一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
SS2017-18,KBSE2017-18(2017-07)

再帰呼び出しを持つC言語サブセットからMalbolgeへのコンパイラ

坂梨　元軌†　河邉　翔平*††　酒井　正彦†††　西田　直樹†††　橋本　健二†††

†,††† 名古屋大学 大学院情報学研究科
†† 名古屋大学 大学院情報科学研究科
E-mail: †sakanashi@trs.css.i.nagoya-u.ac.jp, ††kobe@trs.cm.is.nagoya-u.ac.jp,
†††{sakai,nishida,k-hasimt}@i.nagoya-u.ac.jp

あらまし　難読プログラミング言語 Malbolge は，その解析困難性により知的財産権の保護などに役立つと考えられているが，命令が特殊であるためプログラムの作成は非常に困難である．そのため，Malbolge プログラムを生成するための中間言語として制御付き疑似命令列が提案されているが，C などの通常の言語と比較すると依然としてプログラミングが困難である．本稿では，整数型と真偽型を扱え，while 文などの基本的な制御構造と再帰関数を定義できる C 言語のサブセットのプログラムから Malbolge コードへのコンパイラの実現法を述べる．コンパイラの実現のために，まず，既存の制御付き疑似命令列に配列構文と関数構文を追加し，それにあわせて既存の制御付き疑似命令列から Malbolge への変換系を拡張する．さらに C 言語のサブセットから制御付き疑似命令列へ変換する方法を提案する．

キーワード　難読化，難解プログラミング言語，Malbolge

A compiler that translates to Malbolge from a C-language subset containing recursive calls

Genki SAKANASHI†, Shohei KOBE*††, Masahiko SAKAI†††,
Naoki NISHIDA†††, and Kenji HASHIMOTO†††

†,††† Graduate School of Informatics, Nagoya University
†† Graduate School of Information Science, Nagoya University
E-mail: †sakanashi@trs.css.i.nagoya-u.ac.jp, ††kobe@trs.cm.is.nagoya-u.ac.jp,
†††{sakai,nishida,k-hasimt}@i.nagoya-u.ac.jp

Abstract　*Malbolge* is an esoteric programming language, which is promising to protect intellectual property rights due to its difficulty of analysis. It is, however, very difficult to program because of its peculiar instructions. Tackling this problem, *pseudo-instruction sequences* was developed as an intermediate language for generating Malbolge programs. Nevertheless it is still difficult to program compared with ordinary languages like C. In this article, we present how to implement a compiler that translates to Malbolge from a C-language subset containing the integer type, the Boolean type, basic control structures such as while statement, and recursive calls. In the implementation, we firstly added array syntax and function syntax to pseudo-instruction sequences, and strengthen existing tools for the extention to conform with it. We next propose a translation method from C-language subset to pseudo-instruction sequences.

Key words　obfuscation, esoteric programming language, Malbolge

1 はじめに

難解プログラミング言語は意図的に読解が困難になるように設計されたプログラミング言語であり，その解析困難性により知的財産権の保護などに役立つと期待されている．この目的には難解性が高いほうが好ましいため，特に難解性が高い Malbolge [1] が適している．しかしながら Malbolge は難解なばかりでなくプログラム作成すら困難である．

これまでの研究により Malbolge のプログラミング環境は次第に整ってきている．まず飯澤らによって Malbolge プログラ

(注*)：現在，株式会社トヨタケーラム

ムを生成する低級アセンブラ [5]~[7] が提案され，実行した命令が書き換わってしまうという問題点が解決されたものの演算後には演算対象の変数の直後にジャンプするという新たな問題が生じ，実行制御が難しいという欠点が生じた．その後，河邉らによって提案された手法 [9] により，条件分岐や定数回ループなどの基本的な制御構造が使用可能な制御付き疑似命令列から低級アセンブリプログラムを生成することが可能になった．制御付き疑似命令列は，Malbolge や低級アセンブリ言語と比較するとプログラミング容易な言語であるが，演算は Malbolge 独特の演算命令のみであるし，条件分岐が非常に制限されているなど，C などの通常のプログラミング言語と比較すると依然としてプログラミングが困難である．

本稿では，以上の背景を踏まえて作成した，高級言語から制御付き疑似命令列を生成するコンパイラについて，関数をどのように変換するかについて述べる．当該の高級言語は，整数型と真偽型が扱え，if 文や while 文などの基本的な制御構造と再帰関数が定義可能な C 言語のサブセットである．関数構文の実現に必要な CALL/RETURN 機能を既存の制御付き疑似命令列の命令の組み合わせで実現することが非常に困難であるため，制御付き疑似命令列に CALL/RETURN 命令を導入することで再帰呼出し機能を持たない関数構文を追加する．また低級アセンブリ言語には，関数構文の実現に必要な間接参照ジャンプ（動的なジャンプ命令）がないため，低級アセンブリ言語の命令拡張機能を利用して動的ジャンプ命令を追加する．さらに，高級言語の再帰関数実現のために必要となる配列操作用の命令を，低級アセンブリ言語および制御付き疑似命令列に追加する．これらを利用することで，高級言語の再帰関数呼び出しが実現できる．なお Olmstead が提案した Malbolge [1] はプログラムとメモリを格納するためのアドレス空間が小さすぎるため，制御付き疑似命令列の実現も困難である．このため本稿ではワード長を 10 桁から 20 桁にすることでアドレス空間を拡張した Malbolge20 [6] を用いる．以下では Olmstead の Malbolge の表記に Malbolge10 を用い，Malbolge10 と Malbolge20 の総称として Malbolge を用いる．

本稿の構成は次のとおりである．3 節では，低級アセンブリ言語への動的ジャンプ命令の追加について述べる．4 節では，制御付き疑似命令列への関数構文の追加について述べる．5 節では低級アセンブリ言語への，6 節では制御付き疑似命令列への配列操作用命令の追加について述べる．7 節では高級言語について述べる．

2 準　　備

ここでは，Malbolge，低級アセンブリ言語，制御付き疑似命令列について，本稿で必要な部分のみ説明する．

2.1 Malbolge

Malbolge10 [1]（Malbolge20 [6]）は仮想機械上で動作する機械語であり，C 言語で実装されたインタプリタでその意味が定められている．仮想機械は A（アキュムレータ），C（命令ポインタ），D（データポインタ）の 3 種類のレジスタとメモリ（mem）を持ち，レジスタに格納できる値の範囲，ならびに，メモリアドレス空間は 1 ワード（Malbolge10 では 3 進数 10 桁，Malbolge20 では 20 桁）である．表 1 に Malbolge の命令を示す．命令 OPR で使用される関数 op は，入力 X, Y を 3 進数で表したときの各桁 X_i，Y_i を特殊な規則 [4] に従って計算する．

表 1　Malbolge の命令

オペコード	表記	説明
i	JMP	C レジスタの更新．C:=mem[D].
j	MOV_D	D レジスタの更新．D:=mem[D].
p	OPR	演算命令．A, mem[D]:=op(A, mem[D]).
*	ROT	右ローテート．A, mem[D]:=rotr(mem[D]).
/	INPUT	入力．A:=getchar().
<	OUTPUT	出力．putchar(A).
o	NOP	無操作．
v	HALT	終了．プログラムの実行を停止．
その他	DUP	無操作．

表 2　低級アセンブリ言語の命令

命令	動作	制約*
ROT *label*	A,[*label*]:=rotr([*label*]); PC:= *label* + 1	PC < *label* ≦ PC + *n*
REV ROT	ROT の復元命令; PC:= PC + 1	
OPR *label*	A,[*label*]:=op(A,[*label*]); PC:= *label* + 1	PC < *label* ≦ PC + *n*
REV OPR	OPR の復元命令; PC:= PC + 1	
SKIP *label*	ジャンプ命令; PC:=*label*	PC < *label* ≦ PC + *n*
JMP *label*	ジャンプ命令; PC:=*label*	
REV JMP	JMP の復元命令; PC:= PC + 1	
OUTPUT	putchar(A); PC:= PC + 1	
REV OUTPUT	OUTPUT の復元命令; PC:= PC + 1	
INPUT	A:=getchar(); PC:= PC + 1	
REV INPUT	INPUT の復元命令; PC:= PC + 1	
IF *flag* BRANCH *label*	*flag* が ON の時： PC:=*label* *flag* が OFF の時： PC:= PC + 1 実行後にフラグのサイクル値がインクリメントされる	
NEXT *flag*	*flag* のサイクル値をインクリメント; PC:= PC + 1	
DUP	無操作	

* n はアセンブラの実装から定まる自然数 [10]

また Malbolge は命令実行直後に変換表 xlat2 を用いて mem[C] := xlat2[mem[C] − 33] を実行し，C レジスタが指すメモリの値を置き換える [2]．その後，C レジスタと D レジスタの値をインクリメントする．C レジスタは命令ポインタであるため，実行後に命令が書き換わることになる．

2.2 低級アセンブリ言語

低級アセンブリ言語の仕様と，その命令の実現方法について説明する．

2.2.1 仕　　様

低級アセンブリ言語 [5], [10] は，Malbolge プログラミングを容易にするために開発された言語であり，低級アセンブラにより Malbolge へ変換できる．各命令（表 2）には「*label* : 命令」のようにラベルを付加することができ，*label* はその命令が格納されたアドレスを表す．なお表 2 において，A はアキュムレータの値を，PC はプログラムカウンタの値を，[*label*] は *label* が指すメモリの値を表す．

図 3 に低級アセンブリプログラムの例を示す．このプログラムは，まず変数 X に対して rotr 演算を実行し，次に変数 Y に対して op 演算を実行し，最後に変数 Z に対して rotr 演算を実行するものである．

2.2.2 命令ユニットを利用した命令の実現方法

低級アセンブリ言語の命令は，C レジスタの動作を定義する Malbolge 命令列からなる命令ユニットを利用して実現されている [5]．ROT 命令と OPR 命令の命令ユニットを図 4 と図 5 にそれぞれ示す．

```
1     ROT X
2  X: 10
3     REV ROT
4     OPR Y
5  Y: 20
6     REV OPR
7     ROT Z
8  Z: 30
9     REV ROT
10
```

図3　低級アセンブリプログラムの例

```
      NOP
ROT:  ROT
REV_ROT: JMP
```

図4　ROT 命令の命令ユニット [5]

```
      NOP
OPR:  OPR
REV_OPR: JMP
```

図5　OPR 命令の命令ユニット [5]

表6　メモリの状態

	label	addr	data	data の意味
ROT ユニット		58	104	NOP 命令
	ROT	59	74	ROT 命令
	REV_ROT	60	38	JMP 命令
OPR ユニット		81	81	NOP 命令
	OPR	82	74	OPR 命令
	REV_OPR	83	109	JMP 命令
	X	200	10	X の値
		201	59	REV_ROT－1
		202	81	OPR－1
	Y	203	20	Y の値
		204	82	REV_OPR－1
		205	58	ROT－1
	Z	206	30	Z の値
		207	59	REV_ROT－1

図3の低級アセンブリプログラムから得られる Malbolge プログラムは，その実行により表6に示す Malbolge のメモリの状態となるよう設計されている(注*)．C レジスタが 59 番地，D レジスタが 200 番地をそれぞれ指しているとして，プログラム動作を説明する．

(1) C=59, D=200: ROT X を計算する．このとき 59 番地の ROT 命令は xlat2 の変換によって書き換えられる．

(2) C=60, D=201: JMP (REV_ROT-1) が実行され，C レジスタが 59 番地にジャンプする．xlat2 による変換は命令実行後に行われるため，ジャンプ後の 59 番地の値が変換される．各命令ごとに xlat2 の変換が変換周期が 2 となるアドレスが知られており [5]，59 番地は ROT 命令に対する該当アドレスの一つである．したがって，本ステップにより ROT 命令が復元される．

(3) C=60, D=202: JMP (OPR-1) が実行され，C レジスタが 81 番地にジャンプする．

(4) C=82, D=203: OPR Y を計算する．このとき 82 番地の OPR 命令は xlat2 の変換によって書き換えられる．

(5) C=83, D=204: JMP (REV_OPR-1) が実行され，C レジスタが 82 番地にジャンプする．このとき 59 番地の ROT の場合と同様に 82 番地の OPR が復元される．

(6) C=83, D=205: JMP (ROT-1) が実行され，C レジスタが 58 番地にジャンプする．

(注*)：表6中の addr 欄には説明の便宜上のメモリアドレスが書かれており，実際に図3のプログラムを変換してもそのアドレスに配置されるわけではない．

表7　制御付き疑似命令列の命令

命令	動作
ROT *X*	A,[*X*]:=rotr([*X*]); PC:= PC + 1
OPR *X*	A,[*X*]:=op(A,[*X*]); PC:= PC + 1
INPUT	A:=getchar(); PC:= PC + 1
OUTPUT	putchar(A); PC:= PC + 1
IF *flag* 　命令列 1 ELSE 　命令列 2 IFEND	*flag* が ON の時：PC:= 命令列 1 の先頭 *flag* が OFF の時：PC:= 命令列 2 の先頭
SET *flag*	*flag* を ON にする; PC:= PC + 1
RESET *flag*	*flag* を OFF にする; PC:= PC + 1
REPEAT *n* 　命令列 REPEATEND	*n* が正数の時：*n* 回命令列を実行する *n* が FOREVER の時：無限回命令列を実行する
BREAK	直近の REPEAT を抜ける; PC:= 直近の REPEATEND の直後
SWITCH *X* 　CASE0 　　命令列 1 　CASE1 　　命令列 2 　CASE2 　　命令列 3 SWITCHEND	*X* を 3 進数で表した時の末尾桁の値に応じて分岐する． 末尾桁が 0 の場合：命令列 1 を実行 末尾桁が 1 の場合：命令列 2 を実行 末尾桁が 2 の場合：命令列 3 を実行 ※制約：*X* の末尾以外の桁が 2 であること

(7) C=59, D=206: ROT Z を計算する．このとき 59 番地の ROT 命令は xlat2 の変換によって書き換えられる．

(8) C=60, D=207: JMP (ROT-1) が実行され，C レジスタが 59 番地にジャンプする．このとき 59 番地の ROT が復元される．

命令ユニットの末尾には必ず JMP 命令があるため，実行したい命令の命令ユニットが格納されたアドレスおよび演算対象のデータを並べておきそれを D レジスタで順にアクセスすることで指定した命令を順に実行することができる．すなわち，命令ユニットが配置されたアドレス自体を一種の命令であると考えれば，D レジスタを低級アセンブリ言語上のプログラムカウンタとみなすことができる．なお，説明を簡単にするために各変数をそれぞれ 1 回使用するプログラムを例題に用いたが，同一の変数に対して複数回演算命令を実行したい場合には，たとえ逐次的な実行が記述したい場合であっても図3のような簡潔な記述は不可能である．

低級アセンブリ言語には ROT や OPR などの組み込みの命令ユニット以外にも，ユーザが新たな命令ユニットを定義し，命令として利用する機能がある．

2.3 制御付き疑似命令列

低級アセンブリ言語でのプログラミングは，命令の復元操作が必要なことなどの理由により逐次実行すら複雑な記述を必要とする．そこで，逐次実行が簡潔に記述でき，さらに一部の制御命令を導入した制御付き疑似命令列が開発された [9]．制御付き疑似命令列は表7に示す命令が使用可能で，低級アセンブリ言語に変換可能な言語である．表中の A はアキュムレータを，PC はプログラムカウンタを表す．

3 低級アセンブリ言語への動的ジャンプ命令の追加

制御付き疑似命令列の RETURN 命令を実現するために，低級アセンブリ言語に表8に示す動的ジャンプ命令を追加する．この命令は変数 X に格納された数値をアドレスとみなして，そのアドレスにプログラムカウンタを移動させる命令である．

表 8　DJMP 命令の動作

命令	動作
DJMP X	PC:= [X]

```
1   PROGRAM_START_TO ENTRY@MAIN
2   UNIT{
3           59:DUP
4   DJMP:   60:MOV_D
5   REV_DJMP: 61:JMP
6   }
7
8   ROUTINE MAIN{
9   ENTRY:
10    #JUMP_TO の値（=Y のアドレス）へジャンプ
11    DJMP JUMP_TO
12  #JUMP_TO は Y のアドレスを指す
13  JUMP_TO:Y

          ⋮

20    Y:REV_DJMP
21    END
22  }
23
```

図 9　DJMP を利用した動的ジャンプ(注*)

表 10　メモリの状態

label	addr	data	data の意味
	59	103	DUP 命令
DJMP	60	74	MOV_D 命令
REV_DJMP	61	37	JMP 命令
	⋮		
JUMP_TO	200	300	Y のアドレス
	⋮		
Y	300	60	REV_DJMP − 1
	301	155	END − 1

3.1　動的ジャンプを実現するための命令ユニット

動的ジャンプ命令を低級アセンブリ言語に追加するために，図 9 の 2〜6 行目に示す命令ユニットを設計した．このように命令ユニットを低級アセンブリプログラム中に記述することで，そのプログラム内で DJMP 命令およびその復元のための REV_DJMP 命令が使用可能になる．

3.2　使　用　例

DJMP 命令の例を図 9 に示す．このプログラムは，11 行目の DJMP 命令で 13 行目のラベル JUMP_TO の変数の値である 20 行目のラベル Y に間接ジャンプし，その後 DJMP 命令の命令ユニット内にある MOV_D の復元処理を行い終了するものである．DJMP 命令を実行する際の Malbolge のメモリの状態は表 10 のようになっており，C レジスタは 60 番地を，D レジスタは 200 番地を指している．このプログラムは以下のように動作する．

(1) C = 60, D = 200: MOV_D Y が実行され，D レジスタが 300 番地にジャンプする．このとき 60 番地の MOV_D 命令は xlat2 により書き換えられる．

(2) C = 61, D = 300: JMP (REV_DJMP-1) が実行され，C レジスタが 60 番地にジャンプする．このとき 60 番地の MOV_D 命令が復元される．

(3) C = 61, D = 301: 次の命令（この例の場合は END）の命令ユニットが格納されたアドレスへと C レジスタがジャンプする．

(注*)：図中の ROUTINE 構文はラベルの名前衝突を防ぐための機能であり，CALL/RETURN に相当する機能はない．

```
1   DEF MAIN
          ⋮
10    CALL FUNC     #関数呼び出し
          ⋮
20  END
21  DEF FUNC
          ⋮
30    RETURN
31  END
32
```

図 11　制御付き疑似命令列における関数の例

4　制御付き疑似命令列への関数構文の追加

前節で DJMP 命令を利用することで関数の実現に必要な動的ジャンプを実現できることを示した．本節では制御付き疑似命令列に関数構文を導入し，DJMP 命令を利用する低級アセンブリプログラムへ変換する方法を提案する．

4.1　関 数 構 文

制御付き疑似命令列に導入する関数構文を図 11 に示す．図のように DEF *func_name* と END で囲うことで関数を表す．プログラムの実行は C 言語と同様に MAIN 関数から開始される．CALL 命令を実行することで指定された関数が呼び出され，RETURN 命令を実行することで呼び出し元へ戻り CALL 命令の次の行から処理が継続される．なお変換の複雑さを考慮し，この関数構文では引数及び戻り値の受け渡しには対応しない．

4.2　低級アセンブリ言語による実現

関数呼び出し命令 CALL は，ジャンプ先が固定であるため IF_BRANCH 命令を利用した無条件ジャンプで実現できる．一方 RETURN 命令によるジャンプは関数の呼び元が一般には複数あるため，DJMP 命令を利用する．具体的には，関数からの戻り先のアドレスを格納する変数を用意し，関数呼び出しの際にその変数へ戻り先のアドレスを格納しておき，関数から戻る際はその変数に対して DJMP 命令を実行すればよい．この戻り先アドレスを格納する変数を関数ごとに定義することにより，再帰を含まない多段の関数呼び出しを実現できる．

図 11 の制御付き疑似命令列から変換で得られる低級アセンブリプログラムを図 12 に示す．制御付き疑似命令列の CALL 命令と RETURN 命令は，それぞれ，図 12 の 10〜18 行目と 50 行目に対応する．このプログラムは次のような流れで実行される．

(1) 戻り先アドレスを格納する変数 RETURN_ADDR に対して RETURN_TO_ADDR の値（=RETURN_TO のアドレス）を代入する処理を行う．これは変数のコピーを行う命令列 [5] で実現できる．なお，図 12 ではこの長い命令列を便宜的に *RETURN_ADDR@FUNC = RETURN_TO_ADDR* と一行で記述してある．

(2) 無条件ジャンプにより，関数の処理が記述された箇所へジャンプする．

(3) 関数内の処理が実行される．

(4) DJMP 命令により呼び出し元にジャンプする．ここでは RETURN_ADDR の値は RETURN_TO のアドレスであるため，

```
1   ROUTINE MAIN{
2   ENTRY:
          ⋮
10
11    RETURN_ADDR@FUNC = RETURN_TO_ADDR
12    #関数呼び出し（無条件ジャンプ）
13    NEXT FLAG_JMP
14    IF FLAG_JMP
15    BRANCH ENTRY@FUNC
16
17  RETURN_TO:
18    REV_DJMP
19    #関数呼び出し後の処理
          ⋮
30    END
31  RETURN_TO_ADDR:RETURN_TO
32  }
33  ROUTINE FUNC{
34  ENTRY:
35    #関数内の処理
          ⋮
50    DJMP RETURN_ADDR
51  RETURN_ADDR:0 #初期値は任意
52  }
53
```

図 12　関数を実現する低級アセンブリプログラム

表 13　IND_OPR 命令の動作

命令	動作
IND_OPR X	A,[[X]]:=op(A,[[X]]); PC:= ENTRY@MAIN

17 行目にジャンプする．

（5）DJMP の復元処理等が行われる．

（6）19 行目以降の関数呼び出し後の処理が実行される．

5　低級アセンブリ言語への配列操作用命令の追加

低級アセンブリ言語における配列操作機能は既存研究によって表 13 に示す間接 OPR 命令として実現されている [3]．ここで，表中の ENTRY@MAIN は MAIN ルーチンの ENTRY ラベルを表している．低級アセンブラにおいては，変数に対する操作などのメモリへのアクセスにより，実行が該当のメモリアドレスに移ってしまうという性質がある．このため安藤らの手法 [3] では，配列に用いるアドレスを一つ置きに使い，それらの間のアドレスには ENTRY@MAIN の値を入れておくことで制御可能にしている．Malbolge の性質から，該当アドレスを与えられた値に初期化することは困難であるが一つ置きに同じ値にすることは容易であるため，該当アドレスの値によりジャンプした先に Malbolge10 用の低級アセンブラが他の目的のために生成するコードに合わせて実現していた．しかし，本稿で使用する Malbolge20 用の低級アセンブラでは異なるコードが生成されるため，この実現をそのまま使用することはできない．そこで，Malbolge20 用の低級アセンブラに合わせて IND_OPR 命令を新たに実現する．以下ではこれについてより詳しく述べる．

5.1　安藤らの手法 [3]

Malbolge10 用の低級アセンブラにおける IND_OPR 命令の実現方法について述べる．空いているメモリ空間に対して演算を行うためには，D レジスタをプログラム領域内から該当アドレスにジャンプさせ，演算実行後に制御を元に戻す必要がある．IND_OPR 命令の命令ユニットでは，低級アセンブラが生成する

```
66:       DUP         75:       NOOP
IND_OPR:              76:       MOV_D
67:       MOV_D       UNIT2_4:
UNIT2_1:              77:       NOOP
68:       NOOP        78:       NOOP
69:       NOOP        79:       NOOP
70:       NOOP        80:       NOOP
71:       OPR         81:       NOOP
UNIT2_2:              82:       MOV_D
72:       NOOP        UNIT2_5:
73:       NOOP        83:       MOV_D
UNIT2_3:              UNIT2_6:
74:       MOV_D       84:       JMP
```

図 14　Malbolge20 用 IND_OPR ユニット

表 15　データモジュールに格納されている値

アドレス	値
36	エントリポイントのアドレス
⋮	⋮
42	35
⋮	⋮
83	36

メモリ上の二つの性質を利用して，演算後に元のアドレスへ戻す処理を実現している．

一つ目は，低級アセンブリプログラムを展開した場所より後ろのメモリの偶数番地には，81 が格納されていることである．この値を利用して D レジスタをプログラム内に戻している．

二つ目は，メモリの 81 番地付近に低級アセンブリプログラムのエントリーポイントを示す値が格納されていることである．

一つ目の性質を利用すると，演算実行後に 81 番地に戻ることができる．この付近はデータモジュール [5] が格納されており，入力の低級アセンブリプログラムによらず固定値がいくつか並んでいる．その中の一つがエントリーポイントを示す値となっている（二つ目の性質）ため，D レジスタをプログラムのエントリーポイントに戻すことができる．IND_OPR 命令を利用して配列操作を実現する際には，IND_OPR 命令の実行前にあらかじめエントリーポイント以降に条件分岐を記述しておくことにより，制御を本来の位置，すなわち，IND_OPR 命令の次の命令に戻すことが可能になる．

5.2　Malbolge20 用低級アセンブラへの対応

Malbolge10 用の低級アセンブラと Malbolge20 用の低級アセンブラでは，81 番地付近に格納されている値が異なるため既存の IND_OPR の命令ユニットは使用できない．そこで Malbolge20 用低級アセンブラのために新たに IND_OPR ユニットを設計する（図 14）(注*)．この IND_OPR ユニットでは，データモジュールが表 15 に示すような値であることを利用している．

6　制御付き疑似命令列への配列操作用命令の追加

本節では，前節で低級アセンブリ言語に追加した IND_OPR 命令を利用して，制御付き疑似命令列に表 16 に示す配列操作用命令 IND_OPR 命令を追加する方法について述べる．

制御付き疑似命令列の IND_OPR 命令は，基本的に低級アセンブリ言語の IND_OPR 命令へとそのまま変換すればよい．ただ

(注*)：ここで NOOP は，2.1 節で述べた変換 xlat2 に対して安定な NOP 命令である．

表 16　IND_OPR 命令の動作

命令	動作
IND_OPR X	OPR [X]

```
# PUSH X

# スタックに値を格納する
ROT CON1
OPR TEMP
OPR TEMP
IND_OPR STACK_TOP
IND_OPR STACK_TOP
ROT CON2
OPR X
ROT CON2
OPR X
OPR TEMP
IND_OPR STACK_TOP

# ポインタを移動させる
--STACK_TOP
--STACK_TOP
```

```
# POP X

# ポインタを移動させる
++STACK_TOP
++STACK_TOP

# スタックから値を取り出す
ROT CON1
OPR TEMP
OPR TEMP
OPR X
OPR X
ROT CON2
IND_OPR STACK_TOP
ROT CON2
IND_OPR STACK_TOP
OPR TEMP
OPR X
```

図 17　スタック操作を行う疑似命令列

し，低級アセンブリ言語の IND_OPR 命令の実行後に制御がエントリーポイントに移るため，制御付き疑似命令列の IND_OPR 命令の次に記述された命令に制御を戻す必要がある．そのために，エントリーポイント直後にフラグと IF_BRANCH を利用した条件分岐を挿入することで，IND_OPR 実行後に適切なアドレスにジャンプできるようにする．

7 高級言語

C 言語の機能に制限を設けた高級言語を設計し，高級言語から制御付き疑似命令列に変換するコンパイラを実装する．高級言語で使用可能な機能は，型として int 型と bool 型，制御構造として if 文と while 文，算術演算として加算とインクリメントおよびデクリメント，論理演算として AND/OR/NOT 演算，および引数や戻り値の受け渡しや再帰呼び出しが可能な関数である．ただし int 型は最大値を $3^{20}-1$ とする非負整数である．以下では再帰呼び出しの実現方法についてのみ詳しく説明する．

7.1 再帰関数の実現方法

4 節で述べた制御付き疑似命令列の関数構文では，戻り先を保存するための変数 RETURN_ADDR が関数ごとに一つしか定義できないため，関数を再帰的に呼んだ場合には戻り先のアドレスが保存されず正しく動作しない．そこで，5 節で実現した配列操作用命令 IND_OPR を利用してスタックを実現し，そこに関数の戻り先アドレスを退避することで関数の再帰呼び出しに対応する．スタックへの PUSH/POP を実現する疑似命令列は IND_OPR を利用して図 17 のように実現できる．なお，図中の変数 STACK_TOP は予め定義しておくスタックトップを指す変数であり，++演算子/--演算子はインクリメント/デクリメントを行う疑似命令列 [8] の略記である．

このスタック操作法を利用して，関数呼び出しの前に現在設定されている戻り先アドレスをスタックに PUSH し，関数呼び出し終了後にスタックから POP することで，関数を再帰的に呼び出した場合であっても戻り先アドレスを保持することができるようにする．以下にその変換例を示す．図 18 に示す関数の再帰呼び出しを含む高級言語プログラムを変換したものが図 19 である．

```
1  int func(){
       ⋮
10   func();
       ⋮
20 }
21
```

図 18　関数の再帰呼び出しを含む高級言語プログラム

```
1  DEF FUNC
       ⋮
10   PUSH RETURN_ADDR@FUNC # この時点で設定されている
                           # 戻り先アドレスをスタックに退避
12   CALL FUNC  # 関数呼び出し
13   POP RETURN_ADDR@FUNC # 戻り先アドレスをスタックから復元
       ⋮
30 END
31
```

図 19　図 18 から変換した制御付き疑似命令列

8 まとめ

本稿では，低級アセンブリ言語への動的ジャンプ命令の追加，制御付き疑似命令列への関数構文および配列操作命令の導入，高級言語の設計とそのコンパイラの実現について述べた．この結果と先行研究を併せることで C 言語サブセットから Malbolge20 への変換系が実現でき，プログラミングが容易な高級言語を用いて Malbolge20 プログラムを生成することが可能となった．今後の課題としては，現在の高級言語に不足している算術演算（減算，乗算，除算），配列構文の追加，負数の導入，および生成される Malbolge20 プログラムの高速化がある．

文　献

[1] Ben Olmstead, Malbolge, 1988.

[2] Ben Olmstead, "Reference Malbolge Interpreter", http://www.lscheffer.com/malbolge_interp.html.

[3] 安藤聡，酒井正彦，坂部俊樹，草刈圭一朗，西田直樹，Malbolge の高級アセンブリ言語への配列機能の追加，電子情報通信学会技術報告，Vol.112, No.23, pp.43–48, 2012.

[4] 飯澤恒，坂部俊樹，酒井正彦，草刈圭一朗，西田直樹，難読プログラミング言語 Malbolge におけるプログラム構成手法，電子情報通信学会技術報告，Vol.105，No.129，pp.25–30, 2005.

[5] 飯澤恒，難読言語 Malbolge に基づくプログラム難読化に関する研究，修士学位論文，名古屋大学情報科学研究科，2006.

[6] 加藤起騎，酒井正彦，坂部俊樹，草刈圭一朗，西田直樹，Malbolge のワード長の拡大とそのプログラミング支援ツール，電子情報通信学会技術報告，Vol.113, No.159, pp.73–78, 2013.

[7] 加藤起騎，酒井正彦，坂部俊樹，西田直樹，Malbolge 低級アセンブラにおけるコード配置アドレスの決定法，電子情報通信学会技術報告，Vol.114, No.127, pp.99–104, 2014.

[8] 河邉翔平，増減演算を行う簡潔な Malbolge 低級アセンブリプログラムの開発，卒業論文，名古屋大学工学部，2014.

[9] 河邉翔平，酒井正彦，西田直樹，関浩之，難読性の高い Malbolge コードを生成するコンパイラのための中間言語，電子情報通信学会技術報告，Vol.116，No.127，pp.105–110, 2016.

[10] 長坂哲，難解言語 Malbolge の弱チューリング完全性とプログラミング環境，修士学位論文，名古屋大学情報科学研究科，2010.