制御用ＢＡＳＩＣコンパイラ

平野　文信
㈱アドテックシステムサイエンス

現在マイクロコンピュータ用として供給されているコンパイラの多くは、当初大型コンピュータ／ミニコンピュータ用として開発された汎用言語が多く、その言語の走行環境上で走行するアプリケーションを目的としている。
しかし、マイクロコンピュータでは、コンパイラの走行環境とそのオブジェクトプログラムの走行環境が異なることが非常に多く、従来のコンパイラでは、それが原因して、不向きな事が多かった。　しかも、制御分野に利用できるコンパイラは、非常に少なく、難解な記述のものであったりした。そこで、マイクロコンピュータのユーザーに最もポピュラーなＢＡＳＩＣで制御プログラムを記述できることを目標として、作成されたのが、本コンパイラ　ＺＩＬ　である。

ＺＩＬは

(1)　制御プログラム記述言語としての特徴を持たせる。

(2)　ＲＯＭ化が容易であること。

(3)　プログラムサイズが、コンパクトにまとまっていること。

(4)　割り込み処理記述を容易にできる。

(5)　出力プログラムのブラックボックス化を避けるため出力形式はアセンブラ、ソースコードとする。

以上を骨格にして、設計された。

(1)　制御プログラム記述言語としての特徴

◦メモリ操作
ＰＥＥＫ関数　（指定アドレスの８bitの読み出し）
ＰＯＫＥ文　　（指定アドレスに８bitの書き込み）
ＤＰＥＥＫ関数（指定アドレスの16bitの読み出し）
ＤＰＯＫＥ文　（指定アドレスに16bitの書き込み）
ＳＷＡＰ文　　（指定アドレスと指定アドレス＋１番地の内容の交換）

従来のＰＥＥＫ関数、ＰＯＫＥ文に加えて、その動作を16ビットで行うＤＰＥＥＫ関数、ＤＰＯＫＥ文と、指定アドレスとその次のアドレスの内容を交換するＳＷＡＰ文が、追加された。

・I/O操作

INP関数	(指定I/Oポートからの8bitの読み出し)
OUT文	(指定I/Oポートに8bitの出力)
WAIT文	(指定I/Oポートが任意のデータと一致するまで読み出す)
I8255文	(汎用I/Oポート8255の初期設定)

・BIT操作

BIT関数	(指定アドレスの指定ビットのテスト)
SET文	(指定アドレスの指定ビットのセット)
RES文	(指定アドレスの指定ビットのリセット)

・時間遅延

DELAY	(オペランドをミリ秒として時間遅延を行う)

(2) ROM化が容易

制御プログラムの完成時には多くの場合、ROM化という作業が要求される。ZILの場合、出力形式が、アセンブラ・ソースコードであるため、コンパイル後のプログラムはユーザーにとって、アセンブラで記述した感覚で処理をして行けば良い。

i) インタプリタ、ED、Word Master、PMATE等で、ソースプログラムを作成

ii) ZILでコンパイル

iii) MACRO-80でアセンブル

iv) LINK-80で実行形式(COMファイル・HEXファイル)を作成

v) ROMライトユーティリティにてROMの書き込み

iii)以後の処理は、通常アセンブラで作成する手順と同じである。
次にこの一連の処理を図式化したものを示す。

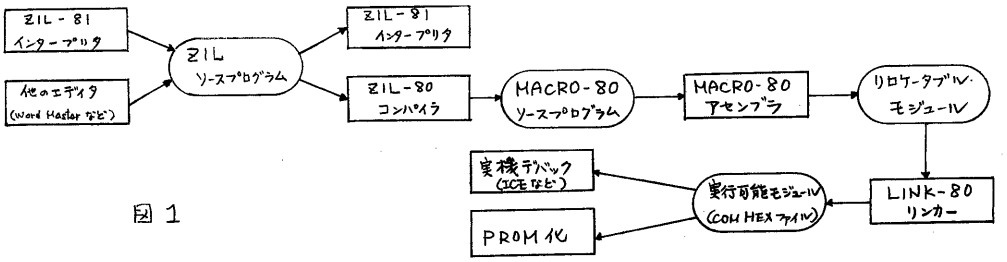


図1

(3) プログラムサイズをコンパクトにまとめる。

プログラムサイズをコンパクトにまとめるため次の様な工夫がされている。

1. 定数演算のたたみ込み
   A＝B＋3＊4－5＋I という数式は
   A＝B＋7＋I として扱れる。

2. 加算、減算における最適化
   加算では、定数＋変数 あるいは 変数＋定数の時
   減算では、変数－定数 において 定数の値が、－3～＋3の範囲に入る時は、加算を行わず、INC命令あるいは、DEC命令により処理される。

3. 乗算における最適化
   定数＊変数、変数＊定数が2のn乗形式で表現できる場合
   次に示すオブジェクトを生成

定数		
2～32	ADD	HL，HLを1～5回加力
64～16384	LD	B，n （nは6～14）

4. 連続する単純代入文に対する最適化
   ここで言う単純代入文とは、左辺が整変数で右辺が整数である場合である。
   この単純代入文が、一つの文内で連続している時左辺の数値によって代入の順序を変えることにより、オブジェクトサイズの縮少と実行スピードを向上する。

5. ランタイムライブラリを、きめ細かく作成することにより、必要最小限のライブラリのみを呼び出す様に設計されている。

(4) 割り込み処理記述を容易にできる。
Z80は、割り込みモードを3種持ち、次に示す通りとなっている。

MODE 0： 8080プロセッサと同等の動作モード
MODE 1： 割り込みが要求されると38番地を呼び出す。
MODE 2： 割り込みベクトルをハードウェアから受け取り、間接的に呼び出す。

以上は、マスク可能な割り込みであるが、マスク不可能な割り込みも用意されている。これをNMIと呼んでいる。

ＮＭｉ　　：　６６番地を呼び出す。

これらを　ＺＩＬでは以下の様に記述する。

```
ＯＮ　ＩＲＱ０　ＧＯＴＯ　行番号１、行番号２……行番号８．
ＯＮ　ＩＲＱ１　ＧＯＴＯ　行番号
ＯＮ　ＩＲＱ２　ＧＯＴＯ　　〃
ＯＮ　ＮＭｉ　　ＧＯＴＯ　行番号
```

これらは、分枝先の宣言文として扱れるが、モード２の宣言文のみ具体的な分枝先は、示されていない。これは、モード２の割り込み分枝はノン８種類あるため、一行に記述することは、非現実的なものとなる。したがって、モード２の分枝先のために特別な宣言文として、ＩＴＡＢＬＥを用意している。これは、ＢＡＳＩＣのＤＡＴＡ文に類似するもので、個々の分枝先を定義する。又、ＩＴＡＢＬＥによって生成されるテーブルは、２５６バイトバウンダリーにセットされている必要があるが、それらは、コンパイラ側で自動的にセットする。

モード０のベクトルは、特に指定しない限り０番地に展開される。又、８２５９（割り込みコントローラ）を使用し、呼び出しアドレスを指定する場合のために、ＩＭＯＯＲＧという宣言文が用意されている。

いずれの割り込みモードの時でも、保護されるレジスタは、ＺＩＬの出力の実行形式が使用する、A.B.C.D.E.H.L.F.IX、IYをセーブし、裏レジスタに対しては、特に何の操作も行なわない。

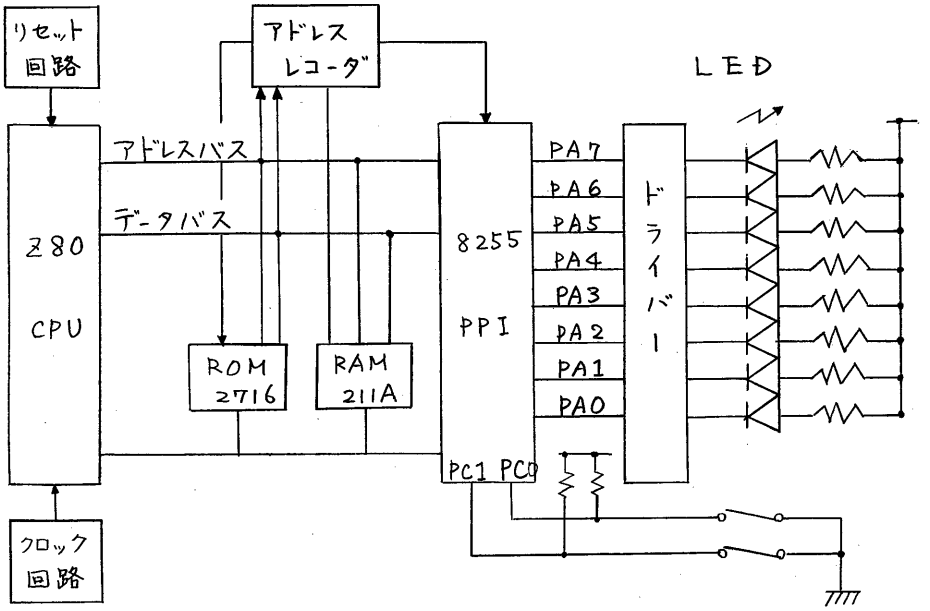
(5)　出力プログラムのブラックボックス化を避ける。

従来のＢＡＳＩＣコンパイラの多くは、出力が実行形式に近く、どの様に展開されているかの情報を得ることが困難であったのに対し、ＺＩＬは、出力が、アセンブラ：ソース・コードであるため、ユーザは、エミュレーションデバック時の実アドレスを見出すことが容易である。
さらに、ＺＩＬの出力したオブジェクトに対し、コンパイル後、手をほどこすことも可能である。又、現バージョンでは、ランタイムライブラリのソースも公開しているため、ユーザの手を施したい所に対しては、より便利になっている。

応用例　　簡単な入出力プログラム

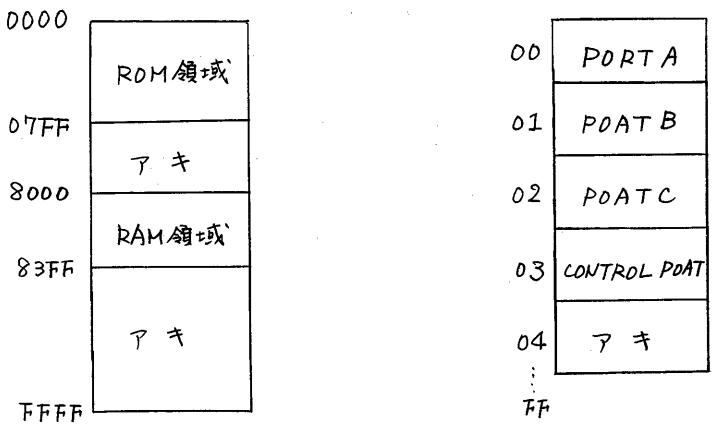
ハードウェアブロック図
本プログラムをテストするため、以下のブロック図に示すハードウェアを必要とした。

図 2.



メモリマップ、I/Oマップ

図 3.



CP/MのED又は、その他のエディタにより、図4の様なプログラムが作成された。このソースリストに通常のBASICには見られないコマンド I8255 と STACK の二つがある。

ソースリスト

図4.

```
10  REM SAMPLE FOR ASC8Z & LED
11  I8255 I,'0003,M=0,A=0,C0=1
12  STACK='8400
13  LET A=(INP(2)AND3)+1
14  ON A GOSUB 16,17,18,19
15  GOTO 13
16  OUT 0,'0000:RETURN
17  OUT 0,'0055:RETURN
18  OUT 0,'00AA:RETURN
19  OUT 0,'00FF:RETURN
20  END
```

行番号11は8255（パラレルI/Oポート）の初期設定である。I8255の書式は次の様になっている。

書式　：I8255, arg1, arg2, arg3, arg4, arg5, arg6, arg7,

arg1：M：メモリマップドI/Oの時。
arg2：8255がアサインされているアドレス，式でよい。
arg3：グループAのモードセット
　　M=nの様に記述する。
　　nは0，1，2，
　　グループBは常にモード0である。
arg4～arg5
　　Portの入出力指定であり，左辺には，A、B、C0、C1が許され“=”のあとに I，0を指定する。
　　C0はPort Cの低位　4Bit
　　C1はPort Cの高位　4Bit　　を意味する。

行番号12はメモリマップから，メモリの最終アドレス＋1の8400番地にスタックポインタをセットする。

書式　：STACK=exc

スタックポイントを定義する命令で、どこでも使用できるが、最初に現れたSTACK文は、実行文を実行する前にSP(スタックポインタを定義します。GOTO文により最初に現れたSTACK文が、最初に実行されない場合も前述の約束が有効である。

フローチャート

図5

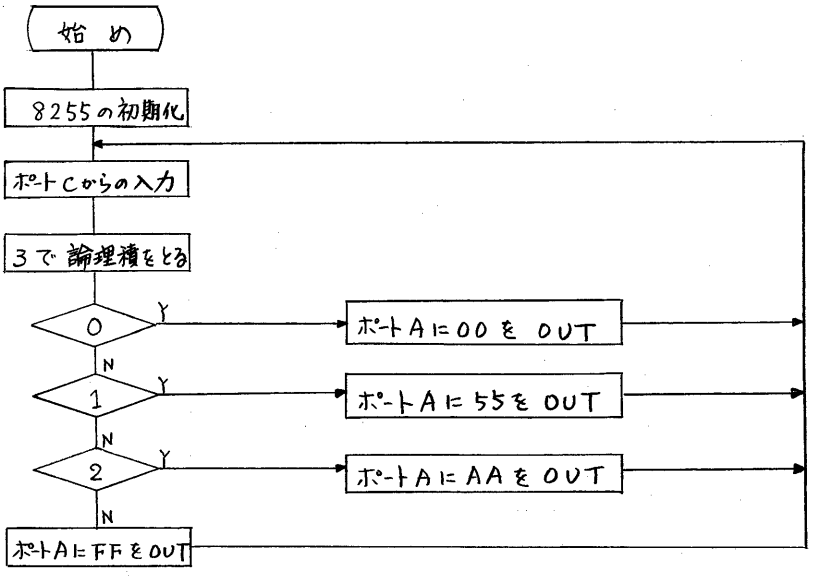


図5のフローチャートは 図4のソースリストのものである。さらに図3におけるメモリマップを満足する様に、ROM化に対応したコードエリアを明確に区分して生成することも、ZILの大きな特徴の一つである。

ソースプログラムのコンパイルからオブジェクトの生成まで

コンパイルオペレーション

```
A > ZIL80  B : ASC8Z1.BAS  B : AO
ZIL80  COMPILER
PHASE - 1
PHASE - 2
PHASE - 3
NO  FATAL  ERROR
COMPILE  COMPLETED
A > ZIL80  n : < filename.typ >  m : ABO
```

コンパイル時におけるnはソースファイルの存在するドライブ名、mはオブジェクトを出力するドライブ名、スイッチABOはそれぞれ次の事を表す。

A... アセンブリソース出力
B... ソースプログラムをコンソールに出力
O... 最適化を施す

尚、コンパイラはPHASE-1.2.3を含む4要素で構成され、次に示す処理を行う。

PHASE-1...字句解析
PHASE-2...構文解析、意味解析
PHASE-3...オブジェクト生成

実行後、コンパイラのオブジェクトとして、MACRO-80(Microsoft社)と互換性を持つソースファイルを出力する。

MACRO-80によるアセンブル、オブジェクト生成

```
A>M80  =ASC821/L
NO FATAL ERROR
A>B:L80/P:0000,/D:8000,B:ASC821,RUNLIB22/S,:ASC821/N/E
LINK-80 3.4 01-DEC-8 COPYRIGHT 1979,80 (c) MICROSOFT
DATA     8000   8010   < 16 >
PROGRAM  0000   0080   < 176 >
```

L80によりオブジェクト・プログラムとRUNLIB22とのリンクを行う。ここでは、オブジェクト効率を高めるため、必要最小限のルーチンのみを結合する。プログラムエリア(ROM化する部分)が176バイト、データエリア(RAMエリア)が16バイトという結果が得られた。