ActiveBasic を取り巻く言語市場の今後

(The future of the computer language and the ActiveBasic.)

東海大学開発工学部(School of high-Technology for Human Welfare Tokai University)
山本 大祐(Daisuke Yamamoto)
dais@muc.biglobe.ne.jp

■はじめに

統合開発環境という言葉をご存知でしょうか？漢字ばかりだし、ちょっと難しそうだと思われるかもしれませんが、なんらかのソフトウェアを製作する際に必要になってくる総合的なツールのことを言います。例えば、1 つのソフトウェアを作成する場合、そのソフトウェアが使うアイコン、ビットマップ、メニュー、ダイアログボックス、更には様々な状況に対応するソースコード郡を効率的に管理し、編集していかなければなりません。それを可能にし、プログラマーの開発効率をより高めるために活躍するのが統合開発環境です。

ActiveBasic は Windows アプリケーションを製作するために必要なツール（コードエディタ、RAD ツール[1]、コンパイラ[2]、デバッガ[3]、リファレンスなど）をすべて取り揃える統合開発環境です（図 1、図 2）。本格的な大規模アプリケーションの開発を想定する反面、初心者ユーザーに対してのサポートも積極的に取り組んでいます。

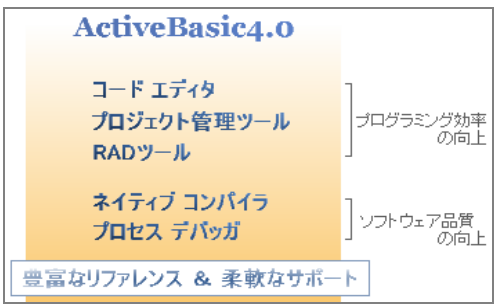


図 1「AB4 のパッケージ構成」

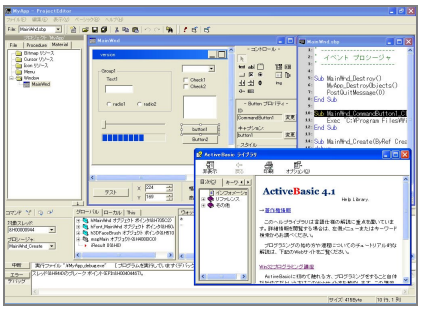


図 2「スクリーンショット」

[1] RAD ツール … マウスなどを使って、視覚的にウィンドウをデザインするツールのこと。
[2] コンパイラ … ソースコードを機械語に変換するための機構。
[3] デバッガ … 生成した実行ファイル（.exe/.dll）を検証するためのテストツール。

■ActiveBasic でなにができるのか

●無料でプログラミングができる

ActiveBasic はフリーウェアという形態をとっているため、誰でも無料で今すぐにプログラミングを始めることができます。すべての機能を無料で使え、サポートも無料で行っています。バージョンアップも頻繁に行われており、常に最新の開発環境を利用できるようになっています。

フリーウェアという形態は今後も継続していく予定です。

●用途が豊富である

ActiveBasic は開発対象となるソフトウェアのジャンルを問いません。ゲームを作るのも、会社で使う業務アプリケーションを作るのも、プログラマー次第です。Windows サーバー上での CGI プログラムの製作、またはサーバー/クライアント アプリケーションなど、コアな分野のソフトウェア開発もお手の物です。ActiveBasic はあらゆる状況、場面に適応できるため、非常に多種多用なソフトウェア開発を行うことができます。

●生成されるソフトウェアの品質が高い

ActiveBasic が提供するコンパイラは、CPU が直接解読できる機械語コード及び PE ヘッダ[4]を生成します。従来型の BASIC 言語はインタプリタという方式を採用しており、処理速度にやや難がありましたが、ActiveBasic は C 言語並みの処理速度を有するソフトウェアを開発できます。

ソフトウェアの動作検証については、プロセスデバッガを用いて詳しく行えます。

●ユーザーのレベルにあったプログラミングスタイル

ユーザーごとのレベルに合わせたプログラミングスタイルを選択できるのも、ActiveBasic の良さの 1 つです。「一昔前に N88BASIC[5]を使っていた経験があるぞ」という方には N88BASIC 互換モードでの開発を、「Win32API[6]を活用してバリバリ Windows アプリ開発を」という方には構造化プログラミング[7]スタイルを、「最新のプログラム技術を活

---

[4] PE ヘッダ … 実行ファイル（.exe/.dll）のヘッダデータのこと。マシンタイプやエントリポイント（実行開始アドレス）など、実行に必要な詳細情報が格納されている。

[5] N88BASIC … 1980 年代に活躍した Basic 言語。NEC の PC-98/PC-88 シリーズに標準搭載されていた。

[6] Win32API … Windows が提供する API(Application Programing Interface)のこと。Win32API で提供されるモジュールを呼び出せば、Windows のほとんどの機能を使える。

[7] 構造化プログラミング … プログラムの制御構造を逐次的に、プログラムの開始部から終了部までの一連の流れを順々に処理するようなプログラミング手法。

用して本格的なソフトウェア開発を」という方にはオブジェクト指向プログラミングを採用していただけます。

初心者なので何がなんだか、わからないという方は、まずは構造化プログラミングをマスターしてプログラムがどのような仕組みで動くのかを体験すると良いかもしれません。

■色々な場面で活躍しています

●Windows アプリケーション全般

一般的な Windows アプリケーションの開発は、ActiveBasic が一番得意とする分野です。Win32API に完全対応していることから、Windows が提供する機能のすべてを使うことができ、更には RAD ツールを利用して効率的にユーザーインターフェイスを作れます。例えば、テキストエディタ、画像エディタ、ファイル管理ツール、サウンド（ムービー）プレーヤー、ランチャーアプリなどを効率的に製作できます。

●2D/3D ゲームアプリケーション

DirectX[8]に完全対応しているので、本格的なゲーム開発を行えます。ABDX（ActiveBasic with DirectX library）という独自のライブラリを利用すれば、DirectX の小難しい話を抜きにして、比較的簡単に 2D/3D プログラミングをマスターできます。

DirectX の扱いに慣れてしまえば、アクションゲーム、シューティングゲーム、RPG、シミュレーションゲームなどの製作も夢物語ではありません。

●ネットワークアプリケーション

OS 上で HTTP サーバーのサービスが稼動していれば、ActiveBasic で作ったプログラムを CGI として利用できます。高速な処理速度を生かした ActiveBasic での Web サービスは、今後の試みとして注目していきたい分野です。

別途、ソケット API もご用意しておりますので、サーバー/クライアントで稼動させるソフトウェアを開発することもできます。

●教育機関での利用

「画面にテキストを表示するだけなのに、数十行のコーディング！？」

Windows プログラミングを始めたばかりの方の多くは、こんな悩みを持つのではないで

[8] DirectX … マイクロソフト社が提供するゲーム及びマルチメディア処理用の API。

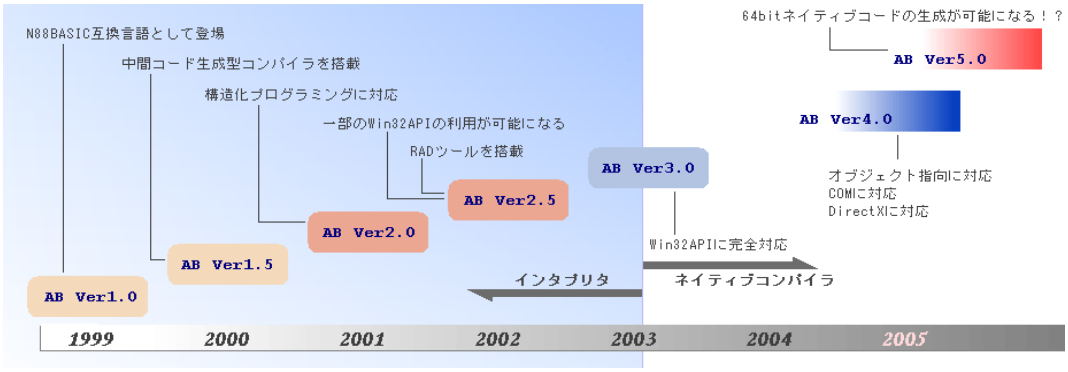
しょうか。まだプログラミングを始めたばかりなのに、いきなり長いプログラムを…となると、誰でも自信がなくなってしまうものです。そこで注目したいのが N88BASIC 互換モードです。昔懐かしい N88BASIC の環境を再現すると共に、学習に最適な環境を提供しています。N88BASIC 互換モードを使えば、プログラミング初心者の方でも、一行プログラムから始めることができます。

●その他

圧縮アルゴリズム、エンコーダ/デコーダ アルゴリズムなど、特殊なプログラム技術を生かしたソフトウェアについても、プログラマーのスキル次第で実現できます。ActiveBasic は DLL ファイルの生成も可能なので、独自の機能を持つライブラリを製作するのも良いでしょう。

■2005 年までのロードマップ

私が ActiveBasic の開発をスタートしたのは今から 7 年前、1999 年のことです。当初は N88BASIC と互換性を持つインタプリタを製作していましたが、Ver2.0 で Win32API に対応し、Ver3.0 でネイティブコンパイラに変化を遂げました。2005 年 5 月にリリースした Ver4.0 ではオブジェクト指向への対応を果たし、現在に至ります。



1999 年～2005 年の ActiveBasic ロードマップ

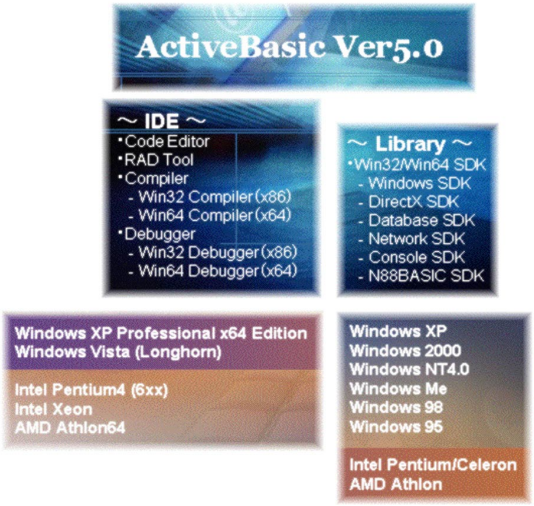
バージョンごとの機能対応表

	Ver1.0	Ver2.0	Ver2.5	Ver3.0	Ver4.0	Ver5.0（開発中）
64 ビット対応	_	_	_	_	_	○
オブジェクト指向	_	_	_	_	○	○
COM コンポーネントの利用	_	_	_	_	○	○
DirectX への対応	_	_	_	_	○	○
構造化プログラミング	_	_	○	○	○	○
Win32API	_	_	○	○	○	○
GUI プログラミング	_	_(_1)	○	○	○	○
N88BASIC 互換	○	○	○	○	○	○
動作速度(_2)	_	_	_	○	○	○

_1 … Ver2.0 では RAD ツールが未搭載なため、手作業に限り GUI プログラムに対応

2 … インタプリタの動作速度を、ネイティブコンパイラの動作速度○で示しています

■AMD64 に対応する ActiveBasic Ver5.0



●64 ビット CPU は何が違う？

Intel Pentium 4(6xx)、Athlon64 などが 64 ビット CPU にあたるプロセッサです。

2003 年 9 月、米 AMD が AMD64 という規格のプロセッサを発表したのがことのはじまりです。AMD64 とは、x86 の命令と互換性を持ちながらレジスタを 64 ビットに拡張し、64 ビット処理に対応する新しい命令を追加したプロセッサのことをいいます。

実はこれ以前に Intel が IA-64（CPU 名：Itanium）というプロセッサをリリースしましたが、x86 との互換性が低いということで、エンドユーザー向けの市場は AMD64 に先を越された形になっています。

ActiveBasic Ver5.0 のコンパイラが生成するのは、AMD64 の規格に沿ったネイティブコードであり、Itanium への対応は今回の段階では行いません。

64 ビットになると何が良いの？という問いに関してですが、これは何といっても扱えるメモリ空間が 16EB（エクサバイト[9]）という無限に近い大きさになることでしょう。いままで、32 ビット CPU が抱えていた 4GB 以上のメモリが扱えないという問題が一気に解消されます。詳しく言えば、AMD64 プロセッサが扱えるメモリ空間には、ハード・ソフト面から生じる様々な制約があり、事実上は数 TB（テラバイト[10]）になりますが、それでも 4GB という環境からすると、十二分なメモリ空間であることは間違いありません。

---

[9] エクサバイト … 1EB = 1024PB（ペタバイト） = 1024*1024TB（テラバイト）
[10] テラバイト … 1TB = 1024GB

●AMD64 はレジスタ数も増えた

一つのレジスタで扱えるビット数が 32 ビットから 64 ビットになると同時に、AMD64 ではレジスタの数そのものが増えています（表 1）。RAX、RCX などという単語は、レジスタの名前です。CPU に直接的に内蔵されている変数のようなものだと捉えてもらえればいと思います。

x86	AMD64
EAX	RAX
ECX	RCX
EDX	RDX
EBX	RBX
ESP	RSP
EBP	RBP
ESI	RSI
EDI	RDI
	R8
	R9
	R10
	R11
	R12
	R13
	R14
	R15
表現可能な数値の範囲は 0 ～ 4294967295	表現可能な数値の範囲は 0 ～18446744073709551615

表 1「86 と AMD64 の汎用レジスタの違い」

●SSE2 を標準搭載

SSE とは、複数の浮動小数演算を同時に行う際や、マルチメディア処理を頻繁に行う際に威力を発揮する拡張命令セットのことです。MMX→SSE→SSE2 という具合に進化を遂げており、マルチメディア演算に強い比較的新しい機構であるといえます。SSE2 は、128 ビットのマルチメディア演算用レジスタを 16 本搭載しています。

x86 ではスタックベースになっている 8 本の FPU レジスタを、メモリを介して利用しなければ実数演算を行えませんでした。

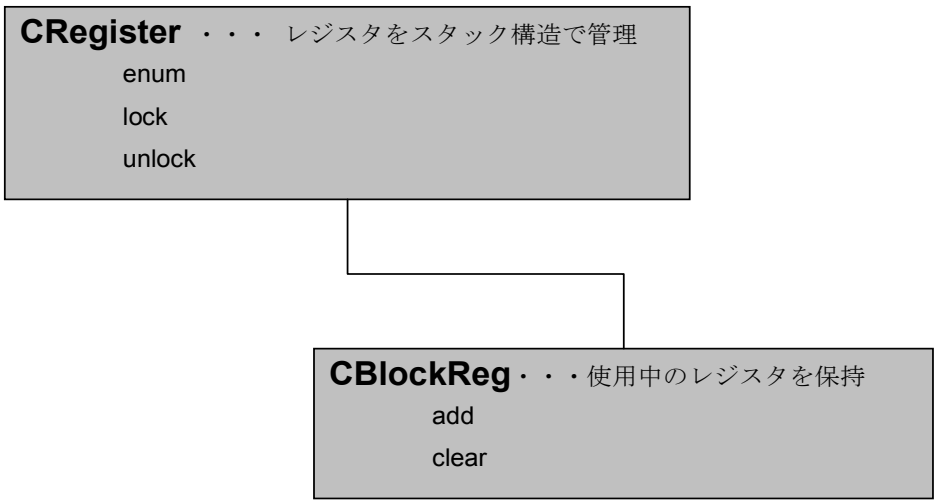
これらのことを踏まえると、SSE2 への移行は高速化がおおいに期待できると言えます。

●最適化されたレジスタ割付

先ほどご紹介したとおり、AMD64 は今までの x86 と比べ、一つのレジスタが持ちあわせるビット数及びレジスタそのものの数が増加しています。レジスタが増えたとなると、それだけメモリへのアクセスを最小限に留めることができるのではないかという期待が出てきますが、ここからがコンパイラの良し悪しに関わってくるところです。

Ver4.0 までの 32 ビットコンパイラは、コンパイラ自身の処理の簡略化のため、演算コード生成部分を完全に逆ポーランド記法に則った形で実現していました。そのため、push/pop を繰り返し、無駄にメモリアクセスを必要としてしまうような少々冗長的なネイティブコードになっていました。

これでは、64 ビット命令を呼び出すことに成功したとしても、処理速度の向上はあまり期待できません。これを回避するためには、メモリへのアクセスを最小限で抑えられるよう、最適にレジスタ割付を行う必要があります。Ver5.0 を構成するコンパイラは内部で独自のクラスモジュール CRegister、CBlockReg を持つことにより、レジスタ割付の状態、演算過程で利用中のレジスタをブロッキングする機構を取り入れています。



●スタックフレームの有効活用

実は ActiveBasic Ver5.0 が生成するネイティブコードは、push/pop 命令は一度も出てきません。なぜかというと、あるモジュール内で使用されるスタック領域は、モジュールの実行開始部分で一気に確保してしまい、あとは SP（スタックポインタ）からのオフセット値を指定することでスタックフレームへのアクセスを可能にしているからです。これは従来のネイティブコードの方式と大きくことなる点であり、SP とそこからのオフセット値だけで、レジスタ値の退避・復元、ローカル変数の利用、関数パラメータの引渡しが可能になっています。

補足：

従来の方式では、push/pop がどのタイミングで使われるかを把握し辛いため、SP からのオフセット値を指定することが、事実上、困難となっていました。そこで登場したのが BP というスタック領域内に設けるベースポインタであり、この BP はモジュール内では特殊なケースを除き、不変でなければなりませんでした（AB x64 コンパイラでは、BP はベースポインタとしての役割を果たしていないので、通常の汎用レジスタとして利用可能になっています)。

●32 ビットから 64 ビットへの移行作業はラク？

AMD64 は、そのアーキテクチャ自体は x86 の延長線上にあるものなので、構造上の大きな変化は伴いません。単純に、レジスタが 64 ビット化しただけという捉え方でも間違いではないのです。

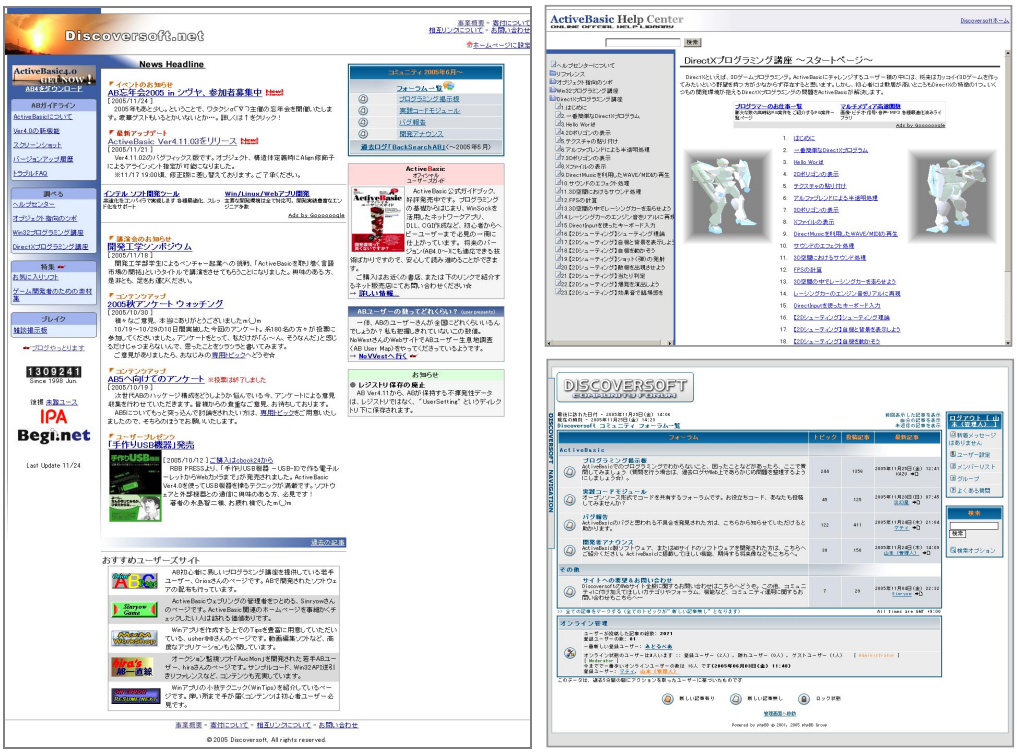
AB プログラマーが既存のソースコードの 64 ビット化を行うとき、一番のネックになるのが、ポインタ表現がすべて 64 ビットになっていることです。ポインタ型のみでの演算が繰り広げられている部分はそれらデータ部が自動的に 64 ビット化されるのでいいのですが、Long 型（32 ビット整数型）を仲介させて演算を行っている部分は注意を払わなければなりません。

ActiveBasic は型のキャストが可能になっているため、この問題を深刻に考える必要があります。32 ビットのソースコードを 64 ビットへ移行する際、キャストを厳密にチェックし、データが間欠してしまう恐れのある部分（例えば、ポインタ型を Long 型へ強制変換している部分）はたとえキャストを行っていたとしても、警告を発する機構を取り入れる必要がありそうです。

■Web コンテンツ、コミュニティサイトへの課題

ActiveBasic のようなソフトウェアを開発したとしても、それを利用してくれるユーザーがどれだけいるかで、その利用価値が問われます。ユーザーは、使いやすいのか、自分が思い描くソフトウェアを作れるのかといった観点から、ActiveBasic を利用するかどうかを決定します。ActiveBasic 自体を良いモノに仕上げるのは当然必要なことですが、良いモノだということを演出する営業的な要素を今後、より一層強めていかなければなりません。

演出に関して、てっとりばやく動けるところが Web サイトコンテンツです。今までは単純に私がコンテンツを作り上げ、ユーザーに閲覧してもらうという形をとってきましたが、今後はコンテンツ執筆者・編集スタッフの確保を行い、多角的な話題で注目度を高めていく必要があると感じています。Web サイトが生み出す小額の利益だけで、どれだけの協力者を集い、それぞれが仕事を進められるかが今後の焦点になってくると思います。



左「公式サイト トップページ」、右上「ヘルプセンター」、右下「コミュニティ」

■ActiveBasic Ver6.0 は.NET 対応なるか！？

ActiveBasic はインタプリタ→ネイティブコンパイラと変化を遂げてきましたが、Windows Vista、Visual Studio.NET の普及で急速に浸透しそうな.NET の存在を視野に入れていく必要があります。

.NET を視野にいれるとなると、機械語コードの生成部に関する作業が簡略化され、言語構造はどうあるべきかに焦点を絞って開発が可能になります。逆を取れば、ネイティブコンパイラとしての存在価値をとことん追求するのか、プログラミング言語のあるべき姿を.NET 技術を活用して追求するのかを、現時点で選択していく必要があるのだと思います。

「これからが本当の未踏！？」

2006 年春は、ActiveBasic Ver5.0 のリリースを行うと同時に、更にその先の方向性を見極める転換期でもあります。私が ActiveBasic を開発してきた力をどのような方向へ生かしていくのか、暖かく見守っていただければ幸いです。