スタック向きプログラミング言語 STOIC

木村　等
中村邦彦

I　まえがき

電子回路の集積化の技術の発達に伴ない，マイクロプロセッサーの性能は急速に向上し価格が低下していくなかで，それに伴なうべきソフトウェアの問題が大きくなってきている。即ちソフトウェアの開発と保守に時間と費用がかかりすぎるのである。このことは何も小さなシステムに限ったことではなく，大型システムに於いても言われていることではあるが大型機の場合は最新の開発ツールを自由に利用できるのに対し，ミニ・マイクロコンピューター（以下マイコンと略す）では，それが使用不可能であるかまたは準備されていないことが多いため情況は少し異なっている。さらにターゲットシステムは制御用として使われる場合が多く，アセンブラーを使わざるを得ないという情況もある。従ってマイコンにはそれに見合うプログラム開発ツールが必要である。

このような考えに基づいて開発された言語にFORTHという言語があり，最近注目されてきている。FORTHは1969年頃アメリカのチャールス.H.ムーアによって考案され，1973年以後はFORTH社がこれを供給している。現在ではRCAのマイコンCOSMAC 1802の標準言語として採用されているのを始めとしてほとんどのマイコンの上で走るようになっている。（第1，第2表）[1]。

FORTHの作成者や販売者の主張するところによれば(1)，

(1) FORTHに関しては特に断らない限り文献〔1〕，〔2〕，〔3〕及びASR社のパンフレットによる。

第1表　FORTH の使える機種

—ASR社の資料による—

機種名	会社名
Series-1	IBM
PDP-11	DEC
NOVA/Eclipse	DG
21MX	HP
8/32	Interdata
Level 6	Honeywel
9900	TI
8086	Intel
8080/8085	Intel
Z80	Zilog
6800	Motrola
6809	Motrola
1802	RCA

第2表　FORTH 系言語

プログラム名	適応機種	開発者
fig-FORTH	8080	FORTH INTEREST GROUP
CPM Multi-FORTH	8080(CP/M)	CREATIVE SOLUTIONS
STOIC	8080	MIT & HARVARD UNIV. BIOMEDICAL ENGINEERING CRENTER
MMS-FORTH	Z-80(TRS-80)	MILLER MICROCOMPUTING
CONVERS	Z-80	THE DIGITAL GROUP, INC.
Z-80 FORTH	Z-80(TRS-80)	
6502 FORTH	6502 (APPLE II, PET 2001)	
6800 FORTH	6800 (SWTPC)	
8080 FORTH	8080 (IMSAI))	PROGRAMMA CONSULTANTS
APPLE-FORTH	6502(APPLE II)	CAP'S SOFTWARE
LAB-FORTH	LSI-11, PDP-11 (RT-11)	LABORATORY SOFTWARE SYSTEMS, INC.
SWING	PDP-11, Series-1	CHILD Incorporated
FORTH FOR PDP-11 (RT-11, STAND ALONE)		DECUS
SAO-FORTH	NOVA	Smithsonian Institution Astrophysical Observatory

(1) プログラムの開発時間をアセンブラーの 1/10, PL/M の 1/6に短縮することができる。

(2) メモリー容量はアセンブラーより小さく PL/M の 1/4 ですむ。

(3) 自己増殖型言語であるので，ユーザー自身による言語仕様の拡張が容易である。

(4) 構造化プログラミングが可能である。

ということになっている。これが事実であれば素晴しいことであるので，是非使用してみたいと考えていたところ，今度FORTHと同種の言語STOIC (Stack Oriented Interactive Compiler)[4] を入手できたのでその性能評価を行なった。評価は主として ①プログラムの実行時間，②メモリー使用量の 2 点に関し，手元にあるZ80アセンブラー，PLZ, UCSD Pascalと比較した。

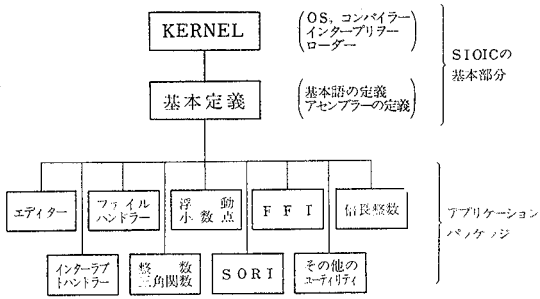
本論では，II節においてSTOIC の簡単な解説と，今回の 移植に際して行なった機能の拡張及び改善について述べ，III節に評価した結果を述べる。

II STOIC の概要

2.1 STOICの特徴

STOIC (Stack Oriented Interactive Compiler) は，1977年 MIT と ハーバード大の Biomedical Engineering Center により開発された8080用の言語で，基本的な構造は FORTHと全く同じであるが，個々には少し異なっており，改善されている面もある。[5]

全体は，コンパイラー，インタープリター，アセンブラー，デバッガー，ローダーとカセットMT 又はフロッピー・ディスク用のオペレーテングシステムからなる 1 つの独立したシステムを構成している。今回入手したものは第 1 図のようになっており，豊富なアプリケーション・パッケージが付いているのも特徴である。このうち KERNEL だけが普通の アセンブラーで 記述されているが，他はすべてSTOIC 自身の言葉で書かれている。



第1図　　STOIC の 構 成

KERNELは，STOIC語で書かれた基本語の定義を受け入れるために必要な最小限の部分で，コンパイラー，インタープリター，ローダー，OS が含まれている。これに“基本定義”を読み込ませることによりSTOICの基本システムが出来上る。あとは必要に応じて各アプリケーション・パッケージを追加していくことができる。

STOICの主な特徴は次のようである。

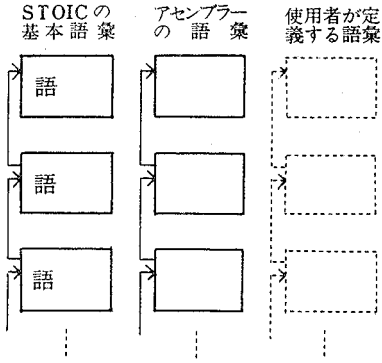
(1) 会話型でプログラムができる。

これは，BASICと同様にプログラムを一行ずつ直接入力することができ，即実行できるということである。

(2) 内部にディクショナリーを持っており，定義された定数，変数，手続き等はすべて語（ワード）としてこれに登録される。

ディクショナリーは語のオーダード・リストとして表現される。この語のうち，STOICの基本システムによってあらかじめ定義されているものを基本語といい，それ以外のものは使用者が自由に追加していくことが可能である。一担定義してしまえば基本語と同様に使用することができる。普通のプログラム言語の場合は，使用者の作成したライブラリーを使おうとすれば，ソースプログラム中に埋め込んで再度コンパイルするか，リンカーにより結合するかしなければならないところが違っている。こうして使用者は自分に固有の語彙をもったシステムを仕立てることができる。またディクショナリー全体は，いくつか

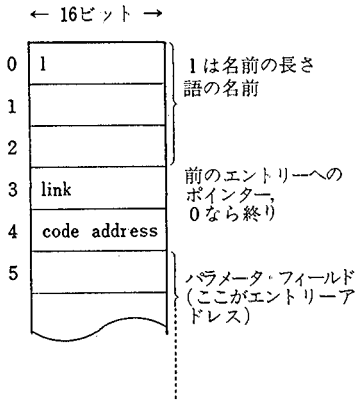
のブランチ（正確にはこれを語彙と呼んでいる）に分けて使い分けることもできる（第2図）



第2図　ディクショナリーの構造

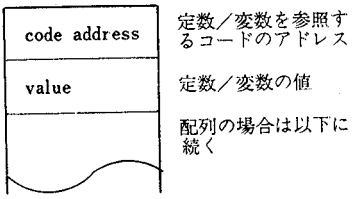
語を識別するための名前は，区切り記号の空白とタブ記号を除く他のすべての英数字と記号が使える（例えばカンマ“，”やセミコロン“；”も語であるし，また語の一部としても使用できる）。これにより新しい演算記号を定義することができる。

語（ワード）の構造は第3図のようになっており，コードアドレス以下の部

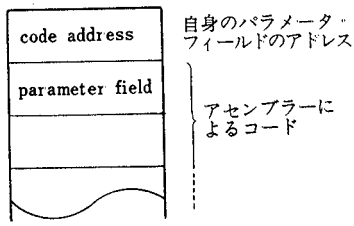


第3図　語のエントリーフォマット

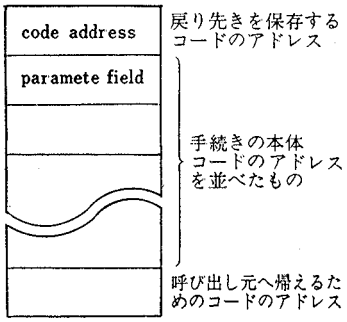
a. 定数・変数を表わす語



b. アセンブラーによるコードを表わす語



c. STOICのコードを表わす語



第4図　語のコード部分のフォーマット

分が，その語の表わす内容により異なるが，主なものは第4図のようになっている。語がマシンコードを表わす場合コードアドレスの部分に自身のコード部分のアドレスを入れておくのは無駄のようであるが，他のものと統一的に扱うためにそうしてある。語が定数又は変数を表わす場合は，パラメータ・フィールドにその値が入り，コードアドレスには，定数あるいは変数参照を行なうコードのアドレスが入る。定数の場合は定数の値がスタックに積まれ，変数の場合は，そのパラメータフィールドのアドレスがスタックに積まれる。

語がSTOIC語のコードを表わす場合は，コード・アドレスには通常の手続き呼び出しにあたる処理を行なうコードのアドレスが入り，以下本体の処理を行うコード・アドレスが並ぶ。最後は呼出し元へ復帰するためのコードへのアドレスがくる。

(3) オブジェクト・コードは indirect threaded code になっている。

これは先の語の中味が（その語がSTOIC語で書かれたコードの場合）別のコードのアドレスを並べただけのものであったが，まさにそのことである。これは一般に機械語のCALL命令よりは短かくなるので，開発者側はプログラムがアセンブラー使用時より小さくなると称しているようである。

(4) プログラム言語自身がスタック向きに設計されており，文は逆ポーランド記法により記述される。

言語プロセッサーがスタック向きに作られている例は多いが，言語自身を逆ポーランド式に書かせるのは一部のポケットカリキュレーターを除いて初めてであろう。この方式によれば，オペレーションはプログラム中にそれが現われた時点で実行できるため，先のオブジェクトコードの生成法と相まってコンパイラーが非常に簡単になる。

スタックは全部で4つ用意されている。

・パラメータ・スタック

　演算用のスタック

・リターン・スタック

　手続きを呼び出すとき，その戻り先きを保存するためのスタック

・ループ・スタック

　プログラム中のループを制御するためのスタック

・語彙スタック（Vocabulary Stack）

　ある時点での使用中の Vocabulary Branch を指定するためのスタック

FORTHにおいてはループスタックと語彙スタックが兼用されているため，別の手続きを呼び出してしまうとループの制御変数が直接参照できないという欠点があったが，STOICでは改善されている。ただしスタックの数をふやせ

ばレジスタ数の少ない機種ではインタープリターのスピードが落ちることになろう。

(5) プログラム中に，アセンブラーによるコードを容易に埋め込むことができる。

これは，基本システム自体がアセンブラーを内蔵していること，引数はスタックを介して受け渡しすること，オブジェクト・コードの構造がうまくできていることにより実現されている。これによりインタプリターのオーバーヘッドを制御できる。ただしアセンブラー自体もスタック向けに設計されているので，後述のようにZilog社のZ-80のニーモニックはそのままでは使えないという欠点がある。

(6) 構造化プログラミングが可能である。

このために用意されている構文としては次のようなものがある。以下大文字で書いたものはキーワード，小文字で書いたものは式又は別の文を表わす。

c IF s FI[2]

IF c THEN s に当る。逆ポーランド式に条件 c が IF の左にくる。

c IF s_1 ELSE s_2 FI

IF c THEN s_1 ELSE s_2 に当る。

n (s)

s を n 回繰返す。n はそのときのスタックの一番上の値（以下 tos という）である。

BEGIN s END

END の直前の tos の値が真になるまで s を繰返す。REPEAT s UNTIL c 型のループ

BEGIN c IF s REPEAT

WHILE c DO s 型のループ

---

(2) FORTH及びSTOICのものは，IF～THEN又はIF～ELSE～THENであるが，この場合THENはIF文の終りを示すだけであるから後述するZ80 STOICではFIとした。

n_2 n_1 DO s LOOP

FOR i=n_1 TO n_2-1 DOに当る。n_1 は tos, n_2 は tos の下の値。制御変数は常にその時点で起動している最も内側のループから順に I, J, K により参照する。

以上非常に簡単に述べたが，詳しくは巻末の文献を参照していただきたい。

2.2 Z80STOIC

STOIC はその中に OS をも含んだ独立のシステムであるので，そのままでは手元の計算機（Zilog 社の MCZ 1/05）では使いにくい。また STOIC は8080 システムのために作成されているので，Z-80 の命令セットを有効に使えない。これらを改善するために Z80STOIC を作成した。以下主な変更について述べる。

(1) STOIC を MCZのOS RIO の下で動くように，KERNELにおける入出力の部分を RIO とのインターフェースに置き換えた。

(2) STOIC 向きの Z80 アセンブラーを作成し，もとのアセンブラーと置き換えた。アセンブラーのニーモニックはできるだけ Zilog 社のものに近づけようとしたが，Zilog 社の形式は STOIC には不向きなため，やむをえず新しいニーモニックを追加したものもある。Z80 STOIC の全アセンブラー命令の形式は appendix 1 にまとめてある。STOIC においては名札というものが存在しないため，後方へジャンプする場合と前方へジャンプする場合とでは処理の仕方が異なる。後方へジャンプする場合は，飛先のアドレスをあらかじめスタックに積んでおいてそこへ飛ぶようにすればよいが，前方へジャンプする場合は一担飛び先未定のジャンプ命令を出力しておき，あとで飛び先のマークが現われた時点でこれを書きなおすということをしている。この前方へのジャンプは IF，又はIFcc，（ここで cc は条件名）とし，対応する飛び先は FI，とした。(3)

(3) これに伴ない STOIC の基本定義分はすべて Z80 STOIC のアセンブラー

---

(3) STOICではIFcc，…… THEN，である。またIFcc，はそのまま条件ccのときジャンプする命令に対応していたが，Z80STOICでは条件ccのとき……が実行されるように変えた。

に書き替えた。その際，減算，乗除算の部分はZ80に固有の命令を用いて書き替えるなどして改善した部分もある。

以上の変更はSTOICをZ80向けにするための最小限の変更に近く，性能を大巾に改善するという程のものではないことをお断りしておく。

Ⅲ　STOICの性能評価

3.1　評価方法

STOICの性能のうち，特にプログラムの実行速度とプログラムの大きさを調べるために，2つの問題を実際にプログラムし，それを他の言語を使ったものと比較した。プログラムの実行のために使用した計算機は，Zilog社のマイクロコンピューターMCZ 1/05である。これはCPUにZ-80を用いており，クロックは2.5MHzである。

比較のために用いた言語及び言語プロセッサーは，Z-80 アセンブラー，PLZSYS, PLZCG, PASCALである。各言語についてのデータはAppendix 2にまとめた。

評価のための問題は，

(1)　8-Queenの問題の解を数える，

(2)　小さい方から1000個の素数をみつける，

の2つである。8-Queenの問題は，再帰的なアルゴリズムを用いたため，手続きの再帰的呼出しがあるのが特徴であり，素数の問題は繰返しが多く，その中で多数の除算が現われるのが特徴である。上の問題を各言語でプログラムしたものはすべて appendix 3 に掲載した。(4)

テストの結果は第3表に示す。なおこのテストに用いたSTOICは，先に述べたZ80STOICであるが，もとのSTOICに比べて，乗除算ルーチンがZ80用のものに替えてあること，インタープリターの中で普通なら一命令実行毎にスタックのあふれ等のエラーチェックを行なうところを，スピード測定の際だけ

(4)　各言語を用いてプログラムする際には，原則として同じアルゴリズムを用いたが言語の特徴を生かすために少し変更した部分もある。同じアルゴリズムであっても書き方の違いによって結果が異なる場合は，成績の良い方を採用した。

第3表 ベンチマークテストの結果

1. 8-Queen 問題

言語	実行時間 (秒)	プログラムサイズ (バイト)
アセンブラー	0.9	167
STOIC	40.4	548(412)*
PLZSYS	27.6	238
PLZCG	2.2	442
PASCAL	35.8	344

* STOICの語の名前とリンクの部分を除いた値

2. 素数をみつける問題

言語	実行時間 (秒)	プログラムサイズ (バイト)
アセンブラー	15.2	2115(115)*
STOIC1	91.8	2268(220)**
STOIC2	63.1	2318(254)**
PLZSYS	47.3	2143(143)*
PLZCG	17.0	2267(267)*
PASCAL	54.2	2174(174)**

* 素数1000個分の配列を除いた値
** 素数1000個分の配列とSTOICの語の名前とリンクの部分を除いた値

はこれを取除いたものを用いた事をお断りしておく。なお第3表中の実行時間とは，最初の出力が表示されるまでの時間であり，プログラムサイズには，インタープリターや外部のライブラリーサブルーチン等は含まれていない。

3.2 実行速度

本家のFORTHの資料には，高級言語でプログラムしたものに比べてはるかに速くとだけ書いてあり，STOICの方も，速い実行速度を保ちながらメモリーの使用効率が良いと述べているのみで具体的な数字は示されていない。IDS-FORTHについてはベンチマークテストの結果が公表されており，これによれ

ば，IDS-FORTH は非常に速いことになっている（第4表）[3]。これについてはあとで検討する。

第4表 IDS-FORTHのベンチマークテストの結果

（文献〔3〕P. 164より抜粋）

言語	メーカー	実行時間	プログラムサイズ
アセンブラー	ザイテン	0.24	183
IDS-FORTH	アドテック	0.8	191
12KSUPER BASIC	ザイテン	32.0	751
BASIC COMPILER*	マイクロソフト	2.4	204
PASCAL-Z**	イサカ・インターシステムズ	1.4	207

* 8080のコードにコンパイルするもの
** Z-80のコードにコンパイルするもの

さて我々のテストによれば，両方の問題共に STOIC が最も悪いという結果になった。問題2に至っては最初に作成したもの（STOIC1）が余りにも遅いのでその原因を追究し，改良したものがSTOIC2である。原因は，最も多く繰返されるところのDO LOOP にあったので，これをwhile～do型のBEGIN～IF～REPEAT に置き換えた。これにより約3割速くなったが，それでもPASCALより16%も遅い。

なおこの問題では，アセンブラーの速度が他の言語に比較してあまり速くないが，これはこのプログラム中で除算が25133回実行されており，ハードウェア除算命令のないZ-80 ではこの部分だけで約12.6秒（83%）を費しているためである。

以上の結果から少なくとも STOIC は普通のインタープリター型言語より実行速度が速いとはとても言えず，むしろ逆にはるかに遅いことがあると言える。それでは本家FORTH の実行速度はどの程度なのであろうか。これを推定するために STOIC のインタープリターの命令のフェッチからコード部分へジャンプするまでの時間（これを仮にフェッチ時間と呼ぶことにする）がどの程

度かを見てみる。これはインタープリターにスタックのあふれ等のエラーチェック機能を追加したときの実行時間から簡単に推定できる。 8-Qneenの場合そのエラーチェックにより実行時間は14.1秒増加する。これよりインタープリターの命令フェッチは約57万回であり，従って全フェッチ時間は約24.2秒となる。この時間だけでもPLZCGの場合の11倍という値であり，従ってFORTHが“高級言語より速い”ということはまず考えられない。またSTOICはもともと8080のために作成されたものであるから，Z-80版を作る際にZ-80の豊富なレジスターを利用して（IDS-FORTHはそうしている） 改善した場合，フェッチ時間は約35%短縮されるが，全体としては8-Queenの場合で2割程度の改善しか見込めない。

次にSTOICにおいては，プログラムの中に簡単にアセンブラーによるコードを埋め込むことができるということを利用して実行時間の変化を見た（第5表)。改良版1というのはチェスボード中のある場所にQueenを置くことができるかどうかを調べる部分のみをアセンブラーで置き換えたもので，改良版2は，さらにQueenを“置く”部分と“取り除く”部分も置き換えたものである。最も時間を費やしていそうなところをうまくみつけることができる場合は，この例のように簡単に実行速度を上げることができるというのが救いである。

第5表 STOICのプログラムの一部をアセンブラーで置きかえた場合
(8-Queenの問題について)

プログラム	実行時間 (秒)	プログラムサイズ (バイト)
もとの版	40.4	548(412)*
改良版 1	16.5	541(405)*
改良版 2	9.1	518(382)*

* STOICの語の名前とリンクの部分を除いた値

最後にIDS-FORTHのベンチマークテストの結果について触れておく。そこで行なわれているテストの内容は $x^2/50-32$ という二次曲線をVideo RAM型

のCRTにプロットするというものである。これは計算した座標値によりメモリー中のあるビットをon-offする操作を含んでいる。高級言語は普通ビット操作の命令を用意しておらず，この問題がはじめから高級言語に不利であることは明らかである。一方IDS-FORTHはどうかと言うと本家のFORTHにはない新しい語SETBITを基本語として追加してそれを使っているので，これはアセンブラーと同レベルの速度が得られるはずである。従ってこれによってFORTHが速いと主張することは誤っている。

3.2　メモリーの使用効率

単純に比較しただけでは，両方の問題共にSTOICが最悪である。ただしSTOICのディクショナリーの中には，語の名前とリンクの部分が，各エントリー当り8バイトずつ含まれており，そのプログラムを実行するだけのターゲットシステムに於いては不要のものである。この部分を除いて比較すれば，STOICのプログラムサイズはPLZCG，即ちマシン語に落すコンパイラーのものよりは小さくなるが，PASCALよりは悪く，PLZSYSよりははるかに大きい。開発者側の資料によれば，アセンブラーの場合より小さくなるということであるが，この点でも期待はずれである。FORTHが用いているインダイレクト・スレッデッド・コードの手法を用いた場合，8080系の命令に比べて短かくなることがはっきりしているのは手続きの呼び出し命令のみであり，その他の基本的なオペレーションに於いては，同程度かあるいは長くなる傾向にある。ただ一時変数（計算の中間結果，ループの制御変数等）はすべてスタックに積むことになっているため，ハードウェアスタックのない機械に比べればその分は節約できる。

3.3　プログラムの記述性その他

STOICのプログラムは，一応高級言語に近い形で書くことができるため，アセンブラーに比べれば明らかに記述性は良い。しかしPLZSYSやPASCALに比べれば明らかに劣っている。

STOICでは式を記述する場合は，逆ポーランド記法を用いるが，長い式の場合はオペレーターとオペランドがはなれてしまう場合があり，従ってオペレー

ターとオペランドとの対応が一見してわかりにくい。この記法はYHP のポケットカリキュレータのように一命令毎に演算を実行してしまう場合にはよいかもしれないが，プログラム言語としては計算機向きではあっても人間向きではない。

また，スタティックな変数をあまり使用しないようにして実行速度を上げようとするとすぐにスタック操作が複雑になってわかりにくいプログラムになってしまうという欠点もある。

以上悪い面を述べたが良い面としては，語の名前は空白，タブ記号を除くすべての英数字・記号を用いて構成できるため，新しいオペレーターを簡単に定義でき，しかも一担登録すればそのオペレーターをコンパイラーを作り変えたかのように自由に使用できること，オペレーションは常にスタックを対象にするため，答として複数の値を返すような関数も式の中で自由に使えること，STOICはそれ自身の中にアセンブラーを含んでいるため，ソースプログラムの中に自由にアセンブラーコードを埋め込むことができる等の点があげられる。

その他プログラム開発期間がどうなるかという問題がある。これについては今回のテストのみで定量的な評価を下すことは差し控えるが，STOICの原理上コンパイラーの速度が非常に速いということ，デバッグがしやすいということは言える。コンパイルが速いということは，その分人間が苦労していることになる訳だから，それがそのまま開発時間の短縮につながらないが，デバッグがしやすいことは開発期間に大いに影響するであろう。

STOICにおけるデバッグの特徴は，プログラム開発中はメモリ内のディクショナリーの中にすべての語の名前が残っているためすべての変数，関数，手続きはその名前で参照することができることと，すべての手続は，必要な引数をスタックに積んで渡し，答もスタックに積んで返すようになっているため，1つ1つの手続きのテストが大変簡単にできるという点である。前者の機能はまさにシンボリック・デバッガーの機能であり，マイコンの高級言語の場合，その機能のないものが少なくないであろう。

Ⅳ むすび

FORTHやSTOICの開発者側が主張するような良い結果があるいは得られるかも知れないという期待を持ってテストを行なったのであるが，残念ながら実行速度は中間言語を用いるコンパイラー並みかそれ以下であること，メモリー使用効率も機械語に落すコンパイラーより少々良い程度でしかないこと，スタック操作を多用すると式が，従ってプログラムの内容がわかりにくくなること等，悪い結果が目についた。しかしながら，この小さなシステムに，コンパイラー，インタープリター，シンボリック・デバッガー，アセンブラ等の機能を詰め込みながら，言語の柔軟な拡張性をもっている点はユニークであり評価してよい。

その他，プログラム中に容易にアセンブラーによるコードを組み込むことができ，その効果が確認されたこと，デバッギングが容易に行なえることが確認された。

全体として言語開発者及び販売者側の誇大宣伝を証明するだけになった感もあるが，この種の言語は今後ますます増加していくであろうミニ・マイクロコンピューター向け言語としてはユニークであり，良い面もあるので更なる研究が望まれる。

参考文献

〔1〕 岡田　聡，FORTHの特徴と原理，インターフェス　1979年6月号
〔2〕 micro FORTH PRIMER　第2版　FORTH, Inc. (1978)
〔3〕 片桐　明，IDS—FORTHの詳細，インターフェス　1980年5月号
〔4〕 CP/M User's Group No.23a
〔5〕 B. W. Lee, An Introduction to North Star Disk STOIC, Dr.Dobb's Journal Vol.4 No.40 (1979)

Appendix 1. Z80STOICのアセンブラー・コード

Definition of the notation

```
r r': any of the register symbols following
      A  B  C  D  E  H  L  (HL)
dd  : any of the register pair symbols following
      BC  DE  HL  SP
pp  : any of BC DE SP
qq  : any of BC DE HL AF
x   : any of the index register symbols IX IY
n   : 8-bit value or symbol
nn  : 16-bit value or symbol
```

8-bit load

Zilog	Z80STOIC
LD r,r'	r' r LD,
LD r,n	n r LDI,
LD r,(x+d)	d x+ r LD,
LD (x+d),r	r d x+ LD,
LD (x+d),n	n d x+ LD,
LD A,(BC)	(BC) LDAX,
LD A,(DE)	(DE) LDAX,
LD A,(nn)	nn LDA,
LD (BC),A	(BC) STAX,
LD (DE),A	(DE) STAX,
LD (nn),A	nn STAX,
LD A,I	I A LD,
LD A,R	R A LD,
LD I,A	A I LD,
LD R,A	A R LD,

16-bit load and exchange

Zilog	Z80STOIC
LD dd,nn	nn dd LDI,
LD x,nn	nn x LDI,
LD HL,(nn)	nn HL LD,
LD x,(nn)	nn x LD,
LD (nn),HL	HL nn ST,
LD (nn),x	x nn ST,
LD SP,HL	HL LDSP,
LD SP,x	x LDSP,
PUSH qq	qq PUSH,
PUSH x	x PUSH,
POP qq	qq POP,
POP x	x POP,
EX DE,HL	HL DE EX,
EX AF,AF'	AF' AF EX,
EX (SP),HL	HL (SP) EX,
EX (SP),x	x (SP) EX,

Auto increment/decrement and repeat

Zilog	Z80STOIC
LDI	LD+
LDIR	LD+R
LDD	LD-,
LDDR	LD-R,
CPI	CP+,
CPIR	CP+R,
CPD	CP-,
CPDR	CP-R,

8-bit arithmetic and logical

Zilog	Z80STOIC	
ADD A,r	r A ADD,	
ADD A,n	n ADDI,	
ADD A,(x+d)	d x+ A ADD,	
ADC A,s	s A ADC,	s is any of r or d x+
ADC A,n	n A ADCI,	shown for ADD
SUB s	s SUB,	
SUB n	n SUBI,	
SBC A,s	s A SBC,	
SBC A,n	n A SBCI,	
AND s	s AND,	
AND n	n ANDI,	
OR s	s OR,	
OR n	n ORI,	
XOR s	s XOR,	
XOR n	n XORI,	
CP s	s CP,	
CP n	n CPI,	
INC s	s INC,	
DEC s	s DEC,	

16-bit arithmetic

Zilog	Z80STOIC
ADD HL,ss	ss HL ADD,
ADC HL,ss	ss HL ADC,
SBC HL,ss	ss HL SBC,
ADD x,pp	pp x ADD,
INC ss	ss INC,
INC x	x INC,
DEC ss	ss DEC,
DEC x	x DEC,

shift and rotate

Zilog	Z80STOIC
RLCA	RLCA,
RLA	RLA,
RRCA	RRCA,

```
        RRA             RRA,
        RLC r           r RLC,
        RLC (x+d)       d x+ RLC,
        RL m            m RL,           m is any of r or
        RRC m           m RRC,          d x+ as shown
        RR m            m RR,           for RLC
        SLA m           m SLA,
        SRA m           m SRA,
        SRL m           m SRL,
        RLD             RLD,
        RRD             RRD,
```

bit operation

```
        Zilog           Z80SIOIC

        BIT b,r         r b BIT         b is the bit
        BIT b,(x+d)     d x+ b BIT,     position
        SET b,r         r b SET,
        SET b,(x+d)     d x+ b SET,
        RES b,r         r b RES,
        RES b,(x+d)     d x+ b RES,
```

Backward jump

```
        Zilog           Z80SIOIC

        JP nn           nn JP,
        JPcc nn         nn JPcc,        cc is any of following
        JR e            e JR,           NZ : non zero
        JRC e           e JRC,          Z  : zero
        JRNC e          e JRNC,         NC : non carry
        JRZ e           e JRZ,          C  : carry
        JRNZ e          e JRNZ,         PO : parity odd
        JP (HL)         JP(HL),         PE : parity even
        JP (x)          JP(x),          NV : not overflow
        DJNZ e          e DJNZ,         V  : overflow
```

Forward jump

```
        Zilog           Z80STOIC

        JP nn           IF, ... FI,
        JP cc,nn        IFcc', ... FI,  cc' is the negated
        JR e            IFR, ... FI,    condition of cc
        JR C,e          IFRNC, ... FI,
        JR NC,e         IFRC, ... FI,
        JR Z,e          IFRNZ, ... FI,
        JR NZ,e         IFRZ, ... FI,
```

Call and Return

```
        Zilog           Z80STOIC

        CALL nn         nn CALL,
        CALL cc,nn      nn CALcc,
        RET             RET,
        RET cc          RETcc,
```

```
        RETI            RETI,
        RETN            RETN,
        RST n           n RST,

Input and Output

        Zilog           Z80STOIC

        IN A,(n)        n INA,
        IN r,(C)        r IN(C),
        INI             IN+,
        INIR            IN+R,
        IND             IN-,
        INDR            IN-R,
        OUT (n),A       n OUIA,
        OUT (C),r       r OUT(C),
        OUTI            OUT+,
        OTIR            OT+R,
        OUTD            OUT-,
        OTDR            OT-R,

Miscellaneous

        Ziolg           Z80STOIC

        DAA             DAA,
        CPL             CPL,
        NEG             NEG,
        CCF             CCF,
        SCF             SCF,
        NOP             NOP,
        HALT            HALT,
        DI              DI,
        EI              EI,
        IM 0            IM0,
        IM 1            IM1,
        IM 2            IM2,
```

Appendix 2. 使用した言語及び言語プロセッサー

A2.1 PLZSYS／PLZCG

PLZというのは，Zilog社が自社のマイクロコンピューター用に開発した，どちらかと言えばPascal風のシステム記述用言語である。PLZSYSはそのコンパイラーであり，ソースプログラムをZ-codeというマシン・インデペンデントな中間言語に翻訳する。Z-codeになったプログラムは，機械に固有のZINTERPというインタープリターにより実行される。PLZCGはZ-codeをZ-80の機械語に変換するcode generatorである。従ってPLZで書かれたプログラムは，Z-codeにしてインタープリターでも実行できるし，さらにPLZCGにより機械語のプログラムとしても実行できる。変数の基本の型はbyte, short integer（以上1バイト長），word, integer（以上2バイト長）の4種類あり，1語1バイトのZ-80を意識したものになっている。しかしZ-codeの段階では，その内容については公表されていないのではっきりしないが，1語が16ビットの機械向けに設計されているらしく，なまじ1バイト変数を使うとかえって実行速度が遅くなり，プログラムサイズも大きくなった。測定結果（第3表）中の8-QueenのPLZSYSの値は掲載したプログラムそのものの値ではなくN, K, Hの3変数をword型にしたものの値である。byteのままのものは実行時間で1.7秒，プログラムサイズで9バイト劣る。なお使用したのはVersion 3.0である。

A2.2 Pascal

PascalはN. Wirthによって開発された教育用の構造化プログラム向けの言語で，ALGOL60の流れを汲んでいる。使用したプロセッサーは，K. L. Bowlesらがミニコン・マイコン用に作成したUCSD Pascal version 1.5である。このインタープリターは8080用のものとZ-80用のものと両方の版があるが，使用したのはZ-80用のものである。Pascalにおける変数の基本の型はinteger, boolean（以上2バイト長），Real（4バイト長）である。integerは常に符号付き2進数として計算されるため，Z-80上で走らせる場合は符号のないデータとしての演算が指定できるPLZやSTOICに比べて少し不利になると思われる。

なおコンパイルする際は変数のサブレンジを変域とする変数の値をチエックしないというオプション{$R—}を指定した。これにより8-Queenは19%，素数の方は13%実行時間が短縮された。

Appendix 3. プログラムリスト

A3.1 アセンブラーによる 8-Queen問題

```
;
;N-QUEEN PROBLEM IN ASSEMBLER
;
        GLOBAL          NQUEEN
        EXTERNAL        PUTDEC
;
NQ:     EQU     8
ALEN:   EQU     NQ+(2*NQ-1)*2   ;SIZE OF WORKING AREA
;
NQUEEN: ;INITIALIZE EMPTY BOARD
        LD      IY,0            ;COUNTER OF SOLUTIONS := 0
        LD      HL,COL
        LD      DE,COL+1
        LD      BC,ALEN-1
        LD      (HL),1
        LDIR
        LD      BC,0            ;N:=0
        LD      IX,X
        CALL    GENER
        PUSH    IY
        POP     HL
        CALL    PUTDEC
        RET
;
GENER:  LD      DE,NQ           ;H:=NQ
NEXTH:  DEC     E
        RET     M               ;IF H<0 THEN RETURN
;
        LD      HL,COL          ;(N,H) FREE?
        ADD     HL,DE           ;COL[H]=TRUE?
        LD      A,(HL)
        AND     A
        JP      Z,NEXTH
        LD      HL,DOWN         ;DOWN[N+H]=TRUE?
        ADD     HL,BC
        ADD     HL,DE
        AND     (HL)
        JP      Z,NEXTH
        LD      HL,UP           ;UP[N-H]=TRUE?
        ADD     HL,BC
        SBC     HL,DE
        AND     (HL)
        JP      Z,NEXTH
;
```

```
        LD      (IX),E          ;X[N]:=H
        LD      HL,COL          ;COL[H]:=FALSE
        ADD     HL,DE
        DEC     (HL)
        PUSH    HL
        LD      HL,DOWN         ;DOWN[N+H]:=FALSE
        ADD     HL,BC
        ADD     HL,DE
        DEC     (HL)
        PUSH    HL
        LD      HL,UP           ;UP[N-H]:=FALSE
        ADD     HL,BC
        SBC     HL,DE
        DEC     (HL)
        PUSH    HL
;
        LD      A,C             ;N=NQ-1?
        CP      NQ-1
        JR      Z,COUNT
         INC    C               ;N:=N+1
         INC    IX
         PUSH   DE              ;SAVE H
         CALL   GENER
         POP    DE              ;RESTORE H
         DEC    C               ;N:=N-1
         DEC    IX
         JP     REMOVE
COUNT:  INC     IY
REMOVE: POP     HL              ;UP[N-H]:=TRUE
        INC     (HL)
        POP     HL              ;DOWN[N+H]:=TRUE
        INC     (HL)
        POP     HL              ;COL[H]:=TRUE
        INC     (HL)
        JP      NEXTH
;
X:      DEFS    NQ
COL:    DEFS    NQ
        DEFS    NQ-1
UP:     DEFS    NQ
DOWN:   DEFS    2*NQ-1

        END
```

A3.2 STOICによる 8-Queen問題

```
% N QUEEN PROGRAM IN STOIC
% USING BYTE ARRAYS
DECIMAL
% CONSTANTS
        8 'NQ CONSTANT
        NQ 1- 'BIAS CONSTANT
        255 'TRUE CONSTANT      0 'FALSE CONSTANT

% VARIABLES AND ARRAYS
        0 'N VARIABLE
        0 'CNT VARIABLE
        NQ 1+ 2/ DUP    'X ARRAY        'COL ARRAY
        NQ DUP          'UP ARRAY       'DOWN ARRAY

'FREE?  :
        COL I + B@ UP N B@ I - BIAS + + B@ AND
                DOWN N B@ I + + B@ AND
        ;

'SETQ :
        I X N B@ + B!                           % X[N]:=H
        FALSE COL I + B!                        % COL[H]:=FALSE
        FALSE UP N B@ I - BIAS + + B!           % UP[N-H+BIAS]:=FALSE
        FALSE DOWN N B@ I + + B!                % DOWN[N+H]:=FALSE
        ;
'PUTCONF :
        NQ 0 DO X I + B@ = LOOP CR      ;
'REMOVQ :
        TRUE DOWN N B@ I + + B!                 % DOWN[N+H]:=TRUE
        TRUE UP N B@ I - BIAS + + B!            % UP[N-H+BIAS]:=TRUE
        TRUE COL I + B!                         % COL[H]:=TREU
        ;
0 'GENENT VARIABLE                              % ENTRY OF GENERATE
'GENERATE :
        NQ 0 DO
        FREE? IF SETQ N 1+! N B@ NQ EQ IF CNT 1+!
                                ELSE GENENT @ EXEC FI
                N 1-! REMOVQ
              FI
        LOOP
        ;
() GENERATE GENENT !
% MAIN PROGRAM
'NQUEEN :
        0 DUP N ! CNT !
        NQ 0 DO TRUE COL I + B! LOOP
        NQ 2* 1- 0 DO TRUE DUP UP I + B! DOWN I + B! LOOP
        GENERATE
        CNT ?
        ;
;F
```

A3.3 PLZ による 8-Queen 問題

```
! N Queen Program in PLZ !
NQUEEN  MODULE

CONSTANT                ! Input/Output unit numbers !
        CONOUT := 2     ! Unit number for console output !
        SYSLIST := 3    ! Unit number for listing device !
        ASCICR := %0D
EXTERNAL
        PUTWORD PROCEDURE(UNIT BYTE, W WORD)
        PUTCHAR PROCEDURE(UNIT BYTE, C BYTE)
        OPEN PROCEDURE(UNIT BYTE, P *BYTE, FLAG BYTE) RETURNS(RCODE BYTE)
        CLOSE PROCEDURE(UNIT BYTE) RETURNS(RCODE BYTE)
INTERNAL
        PRINT PROCEDURE ( B BYTE )
           ENTRY        PUTWORD( COUNT, WORD B )        END PRINT
        NEWLINE PROCEDURE
           ENTRY        PUTCHAR( COUNT, ASCICR )        END NEWLINE
CONSTANT
        TRUE := 1       FALSE := 0
        NQ := 8         BIAS := NQ-1
INTERNAL
        N, K :          BYTE
        COUNT :         WORD
        X, COL :        ARRAY[NQ BYTE]
        UP, DOWN :      ARRAY [2*NQ-1 BYTE]

GENERATE PROCEDURE
   LOCAL        H BYTE
   ENTRY
        H := 0
        DO
          IF COL[H]=TRUE ANDIF UP[N-H+BIAS]=TRUE ANDIF DOWN[N+H]=TRUE THEN
            X[N].:= H; COL[H]:=FALSE; UP[N-H+BIAS]:=FALSE;
            DOWN[N+H]:=FALSE; N+=1
            IF N=NQ !board full! THEN
                ! K:=0 DO PRINT( X[K] ); K+=1 IF K=NQ THEN EXIT FI OD
                NEWLINE         !
                COUNT += 1
                ELSE GENERATE
            FI
            N -= 1
            DOWN[N+H]:=TRUE; UP[N-H+BIAS]:=TRUE; COL[H]:=TRUE
          FI
          H+=1  IF H=NQ THEN EXIT FI
        OD
 END GENERATE

GLOBAL
NQUEEN PROCEDURE
   ENTRY
        N := 0;         COUNT := 0
        K := 0 DO COL[K]:=TRUE; K+=1; IF K=NQ THEN EXIT FI OD
        K := 0 DO UP[K]:=TRUE;DOWN[K]:=TRUE; K+=1; IF K=2*NQ-1 THEN EXIT FI OD
        GENERATE
        PUTWORD(CONOUT,COUNT)   NEWLINE
   END NQUEEN
END NQUEEN
```

A3.4 Pascal による 8-Queen 問題

```
{$L NQUEEN.LIST} {$R-}
PROGRAM NQUEEN;

CONST   NQ = 8;
        NQ2 = 15; { 2*NQ-1 }
VAR     N, K : INTEGER;
        X : ARRAY[0..NQ] OF INTEGER;
        COL : ARRAY [0..NQ] OF BOOLEAN;
        UP  : ARRAY [-NQ..NQ] OF BOOLEAN;
        DOWN: ARRAY [0..NQ2] OF BOOLEAN;
        CNT : INTEGER;

PROCEDURE GENERATE;

VAR     H : INTEGER;
BEGIN
  H := 0;
  REPEAT IF COL[H] AND UP[N-H] AND DOWN[N+H] THEN
     BEGIN
       X[N]:=H; COL[H]:=FALSE; UP[N-H]:=FALSE;
       DOWN[N+H]:=FALSE; N:=N+1;
           IF N=NQ THEN
             BEGIN CNT:=CNT+1;
             { K:=0;
               REPEAT WRITE(X[K]); K:=K+1; UNTIL K=NQ; WRITELN;
             }
             END
             ELSE GENERATE;
           N:=N-1;
           DOWN[N+H]:=TRUE; UP[N-H]:=TRUE; COL[H]:=TRUE
     END;
     H:=H+1
  UNTIL H=NQ
END;

{ ENTRY OF NQUEEN }
BEGIN
  N:=0;  CNT:=0;
  K:=0; REPEAT COL[K]:=TRUE; K:=K+1 UNTIL K=NQ;
  K:=0; REPEAT UP[K-NQ+1]:=TRUE; DOWN[K]:=TRUE; K:=K+1 UNTIL K=2*NQ-1;
  GENERATE;
  WRITELN(CNT)
END {NQUEEN}.
```

A3.5 アセンブラーによる素数の問題

```
; FIND N PRIME NUMBERS
        GLOBAL          PRIME
        EXTERNAL        DIVIDE,PUTDEC
N:      EQU     1000
PRIME:
        LD      IY,PTAB+2*2
        LD      HL,2
        LD      (CNT),HL
        INC     HL
        LD      (X),HL
NEXT:   LD      HL,(X)          ;X:=X+2
        INC     HL
        INC     HL
        LD      (X),HL
        LD      IX,PTAB+2
TEST:   LD      E,(IX)          ;DE:=PTAB[k]
        INC     IX
        LD      D,(IX)
        INC     IX
        LD      BC,(X)
        LD      HL,0
        CALL    DIVIDE          ;X / PTAB[k]
        LD      A,H
        OR      L
        JR      Z,NEXT          ;if remainder=0 then goto NEXT
        EX      DE,HL
        SBC     HL,BC
        JP      C,TEST
STORE:  LD      HL,(X)
        LD      (IY),L
        LD      (IY+1),H
        INC     IY
        INC     IY
        LD      HL,(CNT)
        INC     HL
        LD      (CNT),HL
        LD      DE,N
        AND     A
        SBC     HL,DE
        JP      NZ,NEXT
; PRINT THE LIST OF PRIMES
        LD      BC,N
        LD      IX,PTAB
PLOOP:  LD      L,(IX)
        LD      H,(IX+1)
        CALL    PUTDEC
        DEC     BC
        LD      A,B
        OR      C
        RET     Z
        INC     IX
        INC     IX
        JR      PLOOP
;
X:      DEFS    2       ;CANDINATE OF PRIME
CNT:    DEFS    2       ;NUMBER OF PRIMES FOUND
PTAB:   DEFW    2       ;TABLE OF PRIMES
        DEFW    3
        DEFS    N*2-4
        END
```

A3.6　STOIC による素数の問題

```
%       FINED FIRST N PRIME NUMBERS
DECIMAL
% CONSTANT
        1000 'N CONSTANT
% VARIBLES AND ARRAY
        0 'X VARIABLE
        0 'LIM VARIABLE
        0 'L VARIABLE
        0 'SQR VARIABLE
        0 'PRIM  VARIABLE
        N 'P ARRAY
'PRIME  ;
        % INITIALIZE
        P 2 <-
        1 X !   0 LIM !          4 SQR !
        N 1 DO
           BEGIN
                2 X @ + X !
                SQR @ X @ ULE IF LIM @ 1+ DUP LIM ! 2* P + @ DUP U* SQR ! FI
                -1 PRIM !        1 L !
                BEGIN PRIM @ L @  LIM @  ULT AND IF
                        X @ P L @ 2* + @ UMOD NEZ PRIM !
                        L 1+!
                REPEAT
           PRIM @
           END
           X @ P I 2* + !
        LOOP
        % PRINT OUT
        P N 0 DO DUP @ <#> DUP 8 SWAP - ( SPACE ) TYPE 2 + LOOP
        ;

;F
```

A3.7 PLZ による素数の問題

```
! find first n prime numbers !
PRIME   MODULE
CONSTANT                ! Input/Output unit numbers !
        CONOUT := 2     ! Unit number for console output !
        SYSLIST := 3    ! Unit number for listing device !
EXTERNAL
        NUMFIELDSIZE BYTE       ! Initially 7 !
        PUTWORD PROCEDURE(UNIT BYTE, W WORD)
CONSTANT
        N := 1000
INTERNAL
        P ARRAY (N WORD)
        I, X, LIM, K, SQR: WORD
GLOBAL
PRIME   PROCEDURE
   ENTRY
        NUMFIELDSIZE := 8
        P[0]:=2; X:=1; LIM:=0; I:=1; SQR:=4;
        DO
           NEXT:DO
                        X += 2
                        IF SQR<=X THEN LIM+=1 SQR:=P[LIM]*P[LIM] FI
                        K := 1
                        DO
                                IF K>=LIM THEN EXIT FROM NEXT FI
                                IF (X MOD P[K])=0 THEN REPEAT FROM NEXT FI
                                K +=1
                        OD
                OD
                P[I]:=X
                I+=1    IF I=N THEN EXIT FI
        OD
        I := 0
        DO
                PUTWORD(CONOUT, P[I])
                I+=1 IF I=N THEN EXIT FI
        OD
   END PRIME
END PRIME
```

A3.8 Pascal による素数の問題

```
{$L PRIME.LIST} {$R-}
PROGRAM PRIME;
CONST    N=1000;
TYPE     INDEX=1..N;
VAR
         X,SQR: INTEGER;
         I,K,LIM: INDEX;
         PRIM: BOOLEAN;
         P: ARRAY [INDEX] OF INTEGER;
BEGIN
        P[1]:=2; X:=1; LIM:=1;
        SQR:=4;
        FOR I:=2 TO N DO
        BEGIN
          REPEAT X:=X+2;
            IF SQR<=X THEN BEGIN LIM:=LIM+1; SQR:=P[LIM]*P[LIM] END;
            K:=2; PRIM:=TRUE;
            WHILE PRIM AND (K<LIM) DO
              BEGIN
                PRIM:=(X MOD P[K])<>0;
                K:=K+1
              END
            UNTIL PRIM;
         P[I]:=X;
        END;
        FOR I:=1 TO N DO WRITE(P[I]:8)
END.
```

A3.9　STOIC の一部にアセンブラーを用いた 8-Queen 問題

改良版 1

```
% N-QUEEN PROGRAM IN STOIC USING ASSEMBLER FOR FREE?
DECIMAL
% CONSTANTS
        8 'NQ CONSTANT
        NQ 1- 'BIAS CONSTANT
        255 'TRUE CONSTANT      0 'FALSE CONSTANT
% VARIABLES AND ARRAYS
        0 'N VARIABLE
        0 'CNT VARIABLE
        NQ 1+ 2/ DUP     'X ARRAY          'COL ARRAY
        NQ DUP           'UP ARRAY         'DOWN ARRAY
'FREE? CODE< .L HL LD,   (HL) E LD,        A XOR,          % DE:=I
        A D LD,          N BC LD,          COL HL LDI,     % COL[I]
        DE HL ADD,       A DEC,            (HL) AND,
        DOWN HL LDI,     BC HL ADD,        DE HL ADD,      % DOWN[N+I]
        (HL) AND,
        UP HL LDI,       DE HL SBC,        BC HL ADD,      % UP[N-I+BIAS]
        BIAS DE LDI,     DE HL ADD,        (HL) AND,
        0PUSH JPZ,       -1PUSH JP,        >
'SETQ :
        I X N B@ + B!                      % X[N]:=H
        FALSE COL I + B!                   % COL[H]:=FALSE
        FALSE UP N B@ I - BIAS + + B!      % UP[N-H+BIAS]:=FALSE
        FALSE DOWN N B@ I + + B!           % DOWN[N+H]:=FALSE
        ;
'PUTCONF :
        NQ 0 DO X I + B@ = LOOP CR         ;
'REMOVQ :
        TRUE DOWN N B@ I + + B!            % DOWN[N+H]:=TRUE
        TRUE UP N B@ I - BIAS + + B!       % UP[N-H+BIAS]:=TRUE
        TRUE COL I + B!                    % COL[H]:=TREU
        ;
0 'GENENT VARIABLE                         % ENTRY OF GENERATE
'GENERATE :
        NQ 0 DO
        FREE? IF SETQ N 1+! N B@ NQ EQ IF CNT 1+!
                                 ELSE GENENT @ EXEC FI
                N 1-! REMOVQ
              FI
        LOOP
        ;
() GENERATE GENENT !
% MAIN PROGRAM
'NQUEEN :
        0 DUP N ! CNT !
        NQ 0 DO TRUE COL I + B! LOOP
        NQ 2* 1- 0 DO TRUE DUP UP I + B! DOWN I + B! LOOP
        GENERATE
        CNT ?
        ;
;F
```

A3.10 STOIC の一部にアセンブラーを用いた 8-Qneen 問題

改良版 2

```
% N QUEEN PROGRAM IN STOIC
% USING ASSEMBLER FOR FREE?, SETQ & REMOVQ
DECIMAL
% CONSTANTS
        8 'NQ CONSTANT
        NQ 1- 'BIAS CONSTANT
        255 'TRUE CONSTANT      0 'FALSE CONSTANT
% VARIABLES AND ARRAYS
        0 'N VARIABLE
        0 'CNT VARIABLE
        NQ 1+ 2/ DUP    'X ARRAY        'COL ARRAY
        NQ DUP          'UP ARRAY       'DOWN ARRAY
'FREE? CODE< .L HL LD,  (HL) E LD,      A XOR,          % DE:=I
        A D LD,         N BC LD,        COL HL LDI,     % COL[I]
        DE HL ADD,      A DEC,          (HL) AND,
        DOWN HL LDI,    BC HL ADD,      DE HL ADD,      % DOWN[N+I]
        (HL) AND,
        UP HL LDI,      DE HL SBC,      BC HL ADD,      % UP[N-I+BIAS]
        BIAS DE LDI,    DE HL ADD,      (HL) AND,
        0PUSH JPZ,      -1PUSH JP,      >
'SETQ CODE< .L HL LD, (HL) E LD, 0 D LDI,               % GET H LOOP COUNTER
        X HL LDI, N LDA, A C LD, 0 B LDI, BC HL ADD,    % X[N]:=H
        E (HL) LD,
        COL HL LDI, DE HL ADD, D (HL) LD,               % COL[H]:=0 (FALSE)
        UP HL LDI, E SUB, BIAS ADDI, A C LD,            % UP[N-H+BIAS]=0
        BC HL ADD, D (HL) LD,
        DOWN HL LDI, N LDA, E A ADD, A C LD,            % DOWN[N+H]=0
        BC HL ADD, 0 (HL) LD, NEXT JP,          >
'PUTCONF :
        NQ 0 DO X I + B@ = LOOP CR      ;
'REMOVQ CODE< .L HL LD, (HL) E LD, 0 D LDI,     % GET LOOP COUNTER H
        DOWN HL LDI, N LDA, E A ADD, A C LD,    % DOWN[N+H]=TRUE
        0 B LDI, BC HL ADD, TRUE (HL) LDI,
        UP HL LDI, N LDA, E SUB, BIAS ADDI,     % UP[N-H+BIAS]:=TRUE
        A C LD, BC HL ADD, TRUE (HL) LDI,
        COL HL LDI, DE HL ADD, TRUE (HL) LDI,   % COL[H]:=TRUE
        NEXT JP,                        >
0 'GENENT VARIABLE                              % ENTRY OF GENERATE
'GENERATE :
        NQ 0 DO
        FREE? IF SETQ N 1+! N B@ NQ EQ IF CNT 1+!
                                ELSE GENENT @ EXEC FI
                N 1-! REMOVQ
              FI
        LOOP
        ;
() GENERATE GENENT !
% MAIN PROGRAM
'NQUEEN :
        0 DUP N ! CNT !
        NQ 0 DO TRUE COL I + B! LOOP
        NQ 2* 1- 0 DO TRUE DUP UP I + B! DOWN I + B! LOOP
        GENERATE
        CNT ?
        ;
;F
```