の不可分性	
サポートは各コンパイラが準備しているオプションに応じて異なる (詳しくは『OpenVMS AXP オペレーティング・システムへの移行：再コンパイルと再リンク』を参照)。	/PRESERVE=INSTRUCTION_ATOMICITY 修飾子を使用する。実行速度は半分に低下する。
バイト書き込み操作とワード書き込み操作の不可分性と粒度	

[1]アラインされていないデータは主に性能上の問題となります。アラインされていないデータを参照した場合， VAX システムでは性能をある程度低下させるだけですが， AXP システムでアラインされていないデータをメモリからロードしたり，アラインされていないデータをメモリに格納すると，アラインされた操作の場合より最高 100 倍も時間がかかる可能性があります。

(次ページに続く)

表 4–2 (続き) 移行方式の選択: アーキテクチャに依存する部分の取り扱い

再コンパイル/再リンクした VAX ソース	トランスレートされた VAX イメージ
バイト書き込み操作とワード書き込み操作の不可分性と粒度	
適切にソース・コードを変更し，コンパイラ・オプションを使用することによりサポートされる (詳しくは『OpenVMS AXP オペレーティング・システムへの移行：再コンパイルと再リンク』を参照)。	/PRESERVE=MEMORY_ATOMICITY 修飾子を使用する。実行速度は半分に低下する。
ページ・サイズ	
OpenVMS リンカは省略時の設定により，大きい AXP スタイルのページを作成する。	512 バイトのページ・イメージの大部分はサポートされる。しかし， VEST はゆるやかな保護を割り当てるため，アクセス違反を検出するために厳しい保護に依存しているイメージを，トランスレートした場合， AXP システムで正しく実行されない。
読み込み/書き込みの順序	
適切な同期命令 (MB) をソース・コードに追加することによりサポートされるが，性能は低下する。	/PRESERVE=READ_WRITE_ORDERING 修飾子を使用する。
例外報告の即時性	
コンパイラ・オプションの使用によって部分的にサポートされる (詳しくは『OpenVMS AXP オペレーティング・システムへの移行：再コンパイルと再リンク』を参照)。	/PRESERVE=FLOAT_EXCEPTIONS 修飾子または/PRESERVE=INTEGER_EXCEPTIONS 修飾子を使用する。実行速度は半分に低下する。
VAX アーキテクチャおよび呼び出し規則への明示的な依存[2]	
サポートされない。依存する部分は削除しなければならない。	サポートされる。

[2]VAX アーキテクチャ固有の機能や呼び出し規則への依存としては， VAX 呼び出し規則， VAX 例外処理， VAX AST パラメータ・リスト， VAX 命令の形式と動作，および VAX 命令の実行時作成への明示的な依存などがある。

4.4.1 アプリケーションのトランスレート

アプリケーションを再コンパイルできない場合や，VAX アーキテクチャ固有の機能をアプリケーションで使用している場合には，アプリケーションをトランスレートすることができます。アプリケーションの一部だけのトランスレートもでき，移行プロセスの一段階として一時的にアプリケーションの各部分をトランスレートすることができます。

再コンパイルに影響を与える多くの相違点について第 4.2 節で説明しましたが，これらの相違点は，トランスレートされた VAX イメージの性能にも影響を与える可能性があります。次の方法を使用すれば，VAX アーキテクチャに依存するイメージの互換性を向上できます (詳しくは『DECmigrate for OpenVMS AXP Systems Translating Images』を参照してください)。

- データ・アラインメント

  VEST のトランスレート時修飾子/OPTIMIZE=NOALIGNMENT を使用すれば，VEST は特別なインラインの AXP コードを生成し，アラインされていないデータの参照に対する例外を生成しないようにします。この修飾子を使用した場合，VEST は，アラインされたデータ参照だけを使用するコードよりも実行速度が約 10 倍も遅い AXP コードを作成します (省略時のオプションである/OPTIMIZE=ALIGNMENT を使用した場合には，アラインされていないデータは例外を生成しますが，この結果，アラインされたデータを実行する場合より約 100 倍の時間がかかります)。

- 命令の不可分性

  トランスレータを起動するときに，トランスレート時修飾子 /PRESERVE=INSTRUCTION_ATOMICITY を指定すれば，VEST は，指定した VAX 命令セットに対して AST が不可分に実行されるような Alpha AXP 命令シーケンスを作成します。AST は，このような不可分な操作を実行する Alpha AXP 命令シーケンスの途中でも受け付けることができますが，AST ルーチンが終了したときに，命令シーケンスは最初から再起動されます。このため，/PRESERVE=INSTRUCTION_ATOMICITY 修飾子を指定した場合，特定のコード・シーケンスの実行速度は半分に低下する可能性があります (トランスレータが AST の不可分なコードを生成する VAX 命令の一覧と，ソフトウ

ェア・トランスレータについての詳しい情報については，『DECmigrate for OpenVMS AXP Systems Translating Images』を参照してください)。

- 読み込み/書き込みの粒度

  トランスレート時修飾子/PRESERVE=MEMORY_ATOMICITY を使用した場合， VEST は，バイト書き込みまたはワード書き込みの不可分性を保証します。この修飾子を指定すれば，メインライン・ルーチンと AST ルーチンはロングワードまたはクォドワードに格納されている隣接するバイトを相互に妨害せずに更新できます。/PRESERVE=MEMORY_ATOMICITY 修飾子は，自然にアラインされていないロングワードおよびクォドワード境界と交差するデータの不可分なアクセスを保証します。ただし，これらの修飾子を指定した場合，実行速度は半分に低下する可能性があります。

- ページ・サイズとゆるやかな保護

  VAX イメージを AXP システムで実行できるようにするために， VEST とイメージ・アクティベータは， VAX イメージ・セクションを大きいページにマッピングします。 8KB ページをサポートする AXP プロセッサでは，最大 16 ページの VAX ページを 1 ページに格納できます。しかし，この大きいページは 1 つのページ・テーブル・エントリによってのみ記述されるため，そのページには 1 つの保護と 1 つのバッキング・ストアだけしか割り当てることができません。したがって， VEST が AXP ページに割り当てる保護は，マッピングする AXP イメージ・セクションに関連する保護のうち，もっとも制限のゆるやかな保護です。このため，アクセス違反を検出するために厳しい保護に依存している VAX イメージは，トランスレートされた場合， AXP システムで正しく実行されません。

  この問題に対処するための方法として，省略時のリンカ・オプション/BPAGE を使用して VAX でプログラムを再リンクし，ページを 64KB 境界にアラインする方法があります。

- 正確な算術演算例外

  VEST では，トランスレート時に/PRESERVE=FLOAT_EXCEPTIONS 修飾子または/PRESERVE=INTEGER_EXCEPTIONS 修飾子を使用することにより，特定の例外タイプに対して正確な例外報告を設定できます。これらの修飾子を指定した場合には，一部のコード・セグメントの実行速度は半分に低下します。

- VAX 命令の実行時作成

  実行時に作成される VAX 命令は，トランスレーションのもとでエミュレーションによって実行されます。しかし，エミュレートした命令はトランスレートされた命令より大幅に実行速度が遅く，アプリケーションの重要な部分の性能を向上するために実行時にコードを生成する場合，これは問題になります。

アーキテクチャ上のさまざまな相違点がイメージのトランスレートによってどのようにサポートされるかについては，表 4–2 を参照してください。

4.4.2 ネイティブ・イメージとトランスレートされたイメージの混在

一般に， AXP システムではネイティブな AXP イメージとトランスレートされたイメージを組み合わせて使用できます。たとえば，ネイティブな AXP イメージからトランスレートされた共有可能イメージを呼び出したり，その逆の呼び出しを実行できます。

ネイティブなイメージとトランスレートされたイメージを組み合わせて実行するには， VAX 呼び出し規則と Alpha AXP 呼び出し規則の間で呼び出しを実行できなければなりません。ネイティブ・イメージとトランスレートされたイメージが次の条件を満たす場合には，特別な処理は必要ありません。

- ネイティブな AXP イメージのルーチン・インターフェイス・セマンティックとデータ・アラインメント規則が VAX イメージのこれらのセマンティックおよび規則と等しいこと。
- すべてのエントリ・ポイントが CALL*x*であること。つまり，外部 JSB エントリ・ポイントが存在しないこと。高級言語で作成されたコードの場合には，この条件は満たされます。

一方の呼び出し規則を使用するプロシージャ (呼び出し元) が別の呼び出し規則を使用するプロシージャ (呼び出し先) を呼び出す場合には，**ジャケット・ルーチン**を使用して間接的に呼び出します。ジャケット・ルーチンはプロシージャのコール・フレームと引数リストを解釈し，呼び出し先プロシージャの対応するコール・フレームと引数リストを作成し，制御を呼び出し先プロシージャに渡します。呼び出し先プロシージャが実行を終了すると，ジャケット・ルーチンを通じて，制御を呼び出し元に戻します。ジャケット・ルーチンは呼び出し先ルーチンのリターン・レジス

タの値を呼び出し元ルーチンのレジスタに書き込み，制御を呼び出し元プロシージャに戻します。

OpenVMS AXP オペレーティング・システムは，ほとんどの呼び出しに対してジャケット・ルーチンを自動的に作成します。自動的なジャケット機能を使用するには，アプリケーションの中でネイティブな AXP の部分を作成するときに，コンパイラ修飾子/TIE とリンカ・オプション/NONATIVE_ONLY を使用してします。

特定の場合には，アプリケーション・プログラムは特別に作成したジャケット・ルーチンを使用しなければなりません。たとえば，次のようなライブラリに対する非標準呼び出しの場合には，ジャケット・ルーチンを作成しなければなりません。

- 外部 JSB エントリ・ポイントを含む VAX 共有可能ライブラリ
- 転送ベクタに読み込み/書き込みデータを含むライブラリ
- VAX 固有の関数を登録したライブラリ
- ライブラリのネイティブ・バージョンとトランスレートされたバージョンの間で共有しなければならない資源を使用するライブラリ
- VAX イメージが提供していたすべてのシンボルを提供またはエクスポートしないネイティブな AXP ライブラリ

  (エクスポートという用語は，ルーチンがイメージのグローバル・シンボル・テーブル (GST) に登録されたことを意味します。)

これらの状況でジャケット・イメージを作成する方法については，『DECmigrate for OpenVMS AXP Systems Translating Images』を参照してください。

OpenVMS AXPオペレーティング・システムとともに提供されるトランスレートされた共有可能イメージ (たとえば，ネイティブな AXP コンパイラのない言語のランタイム・ライブラリなど) には，ジャケット・ルーチンが添付されており，これらのジャケット・ルーチンを使用すれば，トランスレートされた共有可能イメージをネイティブな AXP イメージから呼び出すことができます。

5
移行計画の作成

アプリケーションを調査し，移行方法が決まったら，移行計画を作成します。移行計画では，移植性に関する問題と，ハードウェアおよびソフトウェアへの依存に関する問題も含めて，アプリケーションの技術的な分析結果を詳しく記述し，各移行段階の概要を示し，移行に関する各作業を誰が実行するかを指定します。移行計画に記述した情報は，アプリケーションの将来の移植性と，性能を向上するための最適化の可能性に関して，技術的およびビジネス的な判断を下すのに役立ちます。

移行計画の概要

この章では，典型的な移行計画の概要を示します。移行計画の具体的な例については，付録 B を参照してください。

I エクゼクティブ・サマリ

移行の概要。移行作業の目標を記述し，機能，品質，納期，および性能上の目標を設定します。アプリケーションが現在実行されている主なプラットフォームを列記し，アプリケーションで使用されている言語も示します。また，アプリケーションの依存性やリスクが，移行目標や移行の完了にどのような影響を与えるかを記述します。最後に，必要となる資源と完了予定日も指定します。

A. アプリケーションの識別

名前，所有者

B. アプリケーションの機能

アプリケーションが何を実行するのか

C. 移行の全般的な計画

他のリリース計画との関係も指定します。

トランスレーションの使用目的

- アプリケーションを移行するため

- ネイティブ・コンパイラを使用できるようになるまでの準備として

II 技術分析

この部分では，イメージ分析とソース分析の結果をもとに，アプリケーションを OpenVMS AXP に移行するための作業範囲を定義します。

A. アプリケーションの特性

これはアプリケーションの一般的な記述であり，アプリケーションに含まれるイメージの数，プロシージャの数，データ・ファイルの数，コードの行数とモジュール数，使用されている言語と各言語の割合および特定の機能，VAX MACROが使用されている場合はその理由，アプリケーションが一般に移植可能であるのか， VAX アーキテクチャに依存しているのか，アプリケーションを実行または変更するために特殊なハードウェアまたはソフトウェアが必要かも記述します。

B. イメージ調査の結果

- テストしたイメージ
- 検出された VEST メッセージ/エラー

C. ソース分析の結果

どのようなソース分析を実行したかと，検出された問題点の一般的な説明。

付録 A に示した質問を使用してください。

D. 移行に関する問題

次の各要素に対してアプリケーションが依存しているかどうかを記述してください。

- データ・アラインメント
- VAX データ型
- 読み込み/書き込みの粒度
- ページ・サイズ
- 読み込み/書き込みの順序
- 例外の即時報告
- VAX プロシージャ呼び出し規則

- VAX 例外処理メカニズム
- VAX AST パラメータ・リスト
- VAX 命令の形式と動作
- VAX 命令の実行時作成

III 中間目標と成果物

移行の中間目標を示し，スケジュールを決定します。

IV 技術的アプローチ

移行の各段階で対処する問題を示し，これを前に示した中間目標と比較します。主な問題に対処する順序，それらの問題に対処する人，およびその場所を指定します。さらに DEC の移行サービスを利用するかどうかも決定します。

次の例を参照してください。

A. 正しくアラインされていないメモリ参照の修正

B. ページ・サイズへの依存の修正

C. ANSI 以外のコーディング構造の修正

D. 不可分性に関する問題の修正

E. その他の依存性への対処

必要に応じて次の情報も指定してください。

A. ジャケット・ルーチンが自動的に作成されないときに，ジャケット・ルーチンを作成しなければならないかどうか。

B. アプリケーションをコンパイル，リンク，またはトランスレートするために特殊なセットアップ・プロシージャが必要かどうか。たとえば，コンパイラ修飾子である/TIE やリンカ・オプションである/[NO]NATIVE_ONLY を使用するかどうか。

移行の各段階で必要となる作業量

V 依存関係とリスク

他のソフトウェア，ハードウェア，他の組織からのサポート

VI 必要な資源

A. ハードウェア

B. 訪問トレーニング

C. 電話によるサポート

D. 技術的な援助

E. テストの支援

6

アプリケーションの移行

アプリケーションを実際に AXP システムに移行する作業は，次に示すように複数の段階に分かれています。

- 移行環境を設定する
- 移行評価の基礎を設定するために VAX システムでアプリケーションをテストする
- AXP システムで実行するためにアプリケーションを変換する
- 移行したアプリケーションをデバッグおよびテストする
- 移行したアプリケーションをソフトウェア・システムに統合する

6.1 移行環境の設定

ネイティブな AXP 環境は， VAX システムと同様に完全な開発環境です (しかし，いくつかの DEC のコンパイラは，現在のところ AXP システム上で使用できません)。

現在のところ，移行したアプリケーションのデバッグおよびテストは， AXP システム上で行わなければなりません。

Alpha AXP 移行環境の重要な要素は DEC からサポートされ， DEC はアプリケーションの変更，デバッグ，およびテストのために必要な支援を提供できます。

6.1.1 ハードウェア

移行のためにどのハードウェアが必要かを計画する場合，複数の問題を検討しなければなりません。まず，通常の VAX 開発環境でどのような資源が必要であるかを検討してください。

- CPU
- ディスク
- メモリ

AXP 移行環境にとって必要な資源を見積もるには，次の問題を考慮しなければなりません。

- AXP システムでは，イメージ・サイズが従来より大きくなること

  VAX システムと AXP システムとの間で，コンパイルしたイメージとトランスレートしたイメージを比較してください。
- AXP システムでは，ページ・サイズと物理メモリ・サイズが従来より大きくなること
- 必要な CPU

VEST を使用すると，多くの CPU 時間が必要となります (実際に必要な CPU 時間を予測することは困難です。これは，必要な CPU 時間は，アプリケーションのサイズよりもアプリケーションの複雑さに大きく依存するからです)。また， VEST を使用する場合，ログ・ファイルのためのディスク空間， AXP イメージのためのディスク空間，フローグラフのためのディスク空間などが大量に必要になります。新しいイメージには，元の VAX 命令と新しい Alpha AXP 命令の両方が含まれます。したがって，元の VAX イメージより必ず大きくなります。

望ましい構成は次のとおりです。

- 256 MB のメモリを装備した 6 VUP 以上のマルチプロセッサ・システム
- 1 GB のシステム・ディスク
- 各アプリケーション当り 1 GB のディスク

マルチプロセッサ・システムでは，各プロセッサが別々のアプリケーションのイメージ分析を行うことができます。

コンピュータ資源が不足する場合には，次のいずれかの処置で対処してください。

- コンパイラまたは VEST は，作業量の少ない時刻にバッチ・ジョブとして実行してください。
- 移行作業を AXP Migration Center (AMC) で実行してください。
- 移行作業のために必要な機器をリースしてください。

6.1.2 ソフトウェア

効率のよい移行環境を構築するには，次の要素を確認しなければなりません。

- 移行ツール

  次のツールも含めて，互換性のある移行ツールが必要です。

  - コンパイラ
  - トランスレーション・ツール
    - VEST と VEST/DEPENDENCY
    - TIE
  - RTL
  - システム・ライブラリ
  - C プログラムのためのインクルード・ファイル
- 論理名

  VAX バージョンと AXP バージョンのツールとファイルをそれぞれ適切に指すように，論理名は矛盾のないように定義しなければなりません。詳しくは第 6.4 節を参照してください。
- コンパイル/リンク・プロシージャ

  これらのプロシージャは新しいツールおよび新しい環境に適合するように調整しなければなりません。

- ソースの管理とイメージのビルドのためのツール
  - CMS
  - MMS

ネイティブな AXP 開発
VAX で使用できる標準的な開発ツールはすべて， AXP システムでもネイティブ・ツールとして提供されています。

トランスレーション
VEST ソフトウェア・トランスレータは VAX システムと AXP システムの両方で実行されます。 Translated Image Environment (TIE) はトランスレートされたイメージを実行するために必要な環境であり， OpenVMS AXP の一部です。したがって，トランスレートされたイメージの最終的なテストは AXP システムで実行しなければなりません。

6.2 アプリケーションの変換

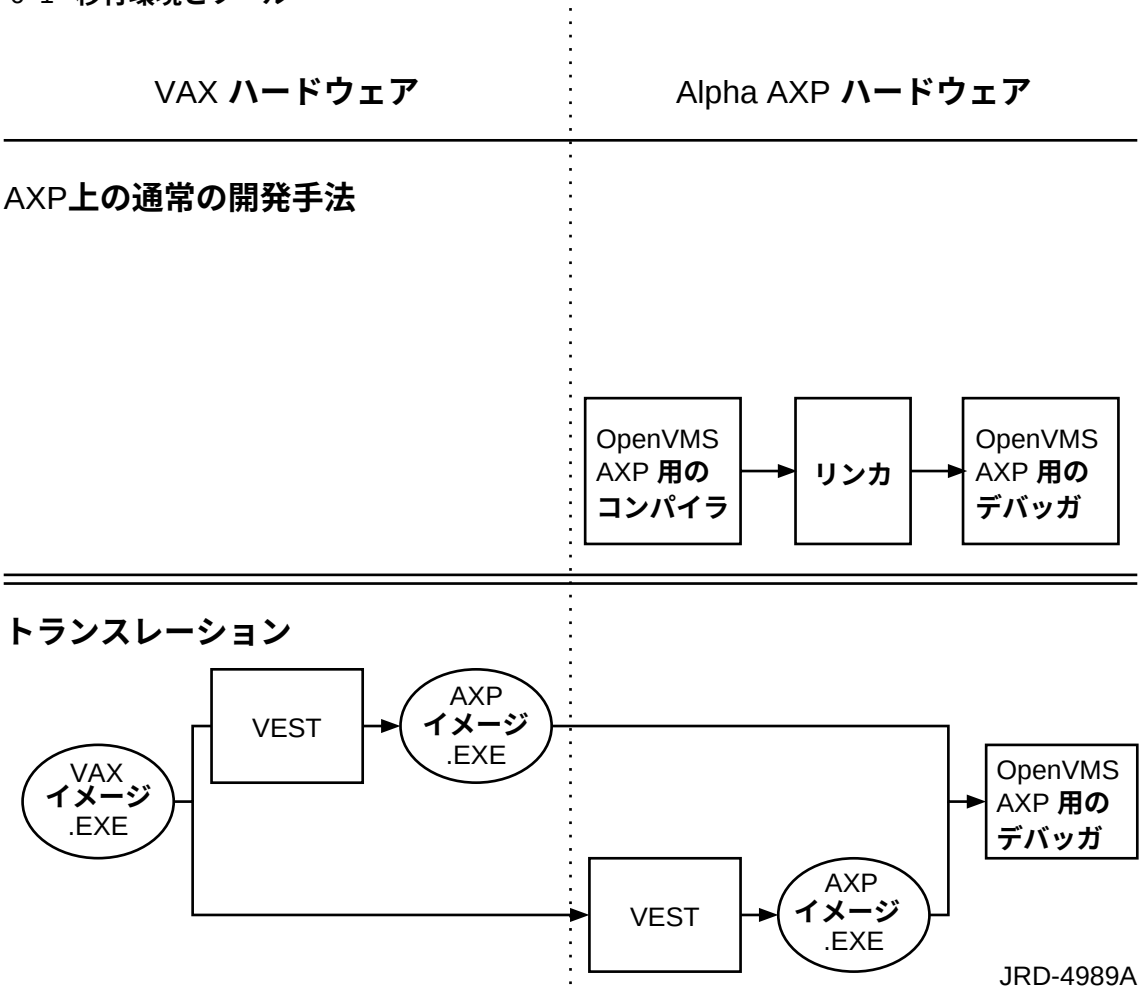
コードを完全に分析し，移行計画を適切に作成している場合には，この最終段階はかなり簡単に終了できます。多くのプログラムはまったく変更せずに再コンパイルまたはトランスレートできます。直接に再コンパイルまたはトランスレートできないプログラムでも，多くの場合，簡単な変更だけで AXP システムで実行できるようになります。

コードの実際の変換についての詳しい説明は， OpenVMS AXP の移行に関する次の解説書を参照してください。

- 『OpenVMS AXP オペレーティング・システムへの移行：再コンパイルと再リンク』
- 『DECmigrate for OpenVMS AXP Systems Translating Images』
- 『Migrating to an OpenVMS AXP System: Porting VAX MACRO Code』

これらの各解説書についての説明は，本書のまえがきを参照してください。

図 6–1 移行環境とツール



2 つの移行環境と，各環境で使用される基本的なツールは，図 6–1 に示すとおりです。

6.2.1 再コンパイルと再リンク

一般に，アプリケーションを移行する場合には，コードの変更，コンパイル，リンク，およびデバッグを繰り返し実行しなければなりません。これらの処理を実行することにより，開発ツールから指摘されたすべての構文エラーとロジック・エラーを解決します。通常，構文エラーは簡単に修正できますが，ロジック・エラーを修正するには，コードの大幅な変更が必要になります。

新しいコンパイラ・スイッチやリンカ・スイッチに対応するために，コンパイル・コマンドとリンク・コマンドは，ある程度変更しなければなりません。たとえば，複数の AXP プラットフォーム間で移植可能にするために，リンカは AXP システムの省略時のページ・サイズを 64KB として設定します。この結果，各プロセッサのシステム・ページ・サイズとは無関係に， OpenVMS AXP イメージはどの AXP プロセッサでも実行可能になります。また， AXP の共有可能イメージは，普遍的なエントリ・ポイントとシンボルを宣言するために， VAX 転送ベクタ・メカニズムではなく，リンカ・オプション・ファイル内のシンボル・ベクタ宣言を使用します。

AXP プラットフォームでソフトウェアを開発し，移行するために，多くのネイティブ・コンパイラとその他のツールが提供されます。

6.2.1.1 ネイティブな AXP コンパイラ

既存の VAX ソースを再コンパイルおよび再リンクすると，ネイティブな AXP イメージが作成され，このイメージは RISC アーキテクチャの性能上の利点をすべて利用して， AXP 環境で実行されます。 AXP コードでは，一連の高度に最適化されたコンパイラを使用します。これらのコンパイラは共通の最適化コード・ジェネレータを備えています。しかし，各言語に対して異なるフロント・エンドを使用し，これらの各フロント・エンドは現在の VAX コンパイラと互換性があります。

OpenVMS AXPオペレーティング・システムバージョン 6.1 では，次の言語に対してネイティブな AXP コンパイラが提供されています。

- Ada
- BASIC
- C

- C++
- COBOL
- FORTRAN
- Pascal
- PL/I
- MACRO-32 (クロス・コンパイラ)

OpenVMS AXPの将来のリリースでは，LISP も含めた他の言語のためのネイティブ・コンパイラも提供されます。

他の言語で作成されたユーザ・モードの VAX プログラムは，VEST を使用してトランスレートすることにより，AXP システムで実行できます。また，サード・パーティからも他の言語のためのコンパイラが提供されます。

一般に，AXP コンパイラには，コマンド・ライン修飾子と言語セマンティックが準備されており，VAX アーキテクチャに依存するコードをほとんど変更せずに，AXP システムでも実行できるようにしています。このような VAX アーキテクチャへの依存のリストについては，表 4–2 を参照してください。詳しい説明は，『OpenVMS AXP オペレーティング・システムへの移行：再コンパイルと再リンク』を参照してください。

VAX プログラムを AXP システムに移行するために，AXP コンパイラを使用する方法についての説明は，『OpenVMS AXP オペレーティング・システムへの移行：再コンパイルと再リンク』を参照してください。

6.2.1.2 その他の開発ツール

ネイティブな AXP イメージを作成するために，コンパイラの他にもいくつかのツールが提供されます。

- OpenVMS リンカ

  OpenVMS リンカは，VAX オブジェクト・ファイルまたは AXP オブジェクト・ファイルを受け付け，VAX イメージまたは AXP イメージを作成できるようになりました。また，VAX ハードウェアで AXP イメージを作成できるため，クロス・リンカとして機能します。

- OpenVMS デバッガ

  OpenVMS AXP で実行される OpenVMS デバッガは，現在の OpenVMS VAX デバッガと同じコマンド・インターフェイスを使用します。 OpenVMS AXP システムでも， VAX システム上のグラフィカル・インターフェイスが提供されています。

- OpenVMS ライブラリアン・ユーティリティ

  OpenVMS ライブラリアン・ユーティリティは， VAX ライブラリまたは AXP ライブラリを作成します。

- OpenVMS メッセージ・ユーティリティ

  OpenVMS メッセージ・ユーティリティを使用すれば， OpenVMS システム・メッセージに独自のメッセージを追加できます。

- MACRO–64 Assembler for OpenVMS AXP

  MACRO–64 Assembler はネイティブな AXP アセンブリ言語を作成します。

- ANALYZE/IMAGE ユーティリティ

  ANALYZE/IMAGE ユーティリティは VAX イメージまたは AXP イメージを分析できます。

- ANALYZE/OBJECT ユーティリティ

  ANALYZE/OBJECT ユーティリティは VAX オブジェクトまたは AXP オブジェクトを分析できます。

- DECset

  DECset は包括的な CASE ツール群であり， Language Sensitive Editor (LSE)， Source Code Analyzer (SCA)， Code Management System (CMS), Module Management System (MMS) をはじめ，多くの要素で構成されます。

6.2.2 トランスレーション

AXP システムで実行するために VAX イメージをトランスレートする処理については，『DECmigrate for OpenVMS AXP Systems Translating Images』を参照してください。一般に，この処理は簡単ですが，エラーなしにトランスレートするには，コードを少し変更しなければならないことがあります。

6.2.2.1 VAX Environment Software Translator (VEST) と Translated Image Environment (TIE)

ユーザ・モードの VAX イメージをOpenVMS AXPに移行するための主なツールは，静的トランスレータと実行時サポート環境です。

- VAX Environment Software Translator (VEST) は， VAX イメージを分析し，イメージをトランスレートし，同じ機能を実行するイメージを作成するユーティリティです。 VEST を使用すれば，次の処理が可能です。
  - VAX イメージをトランスレートできるかどうかを判断する
  - VAX イメージを AXP イメージにトランスレートする
  - イメージ内で OpenVMS AXP と互換性のない部分を識別し，これらの互換性のない部分をソース・ファイルで修正する方法を入手する
  - トランスレートされたイメージの実行時の性能を改善する方法を判断する
- Translated Image Environment (TIE) は，トランスレートされたイメージを実行時にサポートする AXP の共有可能イメージです。 TIE は，トランスレートされたイメージを実行するために OpenVMS VAX に類似した環境を準備し，ネイティブな AXP システムとのすべてのやりとりを処理します。 TIE には次の要素が含まれています。
  - VAX 命令インタプリタ
    次の機能をサポートします。
    - VAX システムでの実行に類似した VAX 命令の実行 (命令の不可分性も含む)
    - サブルーチンとしての複雑な VAX 命令
  - VAX と互換性のある例外ハンドラ
  - ネイティブ・コードとトランスレートされたコードの間のやりとりを可能にするジャケット・ルーチン
  - エミュレートされた VAX スタック

  トランスレートされたイメージを実行する場合には， TIE が自動的に起動されます。 TIE を呼び出す必要はありません。

VEST は，できるだけ多くの VAX コードを AXP コードにトランスレートします。TIE は， Alpha AXP 命令に変換できなかった VAX コードを解釈します。たとえば，次のコードは TIE によって解釈されます。

- VEST が静的に識別できなかった命令
- H 浮動小数点および D56 (56 ビットの D 浮動小数点) 浮動小数点演算

命令の解釈は速度の遅い処理であり，おそらく平均的な 1 つの VAX 命令に対して 100 前後の Alpha AXP 命令が必要となるため， VEST は実行時にインタプリタを通じた解釈の必要性をできるだけ少なくするために，できるだけ多くの VAX コードをトランスレートしようとします。トランスレートされたイメージは，ネイティブな AXP イメージと比較すると，約 3 分の 1 の速度で実行されます。ただし，この速度は TIE がどれだけの VAX コードを解釈しなければならないかに応じて異なります。トランスレートされた VAX イメージは，少なくとも VAX ハードウェアと同じ速度で実行されます (同じレベルのプロセス技術を使用した CPU の場合)。

AXP システムで VAX イメージの動的解釈を指定することはできません。イメージをOpenVMS AXPで実行するには，その前に VEST を使用してイメージをトランスレートしなければなりません。

VAX イメージをトランスレートすると， AXP ハードウェアで実行されるイメージが作成されます。このようにして作成される AXP イメージは，単に VAX イメージを解釈したバージョンやエミュレートしたバージョンではなく，元の VAX イメージ内の命令が実行する操作と同じ操作を実行する Alpha AXP 命令を含むイメージです。 AXP の.EXE ファイルには，元の VAX イメージが完全に登録されているため， TIE は VEST がトランスレートできなかったコードを解釈できます。

VEST の分析機能は，トランスレートする場合だけでなく，再コンパイルする予定のプログラムを評価する場合も役立ちます。

VEST と TIE についての詳しい説明は，『DECmigrate for OpenVMS AXP Systems Translating Images』を参照してください。このマニュアルでは，フローグラフなども含めて， VEST が作成するすべての出力とその解釈方法が詳しく説明されています。また， VEST が作成するイメージ情報ファイル (IIF) から提供される情報を利用して，トランスレートされたイメージの実行時の性能を向上する方法も説明されています。

6.3 移行したアプリケーションのデバッグとテスト

アプリケーションを OpenVMS AXP に移行した後，デバッグしなければなりません。

また，正しい操作が実行されるかどうかを調べるためにアプリケーションをテストすることも必要です。

6.3.1 デバッグ

OpenVMS オペレーティング・システムでは，次のデバッガが準備されています。

- OpenVMS デバッガは VAX プログラムとネイティブな AXP プログラムの両方のデバッグをサポートします。ただし，トランスレートされたイメージのデバッグはサポートしません。

  OpenVMS デバッガはシンボリック・デバッガです。つまり，このデバッガを使用する場合には，プログラムで使用しているシンボル (変数名やルーチン名，ラベルなど) によってプログラムの位置を参照できます。プログラム位置を参照するために，メモリ・アドレスやマシン・レジスタを指定する必要はありません。ただし，メモリ・アドレスやマシン・レジスタを指定したい場合は，必要に応じて指定できます。

  OpenVMS デバッガは通常，トランスレートされたイメージに対して動作しません。しかし，トランスレートされたイメージは内部で VAX レジスタをエミュレートしているので， SHOW CALL や SHOW STATE コマンドでデバッグに有用な VAX コンテキストを得ることができます。

- Delta デバッガは， OpenVMS VAX プログラムと OpenVMS AXP プログラムのデバッグをサポートします。このデバッガはまた，トランスレートされたイメージのデバッグもサポートします。

  Delta デバッガはアドレス・ロケーション・デバッガです。つまり，このデバッガを使用する場合には，アドレス・ロケーションによってプログラム位置を参照しなければなりません。このデバッガは主に，特権付きプロセッサ・モードまたは高い割り込みレベルで実行されるプログラムをデバッグするために使用します。

デバッグは Alpha AXP ハードウェアで実行しなければなりません。

6.3.1.1 OpenVMS デバッガによるデバッグ

OpenVMS デバッガを使用して，移行したアプリケーションを AXP システムでデバッグする場合には，次のことを考慮しなければなりません。

- AXP コンパイラが使用できる言語で作成されたプログラムならばデバッガを使用できます。
- デバッガは，インストールされている常駐イメージのデバッグをサポートしません。インストールされている常駐イメージについての詳しい説明は，『OpenVMS システム管理者マニュアル (下巻)』を参照してください。
- このデバッガはインライン・ルーチンのデバッグをサポートしません。インライン・ルーチンをデバッグしようとすると，デバッガはルーチンをアクセスできないことを示すメッセージを出力します。

```
DBG> %DEBUG-E-ACCESSR, no read access to address 00000000
```

- このデバッガは Register Frame Procedures または No Frame Procedures のデバッグを完全にはサポートしません。これらのプロシージャに対して STEP/OVER コマンドまたは STEP/RETURN コマンドを実行すると，予測できない結果が発生します。

OpenVMS デバッガによるデバッグについての詳しい説明は，『デバッガ説明書』を参照してください。

6.3.1.2 Delta デバッガよるデバッグ

Delta デバッガを使用すれば，部分的または完全にトランスレートされたアプリケーションをデバッグできます。

トランスレートされたアプリケーション

トランスレートされたイメージをデバッグする場合には，次のことを確認しなければなりません。

- トランスレートを行うプログラムが OpenVMS VAX バージョン 6.1 のもとで正しく動作するかどうかを確認しなければなりません。
- ランタイム・ライブラリの IIF ファイルと VEST が使用中の OpenVMS AXP のバージョンと同じリリースであるかどうかを確認しなければなりません。

- AXP 命令および VAX 命令を得るために， VEST の/DEBUG，/LIST および /SHOW=MACHINE_CODE 修飾子を使用します (リストのアスタリスクは，VAX 命令を示しています)。比較を行うために， VAX イメージのマップ・ファイルとリスティング・ファイルを用意してください。

トランスレートされたイメージのデバッグについての詳しい説明は， DEC サービス (Alpha AXP Resource Center) にお問い合わせください。

アプリケーションの混在

ネイティブな AXP コードとトランスレートされたコードが混在するアプリケーションをデバッグするには，/TIE 修飾子を使用してアプリケーションのネイティブな部分がコンパイルされているかどうかを確認しなければなりません。さらに，/NONATIVE_ONLYリンカ・オプションを使用してアプリケーションをリンクしなければなりません。

Delta デバッガによるデバッグについての詳しい説明は，『OpenVMS Delta /XDelta Debugger Manual』を参照してください。

6.3.2 テスト

移行したバージョンの性能と機能を元の VAX バージョンと比較するために，アプリケーションをテストしなければなりません。

テストではまず， VAX アプリケーションに対して一連のテストを実行することにより，アプリケーションに対して基準となる結果を設定します。

アプリケーションを AXP システムで実行した後，次の 2 種類のテストを実行できます。

- アプリケーションの VAX バージョンに対して使用される標準テスト
- アーキテクチャの変更による問題を特に確認するための新しいテスト

6.3.2.1 VAX テスト

新しいアーキテクチャを使用することにより，アプリケーションの一部が変更されるため， OpenVMS AXP にアプリケーションを移行した後，そのアプリケーションをテストすることは特に重要です。アプリケーションの変更によってエラーが発生するだけでなく，新しい環境では， VAX バージョンで検出されなかった問題が発生する可能性があります。

移行されたアプリケーションをテストするには，次の操作が必要です。

1. 移行する前に，アプリケーションにとって必要な標準データを入手する
2. アプリケーションだけでなく，一連のテストも移行する (テストが AXP でまだ準備されていない場合)。
3. AXP システムでテストを検証する
4. 移行したアプリケーションに対して移行したテストを実行する

ここでは，リグレッション・テストとストレス・テストが役立ちます。ストレス・テストは，同期に関するプラットフォームの相違点をテストするために特に重要です。特に，複数の実行スレッドを使用するアプリケーションの場合は，ストレス・テストが役立ちます。

6.3.2.2 AXP テスト

標準テストは，移行したアプリケーションの機能を検証するためにかなり長くかかりますが，移行固有の問題を調べるためのテストをいくつか追加しなければなりません。特に次の点に注意してください。

- コンパイラの相違点

  つまり，最適化およびデータ・アラインメントの変更

- アーキテクチャの相違点

  たとえば命令の不可分性，メモリの不可分性，読み込み/書き込み順序などの変更

- 統合

  異なる言語で作成されたモジュールや，トランスレートしなければならなかったモジュールの統合

6.4 移行したアプリケーションのソフトウェア・システムへの統合

再コンパイルまたはトランスレーションによってアプリケーションを移行した後，移行によって他のソフトウェアとのやりとりに問題が発生していないかどうかを確認しなければなりません。

VAX と Alpha AXP システム間の相互操作性に関する問題の原因として，次のことが考えられます。

- VMS クラスタ環境の AXP システムと VAX システムは，別々のシステム・ディスクを使用しなければなりません。アプリケーションが正しいシステム・ディスクを参照しているかどうか，確認する必要があります。
- イメージ名

  混在する環境では，アプリケーションが正しいバージョンを参照するかどうかに関して，次のことを確認してください。

  – イメージのネイティブ VAX バージョンとネイティブな AXP バージョンが同じ名前を持つこと

  – イメージをトランスレートしたバージョンで名前の最後に "_TV"という文字列が追加されていること
- 再コンパイルしたイメージではデータが自然にアラインされていると考えられますが，トランスレートされたイメージでは，元の VAX イメージと同様にデータはアラインされていない可能性があります。

A

アプリケーション評価チェックリスト

このチェックリストの例は， OpenVMS AXP に移行する対象としてアプリケーションを評価するために DEC が使用したチェックリストをもとにしています。

質問の後の大括弧で囲んだコメントは，その質問の目的を正確に示しています。

アプリケーション評価チェックリスト

開発の履歴と計画

1. アプリケーションは現在，他のオペレーティング・システムまたはハードウェア・アーキテクチャで実行されていますか? ☐ YES ☐ NO

   その場合には，アプリケーションは現在， RISC システムで実行されていますか? ☐ YES ☐ NO

   [その場合には， OpenVMS AXP への移行は容易に実行できます。]

2. 移行後のアプリケーションの開発/保守計画はどのようになっていますか?

   a. 開発は行わない ☐ YES ☐ NO

   b. 保守リリースのみ ☐ YES ☐ NO

   c. 機能を追加または変更する ☐ YES ☐ NO

   d. VAX ソースと AXP ソースを個別に保守する ☐ YES ☐ NO

   [a に対する回答が "YES"の場合には，アプリケーションのトランスレートを考慮してください。 b または c に対する回答が "YES"の場合には，アプリケーションの再コンパイルと再リンクによって得られる利点を評価してください。ただし，トランスレーションも可能です。 VAX ソースと AXP ソースを個別に保守する場合，つまり， d に対する回答が "YES"の場合には，相互操作性と整合性に関する問題を考慮しなければなりません。特に，アプリケーションの異なるバージョンが，同じデータベースをアクセスできる場合には，このことを考慮しなければなりません。]

外部的な依存性

3. アプリケーションの開発環境を設定するために，どのようなシステム構成 (CPU，メモリ，ディスク) が必要ですか? _____

   [この質問は移行に必要な資源の計画を立てるのに役立ちます。]

4. アプリケーションの典型的なユーザ環境を設定するために，どのようなシステム構成 (CPU，メモリ，ディスク) が必要ですか? インストレーション検証プロシージャ，リグレッション・テスト，ベンチマーク，作業負荷も含めて考慮してください。

   [この質問は，ユーザ環境全体を OpenVMS AXP で実現できるかどうかを，判断するのに役立ちます。]

5. アプリケーションで特殊なハードウェアが必要ですか? ☐ YES ☐ NO

   [この質問は，必要なハードウェアを OpenVMS AXP で使用できるかどうかと，アプリケーションにハードウェア固有のコードが含まれているかどうかを判断するのに役立ちます。]

6. a. アプリケーションは現在， OpenVMS のどのバージョンで実行されていますか?

   b. アプリケーションは OpenVMS VAX バージョン 6.1 で実行されていますか? ☐ YES ☐ NO

   c. アプリケーションは OpenVMS AXP で有効でない機能を使用しますか? ☐ YES ☐ NO

   [OpenVMS AXP への移行の基礎となるのは， OpenVMS VAX バージョン 6.1 です。 c に対する回答が "YES"の場合には，アプリケーションは OpenVMS AXP でまだサポートされていない機能を使用する可能性があり，また OpenVMS AXP の現在のバージョンと互換性のない OpenVMS RTL，または他の共有可能イメージに対してリンクされている可能性があります。]

7. アプリケーションを実行するために，レイヤード・プロダクトが必要ですか?

   a. DEC が提供するプロダクト (コンパイラ RTL 以外のプロダクト) ☐ YES ☐ NO

   b. サード・パーティが提供するプロダクト: ☐ YES ☐ NO

   [a に対する回答が "YES"のときに， OpenVMS AXP で DEC のレイヤード・プロダクトが提供されているかどうかがよくわからない場合には， DEC の担当者に質問してください。 b に対する回答が "YES"の場合には，サード・パーティ・プロダクトの業者に問い合わせてください。]

アプリケーションの構造

8. アプリケーションのサイズは?

   モジュール数は? _____

   コードの行数またはキロバイト数は? _____

   必要なディスク空間? _____

   [この質問は，移行のために必要な作業量と資源の "サイズ "を判断するのに役立ちます。]

9. a. アプリケーションはどの言語で作成されていますか? (複数の言語が使用されている場合には，各言語の割合を示してください。) _____

   [コンパイラがまだ提供されていない場合には，アプリケーションをトランスレートするか，または別の言語で再作成しなければなりません。]

   b. VAX MACROを使用している場合には，その理由は何ですか? _____

   c. VAX MACROコードの機能を各高級の言語コンパイラ，またはシステム・サービス (プロセス名を検索するための$GETJPI など) で実行できますか? ☐ YES ☐ NO

   [AXP アプリケーションで，VAX MACROや MACRO–64 Assembler for OpenVMS AXP を使用することは，望ましくありません。アプリケーションを最初に開発した段階では，まだ提供されていなかった OpenVMS システム・サービスを呼び出すことにより，特定のユーザ・モード・アプリケーションで，アセンブリ言語コードを他のコードに置換することができます。]

10. a. アプリケーションを構成する，すべてのソース・ファイルをアクセスできますか? ☐ YES ☐ NO

    b. DEC サービスの使用を検討している場合には，DEC がこれらのソース・ファイルとビルド・プロシージャをアクセスできますか? ☐ YES ☐ NO

    [a に対する回答が "YES"の場合には，ソース・ファイルを入手できない部分の移行はトランスレーションによって実行しなければなりません。 b に対する回答が "YES"の場合には，広範囲にわたって DEC の移行サービスを利用できます。]

11. a. アプリケーションをテストするために必要なリグレッション・テストが準備されていますか? ☐ YES ☐ NO

    b. 準備されている場合には，DEC Test Manager が必要ですか? ☐ YES ☐ NO

    [a に対する回答が "YES"の場合には，これらのリグレッション・テストの移行を考慮しなければなりません。OpenVMS AXP の初期リリースでは，DEC Test Manager は提供されません。リグレッション・テストでこのプロダクトが必要な場合には，DEC の担当者にご連絡ください。]

VAX アーキテクチャへの依存

12. a. アプリケーションで H 浮動小数点データ型を使用しますか? ☐ YES ☐ NO

    b. アプリケーションで D 浮動小数点データ型を使用しますか? ☐ YES ☐ NO

    c. アプリケーションで D 浮動小数点を使用する場合，56 ビットの精度 (16 桁の有効桁数) が必要ですか，または 53 ビットの精度 (15 桁の有効桁数) で十分ですか? ☐ 56 ビット ☐ 53 ビット

    [a に対する回答が "YES"の場合には，H 浮動小数点の互換性を維持するためにアプリケーションをトランスレートするか，またはデータを G 浮動小数点，S 浮動小数点，または T 浮動小数点フォーマットに変換しなければなりません。b に対する回答が "YES"の場合には，アプリケーションをトランスレートして，VAX での D 浮動小数点の完全な 56 ビットの精度の互換性を維持するか，AXP システムが準備している 53 ビットの精度の D-floatin フォーマットを使用するか，またはデータを G 浮動小数点，S 浮動小数点，または T 浮動小数点のいずれかのフォーマットに変換しなければなりません。]

13. a. アプリケーションで大量のデータ，またはデータ構造を使用しますか? ☐ YES ☐ NO

    b. データがバイト，ワード，またはロングワードでアラインされていますか? ☐ YES ☐ NO

    [a に対する回答が "YES"であり，b に対する回答が "NO"の場合には，AXP の最適な性能を実現するために，データを自然なアラインメントにすることを考慮しなければなりません。多くのプロセスで共有されるグローバル・セクションにデータが格納されている場合や，メイン・プログラムと AST との間でデータが共有される場合には，データを自然なアラインメントにしなければなりません。]

14. コンパイラがデータをアラインする方法に関して，アプリケーションで何らかの仮定を設定していますか (つまり，データ構造がパックされていること，自然なアラインメントにされていること，ロングワードでアラインされていることなどをアプリケーションで仮定していますか)? ☐ YES ☐ NO

[回答が "YES"である場合には， AXP プラットフォームでのコンパイラの動作と， VAX プラットフォームでのコンパイラの動作の違いから発生する，移植性と相互操作性の問題を考慮しなければなりません。コンパイラ・スイッチによって，アラインメントは強制的に設定されるため，データ・アラインメントに関するコンパイラの省略時の設定はさまざまです。通常， VAX システムでは，省略時のデータ・アラインメントはパック形式のアラインメントですが， AXP コンパイラの省略時の設定は，可能な限り自然なアラインメントです。]

15. a. アプリケーションで，ページ・サイズが 512 バイトであると仮定していますか? ☐ YES ☐ NO

b. アプリケーションで，メモリ・ページがディスク・ブロックと同じサイズであると仮定していますか? ☐ YES ☐ NO

[a に対する回答が "YES"の場合には， Alpha AXP のページ・サイズに対応できるように，アプリケーションを準備しなければなりません。 Alpha AXP のページ・サイズは 512 バイトよりはるかに大きく，各システムで異なります。したがって，ページ・サイズを明示的に参照することは避け，可能な限り，メモリ管理システム・サービスと RTL ルーチンを使用してください。 b に対する回答が "YES"の場合には，ディスク・セクションをメモリにマッピングする$CRMPSC システム・サービスと，$MGBLSC システム・サービスに対するすべての呼び出しを確認し，これらの仮定を削除しなければなりません。]

16. アプリケーションで， OpenVMS システム・サービスを呼び出しますか? ☐ YES ☐ NO

特に，次の操作を実行するサービスを呼び出しますか?

a. グローバル・セクションを作成，またはマッピングするシステム・サービス (たとえば$CRMPSC, $MGBLSC,$UPDSEC) ☐ YES ☐ NO

b. ワーキング・セットを変更するシステム・サービス (たとえば$LCKPAG，$LKWSET) ☐ YES ☐ NO

c. 仮想アドレスを操作するシステム・サービス (たとえば$CRETVA, $DELTVA) ☐ YES ☐ NO

[これらの質問に対する回答が "YES"の場合には，コードを調べ，必要な入力パラメータが正しく指定されているかどうかを判断しなければなりません。]

17. a. アプリケーションで複数の協調動作プロセスを使用しますか? ☐ YES ☐ NO

使用する場合:

b. プロセスの数? _____

c. 使用するプロセス間通信方式? _____

☐ $CRMPSC ☐ メールボックス ☐ SCS ☐ その他
☐ DLM ☐ SHM, IPC ☐ SMG$ ☐ STR$

d. 他のプロセスとの間でデータを共有するために，グローバル・セクション ($CRMPSC) を使用する場合には，どのような方法でデータ・アクセスの同期をとりますか? _____

[この質問は，明示的な同期を使用しなければならないのかどうかを判断し，アプリケーションの各要素間で，同期を保証するのに必要な作業レベルを判断するのに役立ちます。一般に，高いレベルの同期方式を使用すれば，アプリケーションをもっとも容易に移行できます。]

18. アプリケーションは現在，マルチプロセッサ (SMP) 環境で実行されていますか? ☐ YES ☐ NO

[回答が "YES"の場合には，アプリケーションはすでに適切なプロセス間同期方式を採用している可能性が高いと言えます。]

19. アプリケーションで， AST(非同期システム・トラップ) メカニズムを使用しますか? ☐ YES ☐ NO

[回答が "YES"の場合には， AST とメイン・プロセスが，プロセス空間でデータ・アクセスを共有するかどうかを判断しなければなりません。共有する場合には，明示的にこのようなアクセスの同期をとらなければなりません。]

20. a. アプリケーションに条件ハンドラが含まれていますか? ☐ YES ☐ NO

b. アプリケーションで，算術演算例外の即時報告が必要ですか? ☐ YES ☐ NO

[Alpha AXP アーキテクチャでは，算術演算例外はただちに報告されません。条件ハンドラが条件を修正しようとするときに，例外の原因となった命令シーケンスを再起動する場合には，ハンドラを変更しなければなりません。]

21. アプリケーションは特権モードで実行されますか，または SYS.STB に対してリンクされますか? ☐ YES ☐ NO

その場合は理由を示してください。 _____

[アプリケーションが OpenVMS エグゼクティブに対してリンクされるか，または特権モードで実行される場合には，ネイティブな AXP イメージとして実行されるように，アプリケーションを変更しなければなりません。]

22. 独自のデバイス・ドライバを作成しますか? ☐ YES ☐ NO

[OpenVMS AXP の初期リリースでは，ユーザ作成デバイス・ドライバはサポートされません。この機能が必要な場合には， DEC の担当者にご連絡ください。]

23. アプリケーションで，接続/割り込みメカニズム (connect-to-interrupt) を使用しますか? ☐ YES ☐ NO

使用する場合には，どのような機能を使用しますか? _____

[接続/割り込みは， OpenVMS AXP システムでサポートされません。この機能が必要な場合には， DEC の担当者にご連絡ください。]

24. アプリケーションで，機械語を直接作成または変更しますか? ☐ YES ☐ NO

[OpenVMS AXP では，命令ストリームに書き込まれた命令が，正しく実行されることを保証するには，多大な注意が必要です。]

25. アプリケーションのどの部分が性能にもっとも大きな影響を与えますか? それは入出力ですか，浮動小数点ですか，メモリですか，リアルタイムですか (つまり，割り込み待ち時間)。 _____

[この質問は，アプリケーションの各部分に対する作業に優先順位をつけるのに役立ち，お客様にとってもっとも意味のある性能向上を， DEC が計画するのに役立ちます。]

B

移行計画の例

この付録に示す移行計画は，現実のアプリケーションに対する計画ではありませんが，お客様のアプリケーションのために作成した，実際の移行計画にもとづいています。

Migration Plan for Omega-1

Omega Corporation

B.1 エグゼクティブ・サマリ

Omega-1 はデータのアクセス，分析，管理，および報告のための，企業全体を網羅する情報システムです。

Omega-1 のソース・コードは 400 万行以上のサイズです。ソース・コードの大部分は C プログラミング言語で作成されており，多彩なプラットフォームで共通に使用でき，移植性がかなり高いと考えられます。

しかし， Omega-1 には各プラットフォーム固有のルーチンが含まれており， VAX プラットフォームを対象として，VAX CとVAX MACROで実現されています (この部分は約 350,000 行のコードです)。これらのルーチンは，第 B.2.4 項で説明するように， VAX アーキテクチャ固有の機能に依存する多くの問題を抱えており，移行を成功に導くために解決しなければなりません。これらの問題を解決するには，設計変更も含めて，かなり大量の作業が必要ですが，これらの問題があるからといって， 1992 年 12 月に Omega-1 の顧客への納品が遅れることはないと考えられます。

Omega-1 は，多くの DEC プロダクトとサード・パーティ・プロダクトへの接続をサポートします。これらのプロダクトの一部は， OpenVMS AXP が最初に出荷される時点では， AXP プラットフォームで提供されませんが， Omega-1 を供給する業者である Omega Corporation は，そのときまでに基本的な Omega-1 を移行することを約束しています。

DEC サービスは，この計画の詳細記述に従って，移行プロジェクトの最初から最後まで， Omega Corporation にサポート・サービスを提供します。 Omega Corporation に提供されるサポート計画をまとめると，次のようになります。

- Omega Corporation で AXP アプリケーションを開発するためのハードウェア・ツールとソフトウェア・ツール

- 品質保証テストのための技術的な支援
- AXP Migration Center での AXP システムのアクセス
- DEC の技術者への電話による連絡

  この技術者は Alpha AXP に関する技術情報を提供し，クロス開発ツールをサポートし， Omega Corporation が報告した DEC ソフトウェア・プロダクトに関する問題を解決する責任者となります。
- Omega Corporation の開発者を対象とした 3 日間の技術セミナー (Omega Corporation での訪問セミナー)

もう 1 つの問題として，出荷日より前にプロダクトに対して適切なフィールド・テストを実行するために， Omega にハードウェアを提供しなければならないという問題があります。通常， Omega の開発者は実際にプロダクトを出荷する前に， 30 ～ 40 ヵ所の顧客システムで 4 ヵ月間，フィルード・テストを行います。しかし，この規模のフィールド・テストを実行するために必要な台数の AXP システムを入手できるかどうかが不明です。

B.2 技術分析

Omega Corporation は DEC サービスと協力して技術分析を行いました。

B.2.1 アプリケーションの特性

Omega-1 は大部分の VAX プラットフォーム，および他社のプラットフォームで実行されます。このアプリケーションは 3 つの層で構成されており，各層は， VAX 環境の固有の機能を利用する場合も，利用しない場合もあります。しかし，特定のハードウェア構成や装置に対する直接的な依存はありません。

大部分の機能はアプリケーション層で提供され，この層にはユーザ・インターフェイス，基本的なデータ管理ツール，および Omega の第 4 世代言語 (4GL) が含まれています。 Omega-1 の基本プロダクトは， DEC またはサード・パーティのレイヤード・プロダクトに依存しません。

Omega-1 の追加プロダクトは，基本プロダクトを基礎にしたレイヤード・プロダクトであり，拡張されたデータ管理機能または通信機能を備えています。これらのオプションは， DEC またはサード・パーティのレイヤード・プロダクトに依存し，基礎となるプロダクトがリリースされるときに提供されます。

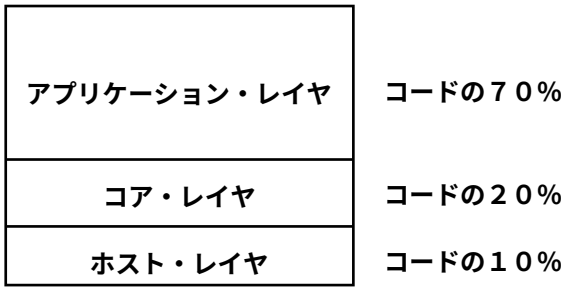
B.2.2 ソフトウェア・アーキテクチャ

Omega-1 は図 B–1 に示すように，層に分かれています。この層構造によって，高いレベルのソフトウェア移植性を実現しています。これは，システムの約 10 パーセントだけが特定の実現方法に依存し，このコードはすべて 1 つのモジュール群，つまりホスト層に含まれているからです。

アプリケーション層とコア層は，すべてのソース・ファイルを再コンパイルおよび再リンクするだけで AXP プラットフォームで実行できると考えられます。そのための前提条件は， Omega ソフトウェア開発ツールを正しく移行できることだけです。しかし，これらの開発ツールも移植可能であると考えられ， OpenVMS AXP のランタイム・ライブラリと C コンパイラにのみ依存します。

ホスト層では， VAX ハードウェアに依存している一部のモジュールを再開発するなど，多くの変更が必要になります。

図 B–1 Omega-1 の層構造



JRD-5174A

- アプリケーション層

  この層はシステムの大部分を構成する層であり，多くのプラットフォームで類似した方法で実現されるため，移植可能であると，考えられます。

- コア層

  この層は， Omega が設計したサービス群を作成する層であり，これらのサービス群は，各プラットフォームで Omega-1 の特殊な必要条件に準拠するように設計されています。特に， Omega-1 の「仮想オペレーティング・システム」の各構成要素はこの層に存在します (各構成要素は移植可能な方法で作成できます)。構成要素としては，上位レベルの入出力，上位レベルのメモリ管理，文字ベースのウィンドウ・システム，実行環境も含む 4GL コンパイラがあります。この層は移植可能です。

- ホスト層

  この層は，固有のオペレーティング・システム要素に対するインターフェイスを提供し，ハードウェア・アーキテクチャに依存する可能性があります。この層の構成要素は次のとおりです。

  - 入出力操作
  - イメージのロードとアンロード
  - メモリ管理
  - 多少のスレッド管理
  - ターミナル・サービス
  - ウィンドウ・インターフェイス

  ホスト層は，各プラットフォームで異なります。 VAX のホスト層は，VAX C と VAX MACROで作成されています。この層は，移行プロジェクトが中心的に対処しなければならない，大部分の問題を含んでいます。

B.2.3 イメージ分析の結果

Omega-1 は再コンパイルおよび再リンクされますが，ホスト層の 26 のイメージを分析するために，まず VAX Environment Software Translator (VEST) を使用しました。これらのイメージの多くには， D 浮動小数点データ型に依存する命令が含まれています。このデータ型はVAX Cの省略時の設定です。 Omega の技術者が D 浮動小数点を別の浮動小数点フォーマットに移行することを決定した場合には，VAX と AXP が混在する VMS クラスタ環境で，データ・ファイルの互換性に関する問題が発生することに注意しなければなりません。

表 B–1 は，イメージ分析で各イメージによって生成されたエラー・メッセージを示しています。

表 B–1 イメージ分析の結果

イメージ名	VEST が検出した%コード	検出した主な要素
OMEGADEV60	-Fatal errors- no code found	VEST-F-PROTISD VEST-F-ISDALIGN VEST-F-ISDMIXED VEST-W-STACKMATCH パック 10 進命令
OMECRTL	92%	VEST-W-STACKMATCH VEST-W-STKUNAL パック 10 進命令
PSCN	64%	VEST-W-STKUNAL VEST-W-STACKMATCH
PVSN	65%	VEST-W-STKUNAL VEST-W-STACKMATCH

次のリストは， Omega イメージを分析した結果，検出された主な要素を説明しています。

- PROTISD は，ユーザ作成システム・サービス・ベクタであり，イメージに 1 つ以上のユーザ作成システム・サービスが含まれることを示します。この問題は，ネイティブな Alpha AXP コンパイラを使用して，コードをコンパイルするときに自動的に処理されます。

- ISDALIGN は，イメージ・セクションが 64KB 境界にアラインされていないことを示します。別の VEST 分析を実行する前に， 64KB のページを使用してイメージを再リンクしなければなりません。リンカは移行プロセスでこの問題を処理します。
- ISDMIXED は，互換性のない VAX イメージ・セクションが，同じ 64KB ブロックにマッピングされたことを示します。リンカは移行処理でこの問題を処理します。
- STKUNAL は警告であり，コード・ブロックがロングワードでアラインされたスタックを，アラインされないスタックに変更したことを示します。この結果，性能が低下します。 Omega の技術者は VEST 分析からログを調べます。
- STACKMATCH は，スタックが特定の時点でバランスのとれない状態になる可能性のあることを示します。 Omega の技術者は， VEST 分析からログを調べます。
- パック 10 進命令はソフトウェアによってのみサポートされ，ハードウェアではサポートされません。これらの命令は，アプリケーションの性能を低下させる可能性があります。 Omega の技術者は，パック 10 進コードを調べます。

B.2.4 ソース分析の結果

前に説明したように，アプリケーション層とコア層の移行はかなり簡単です。しかし， Omega-1 のホスト層には VAX に依存する部分が数多く含まれています。 Omega の技術者と検討した結果，アーキテクチャに依存する部分は次のとおりであることがわかりました。

- データ・アラインメント

  Omega-1 ソフトウェアには，アラインされていないパブリック・データ構造が含まれています。ソース・コードを移植可能にするために， Omega の技術者は DEC C コンパイラの/NOMEMBER_ALIGN 修飾子を使用して，このコードをコンパイルします。

- データ型

Omega-1 では，広範囲にわたって浮動小数点演算とデータ・ファイルを使用します。 VAX バージョンでは，すべての操作に対して D-56 フォーマットを使用しますが，このフォーマットはネイティブな Alpha AXP 命令セットでは実現されていません。 VAX と AXP が混在するクラスタを使用する顧客の場合， D 浮動小数点フォーマットをそのまま使用することが適切です。 Omega では，AXP で提供される D 浮動小数点の 53 ビット・バージョン (56 ビットではない) を使用した結果，浮動小数点の精度が少し失われるものの，特に問題がないと判断しました。

しかし，他の大部分の Omega-1 システムでは IEEE 浮動小数点フォーマットを使用しており，これらのフォーマットは Alpha AXP 命令セットで完全にサポートされます。 IEEE フォーマットを再度実現するには，異なる修飾子を使用して再コンパイルするだけですみますが，その後， OpenVMS VAX ユーザは移行処理の一部として，ファイル内のすべての浮動小数点データを新しいフォーマットに変換しなければなりません。

- 読み込み/書き込み/変更の不可分性

  共有変数に対する操作を明示的な同期方式によって保護しなければならないかどうかを判断するには，いくつかの AST ルーチンを調べなければなりません。この部分には大きな問題はありません。

- バイト操作とワード操作の粒度

  Omega-1 ソフトウェアには，自然なクォドワード境界にアラインされないデータを保護するラッチがあります。 DEC の技術者は Omega の技術者とこの問題に関して検討し，解決方法を模索するためにコード・サンプルを調べました。その結果，共有データ宣言とコンパイラ・ディレクティブを使用することにより，コンパイラがこの種のアクセスを処理できると判断しました。

- VAX ページ・サイズ

  ホスト層には，アプリケーションのかわりにメモリ管理を処理するいくつかのルーチンがあります。既存のアルゴリズムでは，実際にページ・サイズが 512 バイトである必要はありませんが，それでもページ・サイズが 512 バイトとしてハード・コーディングされています。システム起動時にシステム・ページ・サイズを調べ，メモリ管理操作で必要な演算に対して正しいシステム・ページ・サイズを使用するようにモジュールを変更すれば，メモリ管理機能を正しく実行できます。

- VAX プロシージャ呼び出し規則

  ユーザ割り込みが発生したときに呼び出し履歴を判断するために， Omega-1 は呼び出しフレーム・スタックを「追跡」します。 Omega-1 はコードの中で重要でない領域を処理しているときに，呼び出しフレームの 1 つの戻りアドレスを変更することにより，制御の流れを変更します。つまり，割り込みを受け付けます。この処理の大部分は， SYS$UNWIND が実行する処理によく似ていますが， Omega-1 では例外ハンドラのかわりに AST を使用します。

  Omega-1 には， VAX 呼び出し規則に依存する多くの機能が含まれています。たとえば， Omega では setjmp()と longjmp()を独自の方法で実現しています。これらの関数はVAX MACROで作成されており，オペレーティング・システム・コード内で分離されています。 Omega は VAX 呼び出し規則への依存に対処するために，これらの関数を再開発します。

- 例外処理

  Omega-1 は無効な，または例外をおこした浮動小数点演算に対し，統計的に値を補い修正します。ここで問題となるのは， Alpha AXP アーキテクチャで実際にフォルトが発生した命令を現在の設計でデコードできるかどうかということです。 Alpha AXP アーキテクチャでは，例外トラップはただちに報告されません。

- VAX 命令セットとコード生成

  ホスト層にはコード・ジェネレータがあり，プラットフオームにとってネイティブな命令をメモリに書き込み，それを 4GL 言語の一部として実行します。このコード・ジェネレータは多くの作業を実行しますが，特にデータベース入出力操作を実行するために「分散/収集」コードを生成します。移行プロジェクトの初期段階では，コード・ジェネレータとの間で「プラグ互換性」を維持した移植可能なインタプリタを使用できます。しかし， Alpha AXP 命令を生成する新しいジェネレータ・バージョンを最終的に実現しなければなりません。

B.3 中間目標と成果物

Omega-1 の基本プロダクトの出荷予定日は 1992 年 12 月です。表 B–2 は，基本プロダクト・プロジェクトの主な中間目標と成果物を示しています。各成果物についての詳しい説明は，第 B.4 節を参照してください。

表 B–2 中間目標と成果物

中間目標	責任者	DEC の役割	終了日
Omega-1 ライン・モード・プロダクト	Omega/Digital	コンサルティング	1991 年 11 月
新しいクロス・イメージ・ブリッジ	Omega/Digital	コンサルティング	1991 年 12 月
浮動小数点に関する判断	Omega	—	1991 年 12 月
Omega-1 例外モジュール	Omega/Digital	コンサルティング	1992 年 1 月
コード・ジェネレータの実現開始	Omega/Digital	コンサルティング	1992 年 1 月
アプリケーション層の構築	Omega/Digital	コンサルティング	1992 年 1 月
コード・ジェネレータのテスト	Omega/Digital	テスト・スクリプトの実行	1992 年 3 月
アプリケーションのテスト	Omega/Digital	テスト・スクリプトの実行	1992 年 5 月
Motif ユーザ・インターフェイスの実現とテスト	Omega/Digital	テスト・スクリプトの実行	1992 年 7 月
Omega QA とフィールド・テストの開始	Omega	訪問サポート	1992 年 12 月
出荷日	Omega	—	1992 年 12 月

B.4 技術的なアプローチ

この後の節では，この移行プロジェクトの各中間目標を実現するために採用したアプローチについて，詳しく説明します。

B.4.1 ライン・モード・プロンプト

最初の中間目標は Omega-1 にライン・モード・プロンプトを導入し，実行のために意味のある Omega-1 プログラム名とコマンド・シーケンスを入力することです。この目標を達成すれば，開発ツールとランタイム・ライブラリの基本的な機能を示すことができます。この時点で，次の要素を除き，ホスト層は正しく機能します。

- 4GL 機能に対して，コード・ジェネレータのかわりに Omega-1 インタプリタを使用します。
- イメージ・ブリッジは一時的な実現です。
- 例外処理は不完全です。

さらに，コア層も正しく機能し (ウィンドウ・ユーザ・インターフェイスを除く)，少なくとも 1 つの Omega-1 アプリケーションを試すことができます。この作業は DEC からのサポートのもとに， Omega VMS ホスト・グループが実行します。

B.4.2 イメージ・ブリッジ

Omega-1 には，イメージ間ですべてのジャンプをディスパッチする，中心的なブリッジ・ルーチンがあります。このため， Omega-1 はアンロードされているイメージ内のルーチンを呼び出し，これらのイメージを動的にロードすることができます。また，ブリッジ・ルーチンの採用により， Omega-1 ではイメージをアンロードできます。

OpenVMS AXP オブジェクト・ファイルのフォーマットとイメージ起動ルーチンはすでに設定されており， OpenVMS AXP のブリッジの実現は， OpenVMS VAX の場合と同様の方法で実行できます。

この作業は， DEC のサポートのもとに Omega VMS ホスト・グループが実行し，必要な変更は 1991 年 12 月までに終了できます。

B.4.3 浮動小数点フォーマットに関する判断

Omega-1 とそれを利用する多くの顧客は， D-56 浮動小数点データ型に依存します。速度を向上するために D-56 を IEEE の T 浮動小数点に変更できますが，そのためにはすべてのデータに対して， 1 つのフォーマットから別のフォーマットにデータを変換しなければなりません。

この判断は， Omega がビジネス上の判断として下さなければならず， 1991 年末までに判断します。

B.4.4 Omega-1 の完全な例外処理

次に解決しなければならない大きな問題は， OpenVMS AXP で Omega-1 の例外処理をどのように実行するかということです。ファイルのオープン時に発生する実行時アクセス違反など，一般的な例外処理は Omega-1 の例外ハンドラがトラップし，その後，コール・フレームに対して「スタック追跡」を実行することにより処理されます。 Omega の開発者は OpenVMS AXP の新しい呼び出し規則に対応するために必要な変更を行い， DEC C の "setjmp"と "longjmp"機能を使用します。

浮動小数点例外処理に関しては，設計を変更し，再度実現することが必要です。正しい機能を実現するために多くのオプションを選択できますが，さらに性能も考慮して，最適な方法を判断しなければなりません。 Omega の開発者は予定日までにこの問題を解決できるでしょう。

B.4.5 コード・ジェネレータの実現の開始

ホスト層を完全に実現するために必要な最後の構成要素はコード・ジェネレータです。コード・ジェネレータを実現するために， Omega の開発者はネイティブな Alpha AXP 命令セットの完全な定義を必要とします。コード・ジェネレータの開発作業は， Omega の OpenVMS ホスト・グループが完全に行います。 DEC は必要に応じて電話によるサポートを提供します。

B.4.6 アプリケーションの構築

ホスト層が完全な機能を実現した後，1992 年 1 月にアプリケーションを構築できます。これらのアプリケーションはきわめて厳密な移植標準規格に従って C で作成されているため，再コンパイルするだけで実行できると考えられます。 DEC は Omega の開発者に対して，ツールとコンパイラの問題点に関する電話によるサポートを提供します。

B.4.7 コード・ジェネレータのテスト

DEC は 1992 年 3 月に AXP システムで Omega-1 コード・ジェネレータをデバッグし，機能テストを実行するために，必要な援助を提供します。 Omega の開発者は Alpha AXP Migration Center の AXP システムでテスト環境を設定し，テストの実行方法を DEC の技術者に教育し，初期テストを監視します。その後， Omega は DEC の技術者がテストを実行できるように，テスト・スクリプトとデータを郵送します。

B.4.8 完全なアプリケーションのテスト

Omega は数多くのリグレッション・テスト・ストリームを管理しており，移植が正しく実行されたかどうかを検証するために， AXP システムでこれらのテストを実行しなければなりません。このテストを実行できる状態になったときに，Omega が AXP システムをまだ入手していない場合には， DEC は DEC のシステムを使用して Omega-1 基本システム・アプリケーションに対してリグレッション・テストを実行しなければなりません。

このために， Omega の開発者は DEC ラボラトリの AXP システムでテスト環境を設定し，リグレッション・テストの実行方法に関して DEC の技術者を教育し，初期テストを監視します。その後， Omega は DEC の技術者がテストを実行できるようにするために，テスト・スクリプトとデータを郵送します。 DEC はテスト結果を Omega に戻します。この作業は 1992 年 4 月に開始され， 5 月末までに終了します。

B.4.9 DECwindows Motif ユーザ・インターフェイス

開発者が Motif ユーザ・インターフェイスをテストできるように， Omega はフィールド・テストの前に Motif 開発ツール・キットを入手しておかなければなりません。

B.4.10 Omega の品質保証とフィールド・テスト

Omega は最終プロダクトを検証するために多くのテスト・ストリームを管理しています。これらのテストは，フィールド・テストを開始する直前に実行されます。これらのテストは完全な AXP システムで実行しなければなりません。

この時点で， AXP システムでのみ明らかになる性能上の特定の問題を解決するために，いくつかのテストと最適化が必要です。 DEC はこれらの作業を実行するために必要なエンジニアリング・サポートを提供します。

B.5 依存とリスク

Omega-1 を正しく出荷できるかどうかを左右するおもなリスクは，品質保証とフィールド・テスト・プロセスに関係しています。開発者は通常， 30 〜 40 の顧客システムに対してフィールド・テストを実行し，これらのテストが終了するまでに約 4 ヵ月かかります。 Omega と DEC アカウント・チームは， Omega が最低限の条件を満足するテスト・プログラムを実行し， 1992 年 12 月までに終了できる方法を判断しなければなりません。

次のリストは， Omega-1 基本プロダクトのソフトウェアが特定のアーキテクチャに依存する部分を示しています。

- DEC C for OpenVMS AXP コンパイラ
- VAX MACRO–32 Compiler for OpenVMS AXP
- MACRO–64 Assembler for OpenVMS AXP
- OpenVMS デバッガ
- DEC C ランタイム・ライブラリ

- OpenVMS ランタイム・ライブラリ (LIB$)
- 画面管理ライブラリ (SMG$)
- DECTPU

Omega-1 アプリケーションはまた，DEC またはサード・パーティのデータ管理オプションまたはネットワーキング・オプションへのアクセスも提供します。次の例を参照してください。

- Omega/Graph は，CDA ツールを使用して DDIF フォーマットでグラフィック・イメージを作成できます。
- Omega/Access は Rdb/VMS または ORACLE データベースをアクセスできます。
- Omega/Share と Omega/Connection は，ULTRIX Connection (UCX) を使用して TCP/IP を介してリモート・データをアクセスできます。

B.6 必要な資源

第 B.3 節に示した計画をサポートするために使用される DEC の資源は，表 B–3 に示すとおりであり，この後の節で詳しく説明します。

表 B–3 DEC サポートのまとめ

資源	時期	作業内容	作業レベル
訪問トレーニング	91 年 12 月	トレーニング	1 人の技術者が 3 日間
電話によるサポート	91 年 12 月～ 92 年 8 月	一般サポート	1 人の技術者が 1 週間に 8 時間
技術支援	92 年 3 月	コード・ジェネレータのテスト	1 人のフルタイム技術者が 2 週間，ハーフタイム技術者が 4 週間

(次ページに続く)

表 B–3 (続き) DEC サポートのまとめ

資源	時期	作業内容	作業レベル
AXP ハードウェア	92 年 3 月	コード・ジェネレータのテスト	1 週間に 5 日間の割合で 2 週間，1 週間に 2 日間の作業を 4 週間
技術支援	92 年 4 月～ 5 月	アプリケーションのテスト	1 人の技術者がハーフタイムで 8 週間
AXP ハードウェア	92 年 4 月～ 5 月	アプリケーションのテスト	1 週間に 2 日ずつの割合で 8 週間
訪問サポート	92 年 1 月～ 8 月	Omega QA	1 ヵ月間に 1 週間

B.6.1 ハードウェア

Omega は通常の開発作業に影響を与えずに移行作業を終了するまでに，システムを DEC から借りなければなりません。

Omega-1 基本プロダクトは開発システム (VAX 6000 Model 550) で RA90 ドライブを複数使用します。完全なアプリケーションを構築する場合，ディスク空間が問題になる可能性があります。

B.6.2 訪問トレーニング

DEC は 1991 年 12 月に開始された Omega 移行プロジェクトを積極的にサポートするために，Omega のアプリケーション層開発者とコア層開発者を対象にして 3 日間の AXP 技術セミナーを開催します。

B.6.3 電話によるサポート

1992 年 1 月から 3 月までに， Omega の開発者は DEC が提供するプラットフォームでクロス・ツールを使用して，ホスト層を移行する作業を行います。この期間だけでなく，移行作業全体にわたって， Omega は DEC のソフトウェア技術者から電話によるサポートを受けることができます。 DEC の担当技術者は Omega の問題に関して 1 週間に約 8 時間費やし，バグの報告を処理し，クロース・ツールをサポートします。

B.6.4 テストの支援

DEC は Omega-1 アプリケーションの複数のテスト段階をサポートします。

B.6.4.1 コード・ジェネレータのテスト

コード・ジェネレータをテストするために， 3 月の最初の 2 週間に 1 人のフルタイムの DEC 技術者が必要です。この期間に， Omega の開発者が DEC の技術者を教育し，初期テストを実行します。その後の 4 週間に DEC の担当技術者は就業時間の 50%を費やしてフォローアップ・テストを実行し，テスト結果を Omega に報告します。

コード・ジェネレータのテストでは，最初の 2 週間に AXP システムをほとんどフルタイムで使用しなければなりません。その後の 4 週間には， 1 週間に 2 日ずつの割合で AXP ハードウェアを使用しなければなりません。

B.6.4.2 アプリケーションのテスト

アプリケーションのテストは 1992 年 4 月と 5 月に DEC で実行され，リグレッション・テストを実行し，テスト結果を Omega に報告するために， 1 人の技術者の勤務時間の約 50%を必要とします。この作業には， 8 週間のテスト期間中に 1 週間に 2 日ずつの割合で AXP ハードウェアを使用しなければなりません。

B.6.4.3 Omega の品質保証

DEC の技術者は Omega で訪問サポートを提供するために， 1992 年 9 月～ 11 月の期間中に 15 日間， Omega に出向します。

用語集

CISC
: **複雑命令セット・コンピュータ**を参照。

IIF
: VAX イメージ間のインターフェイスに関する情報を記述した ASCII ファイル。 VEST は IIF を使って他のイメージに対する参照を解決し，適切なリンクを作成する。

PALcode
: **特権付きアーキテクチャ・ライブラリ**を参照。

RISC
: **縮小命令セット・コンピュータ**を参照。

Translated Image Environment (TIE)
: **トランスレートされたイメージ**の実行をサポートするネイティブな AXP 共有可能イメージ。 TIE はトランスレートされたイメージとネイティブな AXP システムとのすべてのやりとりを処理する。また， VAX の状態を管理し，例外処理や AST の実行要求，複雑な VAX 命令など， VAX の機能をエミュレートし，トランスレートされていない VAX 命令を解釈することにより，トランスレートされたイメージに対して OpenVMS VAX に類似した環境を提供する。

VAX Environment Software Translator (VEST)
: ソフトウェア移行用のツールであり， VAX の実行可能イメージと共有可能イメージを， AXP システムで実行されるトランスレートされたイメージに変換する。**トランスレートされたイメージ**を参照。

VEST
: VAX Environment Software Translatorを参照。

アラインされたデータ (aligned data)
アラインメントの用件を満たすデータを，**アラインされたデータ**と呼ぶ。

アラインメント (alignment)
自然なアラインメントを参照。

イメージ・セクション
イメージを仮想メモリに割り当てるときの単位となる，同じ属性を持つプログラム・セクションの集合。この場合属性とは，たとえば読み込み専用アクセス属性，読み込み/書き込みアクセス属性，固定アドレス属性，再配置可能属性などである。

インターロック命令
インターロック命令は，マルチプロセシング環境で 1 つの中断されない操作として完全な結果を保証できる方法で動作を実行する。インターロック命令が終了するまでの間，他の衝突する可能性のある操作はブロックされるため，インターロック命令は性能を低下させる可能性がある。

書き込み可能グローバル・セクション
プロセス間通信で使用するためにシステム内のすべてのプロセスが使用できるデータ構造 (たとえば FORTRAN のグローバル・コモン) や共有可能イメージ・セクション。

クォドワード
任意のアドレッシング可能なバイト境界から始まる連続した 4 ワード (64 ビット)。各ビットには右から左に 0 〜 63 の番号が付けられる。クォドワードのアドレスは下位ビット (ビット 0) を含むワードのアドレスである。アドレスを 8 で割り切れる場合には，クォドワードは**自然にアラインされる**。

クォドワード粒度
メモリ・システムの特性であり，隣接する**クォドワード**を異なるプロセスまたはプロセッサが同時に個別に書き込むことができる特性。

クロス開発
1 つのシステムで実行されるツールを使用して，別のシステムを対象としたソフトウェアを作成する処理。たとえば， VAX システムで実行されるツールを使用して， AXP システムのためのコードを作成する処理。

互換性

あるコンピュータ・システム (OpenVMS VAX) のために作成されたプログラムを別のシステム (たとえば OpenVMS AXP) で実行できる能力。

自然なアラインメント

データ・アドレスをデータ・サイズ (バイト数) で割り切れるメモリ内のデータ。たとえば，自然にアラインされたロングワードは 4 で割り切れるアドレスを持ち，自然にアラインされたクォドワードは 8 で割り切れるアドレスを持つ。構造のすべてのメンバが自然にアラインされている場合には，その構造も**自然にアラインされる**という。

ジャケット・ルーチン

1 つの呼び出し規則から別の規則にプロシージャ呼び出しを変換するプロシージャ。たとえば，トランスレートされた VAX イメージ (VAX 呼び出し規則を使用するイメージ) とネイティブな AXP イメージ (AXP 呼び出し規則を使用するイメージ) との間で呼び出しを変換する。

縮小命令セット・コンピュータ (RISC)

複雑さが削減された命令セットを使用するコンピュータ。ただし，命令の数が必ずしも削減されているとは限らない。 RISC アーキテクチャは通常，特定の操作を実行するためにCISCアーキテクチャより多くの命令を必要とする。これは，各命令が CISC 命令より少ない作業しか実行しないからである。

同期

マルチプロセシング環境や共有データを使用するユニプロセシング環境で操作するときに，きちんと定義された予測可能な結果が得られるように，一部の共有資源に対するアクセスを制御する方法。

同時実行/並列処理

複数のエージェントが共有オブジェクトに対して操作を同時に実行すること。

特権付きアーキテクチャ・ライブラリ (PAL)

特定のオペレーティング・システム固有の命令を実行するための呼び出し可能ルーチンを登録したライブラリ。特殊な命令がルーチンを呼び出し，これらは中断せずに実行される。

トランスレーション

VAX バイナリ・イメージを AXP イメージに変換する処理。変換されたイメージは AXP システムで TIE の援助によって実行される。変換は静的処理であり，できるだけ多くの VAX コードがネイティブな Alpha AXP 命令に変換される。実行時に変換されなかった VAX コードに対しては， TIE が最終的に解釈する。

トランスレートされたイメージ

VAX イメージのオブジェクト・コードの**トランスレーション**によって作成された AXP 上の実行可能イメージまたは共有可能イメージ。トランスレートされたイメージは，トランスレーションのもとになる VAX イメージと同じ機能を実行し，**トランスレートされたコード**とオリジナル・イメージの両方を含む。VAX Environment Software Translatorを参照。

トランスレートされたコード

トランスレートされたイメージ内の AXP オブジェクト・コード。トランスレートされたコードとしては，次のコードがある。

- 元のイメージの対応する VAX コードの動作を再現する AXP コード
- Translated Image Environment (TIE)の呼び出し

ネイティブなイメージ

OpenVMS AXP コンパイラ， OpenVMS AXP リンカを使用して作られた OpenVMS AXP 上の実行可能イメージ，または共有可能イメージを，**トランスレートされたイメージ**に対してこう呼ぶ。

バイト粒度

メモリ・システムの特性であり，隣接するバイトを異なるプロセスまたはプロセッサが同時に独立して書き込むことができる特性。

不可分な操作

AST(非同期システム・トラップ) サービス・ルーチンなど，他のシステム・イベントによって中断することができない操作。**不可分な操作**は他のプロセスにとって 1 つの操作であるかのように見える。不可分な操作が開始された後，その操作は中断されずに，必ず最後まで終了する。

読み込み/変更/書き込み (リード・モディファイ・ライト) 操作は通常，RISC マシンの命令レベルでは不可分な操作ではない。

不可分な命令 (atomic instruction)
単一の分割不能な操作で構成される命令であり，これらの操作は中断せずに 1 つの操作としてハードウェアで処理される。

複雑命令セット・コンピュータ (CISC)
メモリ内の位置に対して直接実行される複雑な操作も含めて，複雑な操作を実行する命令を取り扱うコンピュータ。このような操作の例としては，複数バイトのデータ移動や部分文字列検索を実行する命令がある。 CISC コンピュータは通常，RISC(**縮小命令セット・コンピュータ**) コンピュータの反対語である。

複数命令発行
1 つのクロック・サイクルで複数の命令を出すこと。

プログラム・カンウタ (PC)
CPU の中で，次に実行される命令の仮想アドレスを含む部分。現在の大部分の CPU はプログラム・カウンタをレジスタとして実現している。プログラマは命令セットを通じてこのレジスタを確認できる。 Alpha AXP システムでは，プログラム・カウンタはレジスタではないので注意が必要。

プロセッサ・ステータス (PS)
AXP システムでは，**クォドワード**の情報で構成される特権付きプロセッサ・レジスタであり，現在のアクセス・モード，現在の割り込み優先順位レベル (IPL)，スタック・アラインメント，複数の予約フィールドなどを含む。

プロセッサ・ステータス・ロングワード (PSL)
VAX システムで， 1 ワードの特権付きプロセッサ・ステータスと，**プロセッサ・ステータス・ワード**自体で構成される特権付きプロセッサ・レジスタ。特権付きプロセッサ・ステータス情報には，現在の割り込み優先順位レベル (IPL)，前のアクセス・モード，現在のアクセス・モード，割り込みスタック・ビット，トレース・トラップ・ペンディング・ビット，互換モード・ビットなどが含まれる。

プロセッサ・ステータス・ワード (PSW)

VAX システムで**プロセッサ・ステータス・ロングワード**の下位ワード。プロセッサ・ステータス情報には条件コード (キャリ，オーバーフロー， 0，負)，演算トラップ・イネーブル・ビット (整数オーバーフロー， 10 進オーバーフロー，浮動小数点アンダーフロー)，およびトレース・イネーブル・ビットが含まれる。

ページ・サイズ

システムのハードウェアが補助記憶との間でアドレス・マッピング，共有，保護，および移動のための単位として取り扱うバイト数。

ページレット

AXP 環境で， 512 バイトのメモリ・サイズを指す表現。 AXP システムでは，特定の DCL コマンドとユーティリティ・コマンド，システム・サービス，およびシステム・ルーチンは，必要なメモリとクォータをページレット単位で入力として受け付け，出力として提供する。この結果，これらの構成要素の外部インターフェイスは VAX システムの外部インターフェイスと互換性を維持するが， OpenVMS AXP は内部的には CPU メモリのページ・サイズの整数倍でのみ，メモリを管理する。

読み込み/書き込み (リード・ライト) の順序

1 つの CPU のメモリに対する読み込み，書き込みなどの操作を実行エージェント (密接に結合されたシステム内の別の CPU または装置) から確認できるようになる順序。

読み込み/変更/書き込み操作 (リード・モディファイ・ライト操作)

メイン・メモリのデータを 1 つの割り込み不可能な操作として読み込み，変更し，書き込むハードウェア操作。

粒度

1 つの命令によって読み込むか，または書き込むことができるデータ・サイズ，つまり，個別に読み込みまたは書き込みを実行できるデータ・サイズを定義する記憶システムの特性。 VAX システムの粒度はバイト粒度またはマルチバイト粒度であるが，ディスク・システムの粒度は通常， 512 バイト以上の粒度である。

ロード/ストア・アーキテクチャ

データが最初にプロセッサ・レジスタにロードされ，操作された後，最終的にメモリに格納されるようなマシン・アーキテクチャ。ロード/ストア以外のメモリ操作は，このような命令セットには準備されていない。

ロングワード

任意のアドレッシング可能なバイト境界から始まる連続した 4 バイト (32 ビット)。各ビットには右から左に 0 ～ 31 の番号が付けられる。ロングワードのアドレスは下位ビット (ビット 0) を含むバイトのアドレスである。アドレスが 4 で割り切れる場合には，ロングワードは**自然にアラインされる**という。

ワード粒度

メモリ・システムの特性であり，隣接するワードを異なるプロセスまたはプロセッサが同時に個別に書き込むことができる特性。

索引

R

S

T

U

V

X

ア

イ

シ

ス

セ

ソ

チ

テ

ト

ナ

ネ

ハ

ヒ

フ

リ

レ

ロ

ワ

OpenVMS AXP **オペレーティング・システム**
OpenVMS AXP **オペレーティング・システムへの移行：システム移行の手引き**

1994 年 7 月　発行

日本ディジタル イクイップメント株式会社

〒 167 東京都杉並区上荻 1-2-1

電話　(03)5349-7111 (大代表)

AA-PU8KC-TE