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
SS2010-37 (2010-10)

難解言語Malbolgeのチューリング完全性について

長坂 哲† 酒井 正彦†† 坂部 俊樹†† 草刈 圭一朗†† 西田 直樹††

†,†† 名古屋大学 大学院情報科学研究科 〒464-8603 愛知県名古屋市千種区不老町
E-mail: †s_nagasaka@sakabe.i.is.nagoya-u.ac.jp, ††{sakai,sakabe,kusakari,nishida}@is.nagoya-u.ac.jp

あらまし Malbolgeは最も難解なプログラミング言語として知られている．本研究では，飯澤らが提案したプログラミング手法に基づいて，Malbolgeが弱チューリング完全性を持つこと示す．そのために，チューリング完全性を持つ正規形のNプログラムをMalbolgeコードに変換できることを示す．ここで，本稿で示す性質が弱チューリング完全性であるのは，Malbolgeが固定されたメモリ空間およびレジスタ長の仮想機械により意味が定められているためである．
キーワード チューリング完全性，Nプログラム，プログラミング言語，Malbolge

On Turing Completeness of an Esoteric Language, Malbolge

Satoshi NAGASAKA†, Masahiko SAKAI††, Toshiki SAKABE††,
Keiichirou KUSAKARI††, and Naoki NISHIDA††

† Graduate School of Information Science, Nagoya University Furo-cho, Chikusa-ku, Nagoya-City, Aichi, 464-8603 Japan
E-mail: †s_nagasaka@sakabe.i.is.nagoya-u.ac.jp, ††{sakai,sakabe,kusakari,nishida}@is.nagoya-u.ac.jp

Abstract Malbolge is known as one of the most esoteric programming languages. In this paper, we prove that Malbolge is weakly Turing complete. The proof is based on the Malbolge programming method proposed by Iizawa, et al. We give a transformation from the normal form N-programs known to be Turing complete into Malbolge programs. Completeness that this paper shows is weak one due to the fact that the semantics of Malbolge is hard coded into a virtual machine of which memory space and register length are fixed.
Key words Turing Completeness, N-program, programming language, Malbolge

1. はじめに

難解プログラミング言語は難読性や難解性を持つように意図的に設計された言語である．このような言語は，情報セキュリティにおいてプログラムの改ざん防止やソフトウェア保護に役立つと考えられる．難解言語の中でもMalbolge [2] は最も難解な言語の一つであるが，その難解性からプログラムの解読や変更だけでなく直接のプログラミングも非常に困難である．

Malbolgeではこれまでに特定の文字列を出力するプログラムや入力をそのまま出力するプログラムが知られていた [5]．近年，飯澤らが提案した手法 [2], [3] により，困難であると考えられてきたMalbolgeのループプログラム作成法が明らかにされ，ループ耐性を持った低級アセンブリ言語の提案とそれからのMalbolgeコードの生成が可能となった．また，飯澤らは高級アセンブリ言語を提案し，その実現のために，低級アセンブリ言語による基本モジュール方式を提案している．

本稿では，飯澤らの成果を基にMalbolgeが弱チューリング完全性を持つことを証明する．このために，すでにチューリング完全性が知られている正規形のNプログラム [4] が高級アセンブリプログラムで実現できることを示す．さらに，高級アセンブリプログラムを低級アセンブリ言語で実現する手法を，文献 [2], [3] の考え方に基づいて具体的に設計した．

2. Malbolge

ここでは、Malbolgeの仕様についてその重要な部分を説明し、Malbolgeの命令を拡張した疑似命令について紹介する．

2.1 Malbolgeの仕様

Malbolge [1] は仮想機械上で動作する機械語であり，インタプリタによって意味が定められている．仮想機械は三つのレジスタ (A,C,D) とメモリ (mem) を持ち，値は三進数十桁 (10trits) で表現される．よって値は 0000000000t～2222222222t(十進数で 0～59048(= $3^{10}-1$)) となり，メモリのアドレス空間もmem[0]～mem[59048] で定義される．

表1にMalbolgeの命令をわかりやすく表した．

表 1　Malbolge の命令

Table 1　Malbolge Instruction

命令表記	説明
JMP	ジャンプ. C:=mem[D].
MOV_D	D レジスタの更新. D:=mem[D].
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).
NOP	無操作. 何も行わない.
HALT	終了. プログラムの実行を停止.

表 2　op の各桁の演算

Table 2　Operation of Each Trit of 'op'

		X_i		
		0	1	2
	0	1	0	0
Y_i	1	1	0	2
	2	2	2	1

ここで，OPR 命令の演算子 op(X,Y) は 2 引数の各桁同士で表 2 に従って trit 演算を行う命令である．以下にその例を示す．

op(0120120120t, 0001112222t) → 1001022212t

また，ROT 命令の演算子 rotr を例で示すと以下のようになる．

rotr(0001112222t) → 2000111222t

2.2　疑似命令 [2]

疑似命令は，Malbolge の演算命令の表現をより単純化し，引数にメモリ名を指定できるようにしたものである．

例えば，以下の疑似命令列を考える．

```
ROT  X
OPR  Y
```

これは以下の計算を意味する．

```
A,X  :=  rotr(X)
A,Y  :=  op(A,Y)
```

また，疑似命令の引数に定数として Con0(=0000000000t), Con1(=1111111111t), Con2(=2222222222t), Pat21(=2222222221t) を利用できる．

なお文献 [2] では INPUT と OUTPUT に対応する疑似命令が定義されていないが，これらは Malbolge 命令から自然に定義できる．

3.　アセンブリ言語

この章では，文献 [2] で設計された低級アセンブリ言語と高級アセンブリ言語について説明する．

3.1　低級アセンブリ言語

低級アセンブリ言語は，Malbolge プログラムの命令が実行により書き換えられることからループプログラム作成が困難という問題を解決するために設計された言語である [2]．以下ではこの重要な部分について説明する．

低級アセンブリ言語は Malbolge と同じメモリ空間を持つ仮想機械として定義される．レジスタは PC と A の 2 つを持ち，値は 10trits(0〜59048) で表現される．

低級アセンブリプログラムは，メモリアドレスを表すラベルを付加可能なデータの列によって定義される．各データは変数と，命令，および，フラグのいずれかであり，一行で記述される．命令やフラグは引数にラベルを用いている．命令には，演算命令 (U_JMP, U_ROT, U_OPR, U_MOV_PC, U_INPUT, U_OUTPUT) と，U_JMP を除く各演算の復元命令 (R_ROT, R_OPR, R_MOV_PC, R_INPUT, R_OUTPUT) がある．このため，実行時には演算命令が実行された後，復元命令が実行されるようにプログラムを記述する必要がある．フラグはフリップフロップの役割を果たしており，これによって実行を制御する．

表 3　低級アセンブリプログラムの実行ステップ

Table 3　Execution Step of Low Level Assembly Program

step	PC	説明
1	Entry	*1, PC:=Flag_1
2	Flag_1	PC:=Rev_Rot, FLAG1:(ON → OFF)
3	Rev_Rot	U_ROT 命令復元, PC++
4	Opr_Var1	*2, PC:=FLAG_1
5	Flag_1	PC:=Rev_Opr1, FLAG1:(OFF → ON)
6	Rev_Opr1	U_OPR 命令復元, PC++
7	Opr_Var2	*3, PC:=Rev_Opr2
8	Rev_Opr2	U_OPR 命令復元, PC++
9	End_Prog	終了

以下に低級アセンブリ言語のプログラム例を示す．ここで，"#" から改行までをコメントとする．また，Entry ラベルは PC の初期値であるエントリーポイントを表しており，FLAG1 の初期値は ON である．

```
Rev_Rot        :  R_ROT
Opr_Var1       :  U_OPR  Var1
Entry          :  U_ROT  Var1       # エントリーポイント
Var1           :  30537             # = 1112220000t
Flag_1         :  FLAG1
Jmp_to_Rev_Rot :  R_ROT
Rev_Opr1       :  R_OPR
Opr_Var2       :  U_OPR  Var2
Var2           :  11355             # = 0120120120t
Rev_Opr2       :  R_OPR
End_Prog       :  END
```

この低級アセンブリプログラムは，次の疑似命令列を実現する．

```
ROT  Var1
OPR  Var1
OPR  Var2
```

このプログラムは表 3 のように実行される．ここで，*1,2,3 はそれぞれ以下を表している．

Var1,A:=rotr(1112220000t) (=0111222000t),

Var1,A:=op(0111222000t,0111222000t) (=1000111111t),

Var2,A:=op(1000111111t,0120120120t) (=0121121121t)

上記のプログラム例で用いられなかったデータに関して以下でその意味を説明する．

- 命令：U_INPUT
  A:=getchr(), PC:=PC+2 の動作が行われる．
- 命令：U_OUTPUT
  putchr(A), PC:=PC+2 の動作が行われる．
- 命令：U_MOV_PC Label
  PC:=Label+1 に設定．

- 命令：U_JMP Label
  PC:=Label+1 に設定．ただし，PC<Label≦k (k:ある定数)．
- フラグ：FLAG i +1
  i 番目のフラグを反転し，PC:=PC+1 が行われる．

3.2 高級アセンブリ言語

高級アセンブリ言語 [2] は，4 章で説明する基本モジュール方式のメインルーチンをより一般的に表現することを目的として設計された言語である．このため，値の取る範囲は 10trits(0〜59048) である．高級アセンブリプログラムは，アドレスを表すラベルを取り付け可能な命令の列によって定義される．各命令は 1 行で定義され，引数には変数やラベルが用いられる．便宜上，以下で定義する命令はチューリング完全性証明に必要な 6 種類に限定する．

- インクリメント命令 "INC X"：X:=X+1 を計算する．ただし，X の値が 59048 のときの演算結果は 0 とする．
- デクリメント命令 "DEC X"：X:=X−1 を計算する．ただし，X の値が 0 のときの演算結果は 59048 とする．
- 変数コピー命令 "MOV X,Y"：Y:=X の代入命令を行う．
- 条件分岐命令 "BRANCH X,Y"：X が 0 の場合，PC が Y でラベル付けされた命令のアドレスに設定される．X が 0 以外の場合，何も行わない．
- 入力命令 "INPUT X"：X:=getchar() を計算する命令．ただし，EOF が入力されたときは X に 59048 が代入される．
- 出力命令 "OUTPUT X"：putchar(X) を計算する命令．
- 停止命令 "STOP"：実行を停止する命令．

以下にプログラム例を示す．これは，標準入力により入力された文字を標準出力するプログラムであり，EOF(=59048) が入力されるまで繰り返し実行される．

```
Loop:  INPUT Var_X
       MOV Var_X,Var_Y
       INC Var_Y
       BRANCH Var_Y,Exit
       OUTPUT Var_X
       BRANCH Zero,Loop
Exit:  STOP
```

ここで，プログラム中の Zero は定数の 0 を表している．

4. 高級アセンブリ言語の実現

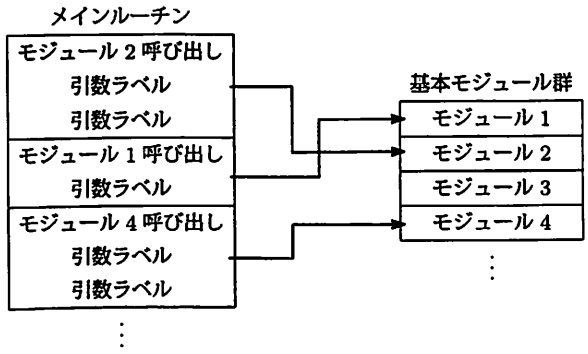


図 1 基本モジュール方式

表 4 モジュール作成に必要な機能一覧
Table 4 Function List for Implementation of Modules

機能番号	機能説明
(1)	引数ラベルの変数値をレジスタに設定
(2)	レジスタ値を引数ラベルの変数に設定
(3)	IP のインクリメント
(4)	レジスタ値のインクリメント
(5)	レジスタ値のデクリメント
(6)	レジスタ値の 0 判定
(7)	IP を引数ラベルのアドレス値に設定
(8)	入力値をレジスタ値に設定
(9)	レジスタ値を出力
(10)	停止命令の実行

表 5 モジュールと機能との関係
Table 5 Relation between Modules and Functions

モジュール	必要な機能
インクリメントモジュール	(1),(2),(3),(4)
デクリメントモジュール	(1),(2),(3),(5)
変数コピーモジュール	(1),(2),(3)
条件分岐モジュール	(1),(3),(6),(7)
入力モジュール	(2),(3),(8)
出力モジュール	(1),(3),(9)
停止モジュール	(10)

高級アセンブリ言語を低級アセンブリ言語の基本モジュール方式 [3] を利用して実現する方法を示す．

基本モジュール方式は，低級アセンブリ言語で大規模なループプログラムを作成するために考案されたプログラミング手法であり，いくつかの基本関数を実現する基本モジュール群と，それらをサブルーチンコールのように呼び出すメインルーチンで構成される (図 1)．

実現には，高級アセンブリ言語の各命令と同じ機能を持つ基本モジュールを作成すればよい．各モジュールに対して必要な機能を表 4,5 にまとめた．ここで，表 4 の IP はインストラクションポインタの略記である．また，高級アセンブリ言語のレジスタや IP は，低級アセンブリ言語では特定の変数を指しているため，低級アセンブリ言語のレジスタ (A,PC) とは異なることに注意されたい．

文献 [2] において，機能 (1)〜(4),(7) については実現方法が具体的に記述されており，(10) については容易に実現できるため，それ以外の機能を実現する擬似命令列を具体的に設計する．これらは，文献 [2] により，低級アセンブリプログラムに変換できる．具体的な低級アセンブリプログラムについては [6] の URL を参照されたい．

4.1 デクリメント機能

デクリメントを低級アセンブリ言語で実現するには，対象となる値に対して表 6 の trit 演算と右ローテートを桁下げが発生しなくなるまで (最大 10 回) 行う．その後は，右ローテートのみを合計で 10 回になるまで行えばよい．

以下は，変数 R に対して表 6 の trit 演算と，桁下げ発生の判定を同時に行う擬似命令列である．ここで，変数 X,Y の初期値は，0(= 0000000000t)，59047(= 2222222221t) とする．ま

表 6　デクリメント用 trit 演算
Table 6　Trit Operation for Decrement

Y_i \ X_i	0	1	2
0	2	0	1
1	2	0	1
2	0	1	2

た，CF は桁下げ発生判定用の変数であり，その初期値の三進数表現は最下位の値が 0，または，1 とする．

```
ROT  Con1        OPR  Y         ROT  Con2
OPR  X           OPR  R         OPR  Y
ROT  Con2        ROT  Con2      ROT  Con2
OPR  R           OPR  R         OPR  Y
ROT  Con2        ROT  Con0      OPR  CF
OPR  R           OPR  Pat21
OPR  X           OPR  CF
```

この命令列の実行後，変数 R の最下位値がデクリメントされる．また，CF の値は，R の三進数表現の最下位値が 1，または，2 である場合，実行前の値の最下位を 1 にした値となり，R の三進数表現の最下位値が 0 である場合は実行前の最下位値を 0 にした値となる．

桁下げ発生に関する実行制御方法として，低級アセンブリ言語の U_MOD_PC 命令を CF に対して行えば，次に実行する命令が桁下げ判定結果によって変わるため実現できる．

4.2　0 判定分岐

デクリメント機能を利用することで実現できる．判定したい変数に対してデクリメントを行うと，変数値が 0 である場合に限り，CF の三進数表現の最下位値が 0 となるため，この CF を引数として低級アセンブリ言語の U_MOV_PC 命令を行うことで実現できる．

4.3　入力機能

入力された値を変数 R に設定するには，変数コピー用の疑似命令列 [2] と同様に実現できる．

以下の疑似命令列は，入力 R:=getchar() を実現する．

```
ROT  Con1        OPR  X
OPR  R           INPUT
OPR  R           OPR  X
OPR  X           OPR  R
```

この疑似命令列では，初めの 5 行で X と R の値を 29524(= 1111111111t) に設定する．6 行目で INPUT 命令を実行し A レジスタに入力値が設定された状態で最後の 2 行を実行すると，R が入力値となる．

4.3.1　出力機能

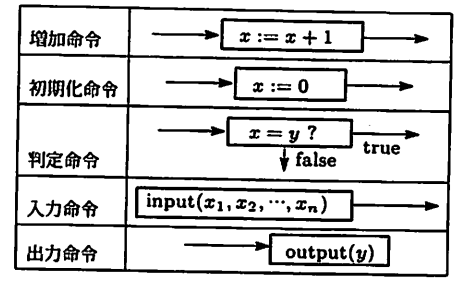
変数 R の値出力は，以下の疑似命令を利用すればよい．

```
ROT     Con2
OPR     R
ROT     Con2
OPR     R
OUTPUT
```

5.　Malbolge のチューリング完全性

正規形の N プログラム (以下では **NNP** と書く) はチューリング完全であり，表 7 形式の命令をノードに持つ有限の有向グラフとして定義される [4]．ここで，$x, y, x_1, x_2, \cdots$ は変数であり，任意の自然数を保持できる．

表 7　NNP の命令
Table 7　NNP Instruction



以下では，変数の値，および，変数と命令の個数にある上限を持たせた NNP が高級アセンブリ言語で実現できることを示す．このような条件を持たせた NNP を**弱チューリング完全**と定義する．低級アセンブリプログラムは Malbolge に変換できる [2] ため，本章と 4 章から Malbolge の弱チューリング完全性が示せる．

対象の NNP 中の命令数を n 個とする．まず，これらの命令を適当に 1 列に並べて順番にラベル (Label_1〜Label_n) を割り当てる．次に，NNP の実行制御順序が適切になるように無条件分岐を置く．これは，高級アセンブリ言語の条件分岐命令の第一引数に Zero(= 0)，第二引数に適当なラベル：Label_m ($1 \leqq m \leqq n$) を用いる．また，5.1 節から 5.4 節で示すように NNP の各命令を高級アセンブリ言語で実現する．これらにより，NNP は高級アセンブリ言語に変換できる．

5.1　判定命令の実現

NNP の判定命令の実現は 2 つの変数 X,Y の等しさを判定する必要がある．そこで，2 変数の値を少なくともどちらか一方が 0 になるまでデクリメントを行うことにより，高級アセンブリ言語の条件分岐命令のみで判定命令を実現できる．

```
Copy_X          :  MOV  X, X_Temp
Copy_Y          :  MOV  Y, Y_Temp
X_Check         :  BRANCH  X_Temp, Y_Check_2
Y_Check_1       :  BRANCH  Y_Temp, Goto_False
Jmp_Dec         :  BRANCH  Zero, Dec_X_Temp
Y_Check_2       :  BRANCH  Y_Temp, Goto_True
Jmp_Goto_False  :  BRANCH  Zero, Goto_False
Dec_X_Temp      :  DEC  X_Temp
Dec_Y_Temp      :  DEC  Y_Temp
Jmp_Check       :  BRANCH  Zero, X_Check
```

ここで，Goto_True(Goto_False) は判定結果が true(false) の場合に実行すべき NNP の命令に割り当てるラベルを表している．

5.2　入力命令の実現

NNP の入力命令は n 個の任意の自然数が入力対象となる．これに対して，高級アセンブリ言語の入力命令は 1 文字入力 (getchar) であるため，実現には各引数毎に以下の様々なモジュールの作成が必要となる．

モジュール 1：特定の文字が入力されるまで入力を繰り返す．

モジュール 2：入力された文字コードを数に変換する．
モジュール 3：任意の n 桁の自然数と 1 桁の自然数 (0〜9) を使って $n+1$ 桁の数へと統合する (例: 23 と 5 を 235 に)

ここで，引数が n 個の場合には上記のモジュール 1〜3 を組み合わせて作成したモジュールを n 個並べればよい．

- モジュール 1 について
  入力値の桁数は実行時までわからないので，特定の文字が入力されるまで入力命令を繰り返す必要がある．
  特定の文字として改行文字 ('\n') を用いる．この実現には改行文字の ASCII コード値が 10 であることを利用する．具体的には，デクリメント命令と判定命令により改行文字かどうかを判定し，改行文字の場合は次の引数を入力するモジュールへと分岐すればよい．
  以下に，変数 X の改行文字判定を行うモジュール 1 を示す．

```
Copy_X       :  MOV  X, X'
Dec_X'-1     :  DEC  X'
    ⋮           (上の 2 行目を 10 回繰り返す)
Dec_X'-10    :  DEC  X'
Check_10     :  BRANCH  X', Goto_Next_Arg
    ⋮           (入力数字 X に対する処理)
```

- モジュール 2 について
  '0' の ASCII コード値が 48 であることを利用すれば，デクリメント命令 (DEC) により容易に実現できる．

```
Dec_X-1      :  DEC  X
    ⋮           (上の一行目を 48 回繰り返す)
Dec_X-48     :  DEC  X
```

- モジュール 3 について
  2 つの値を加算するサブモジュールと数値を 10 倍するサブモジュールを作成することで実現できる．
  Z:=X+Y の加算サブモジュールを以下に示す．Y の値が 0 までデクリメントする間，X をインクリメントすることで実現している．

```
Copy_X       :  MOV  X, X_Temp
Copy_Y       :  MOV  Y, Y_Temp
Y_Check      :  BRANCH  Y_Temp, Post
Inc_X_Temp   :  INC  X_Temp
Dec_Y_Temp   :  DEC  Y_Temp
Jmp_Check    :  BRANCH  Zero, Y_Check
Post         :  MOV  X_Temp, Z
```

Y:=X×10 の 10 倍サブモジュールを以下に示す．この実現に上記の加算サブモジュールを用いた．具体的には，初期値が 10 である変数 Count_10 を利用して，X を 10 回加算している．

```
Init_Count_10     :  MOV  Zero, Count_10
Set_Count_10-1    :  INC  Count_10
    ⋮                (上の 2 行目を 10 回繰り返す)
Set_Count_10-10   :  INC  Count_10
Init_Temp         :  MOV  Zero, Temp
10_Check          :  BRANCH  Count_10, Post
| Temp := Temp + X の加算サブモジュール |
Dec_Count_10      :  DEC  Count_10
Jmp_10_Check      :  BRANCH  Zero, 10_Check
Post              :  MOV  Temp, Y
```

以上の二つのサブモジュールを用いることにより，モジュール 3 が実現できる．

モジュール 1,2,3 を利用した入力モジュールの一部 (m 引数のみ) を以下に示す．ここで，m 引数目の入力値は変数 X_m に格納され，n 引数目で改行文字が入力された場合は次の NNP の命令に割り当てたラベルを記述する．

```
Pre_X_m          :  MOV Zero, X_m

Input_X_m_Temp   :  INPUT X_m_Temp
| モジュール 1(X_m_Temp が'\n' の場合は Pre_X_m + 1 に分岐) |
| モジュール 2(X_m_Temp を ASCII 値から数値に変換)          |
| モジュール 3(X_m と X_m_Temp を統合)                     |
Jmp_Input        :  BRANCH Zero, Input_X_m_Temp
```

5.3 出力命令の実現

NNP の出力命令を実現するためには，以下のモジュールの作成が必要となる．
モジュール a：出力値の桁数を求める (例: 6789 → 4 桁)
モジュール b：値を最上位桁の数とそれ以外の値に分ける (例: 789 → 7 と 89)
モジュール c：出力する数を ASCII 文字に変換する

- モジュール a
  $X \times 10^Y$ を計算するサブモジュール，および，引き算サブモジュールを作成することで実現できる．
  以下に $Z:=X \times 10^Y$ サブモジュールを示す．実現のために，5.2 節の 10 倍サブモジュールを用いている．

```
Init_Temp     :  MOV  X, Temp
Check_Y       :  BRANCH  Y, Post
| Temp := Temp×10 の 10 倍サブモジュール |
Dec_Y         :  DEC  Y
Jmp_Check_Y   :  BRANCH  Zero, Check_Y
Post          :  MOV  Temp, Z
```

以下の S1:=T−U , S2:=U−T を行う引き算サブモジュールを示す．これは，デクリメント命令と条件分岐命令により容易に実現できる．ただし，T<U の場合は S1=0，U<T の場合は S2=0 とする．

```
Copy_T      :  MOV  T, Temp_T
Copy_U      :  MOV  U, Temp_U
Check_T     :  BRANCH  Temp_T, Post1
Check_U     :  BRANCH  Temp_U, Post1
Dec_Temp_T  :  DEC  Temp_T
Dec_Temp_U  :  DEC  Temp_U
Jmp_Check   :  BRANCH  Zero, Check_T
Post1       :  MOV  Temp_T, S1
Post2       :  MOV  Temp_U, S2
```

上記の二つのサブモジュールを用いて，桁数 DN(Digits Number) を計算するモジュール a を以下に示す．具体的には，桁数を求めたい値 (Num≧ 0) に対して，Num≦ 1×10^{Y} となる最小の Y を求めることで実現している．

```
Init_X      :  MOV  Zero, X
Init_Y      :  MOV  Zero, Y
Inc_X       :  INC  X
Inc_Y_1     :  INC  Y
| Z := X ×10^Y を計算するサブモジュール |
| S1:=Num−Z, S2:=Z−Num の引き算サブモジュール |
Check_S1    :  BRANCH  S1, Check_S2
Jmp_Inc_Y   :  BRANCH  Zero, Inc_Y_1
Check_S2    :  BRANCH  S2, Inc_Y_2
Dec_Y       :  DEC  Y
Inc_Y_2     :  INC  Y
Post        :  MOV  Y, DN
```

- モジュール b

以下に示したモジュール b は，モジュール a で作成した二つのサブモジュールを用いている．ここで，分割したい値 (Num) の最上位の値を Num1 とし，最上位を除いた値を Num2 とする．また，Num の桁数 (DN) はあらかじめわかっているものとし，DN=1 の場合は Num2=0 とする．具体的には，Num≦ X $\times 10^{DN-1}$ となる最小の X を求めることで Num1 を求め，Num−Num1$\times 10^{DN-1}$ により Num2 を求めている．

```
Init_X      :  MOV  Zero, X
Init_Y      :  MOV  DN, Y
Dec_Y       :  DEC  Y
Inc_X       :  INC  X
| Z := X ×10^Y を計算するサブモジュール |
| S1:=Num−Z, S2:=Z−Num の引き算サブモジュール |
Check_S1    :  BRANCH  S1, Check_S2
Jmp_Inc_X   :  BRANCH  Zero, Inc_X
Check_S2    :  BRANCH  S2, Post1
Dec_X       :  DEC  X
Post1       :  MOV  X, Num1
| Z := X ×10^Y を計算するサブモジュール |
Post2       :  MOV  Z, Num2
```

- モジュール c

変換したい変数に対して 48 回インクリメントすればよい．

```
Inc_X+1     :  INC  X
    ⋮    (上の 1 行目を 48 回繰り返す)
Inc_X+48    :  INC  X
```

モジュール a,b,c を用いて実現した出力モジュールを以下に示す．出力したい値 (Num) が 100 など 0 を含む値に対しても正しく出力するように，始めに Num の桁数を求め，デクリメント命令を用いて桁数回だけループする構造となっている．また，NNP は出力命令で終了するため，モジュールの最後で停止命令を記述している．

```
| モジュール a(Num の桁数 (DN) を求める) |
| モジュール b(Num を Num1 と Num2 に分解) |
| モジュール c(Num1 の数値を ASCII 値に変換) |
Output_Num1 :  Output  Num1
Copy_Num2   :  MOV  Num2, Num
Dec_DN      :  DEC  DN
Check_DN    :  BRANCH  DN, (モジュール b の先頭命令)
End_Exe     :  STOP
```

5.4 その他の命令の実現

増加命令，初期化命令は，高級アセンブリ言語のインクリメント命令と変数コピー命令で容易に実現できる．

6. ま と め

チューリング完全性を持つ正規形の N プログラムを利用して，Malbolge が弱チューリング完全性を持つことを示した．

Malbolge の値は 10trits で表現されており，資源が有限であることは一般的な現実機械と同じ状況であるが，高級アセンブリ言語には配列や間接参照の機能がなく，プログラミングには不向きである．そこで，Malbolge の実用化のためには，これらの機能を拡張することが望ましい．

Malbolge プログラムをソフトウェア保護の目的に利用する場合には，本稿で述べたようなプログラムの作成法で作られたことが分かってしまうと，難解性を低下する恐れがある．そこで，変換時にコードの最適化を施したり，乱数を用いた構造変換などを行うことでより難解性を高めることが望ましい．

謝辞 本研究は一部，科研費 #22650003 の助成を受けている．

文 献

[1] Ben Olmstead：“Malbolge: Programming from Hell”, http://www.bouletfermat.com/danny/malbolge/, 1998.
[2] 飯澤恒：難解言語 Malbolge に基づくプログラム難読化に関する研究，名古屋大学修士論文，2006.
[3] 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，信学技報，電子情報通信学会，Vol.105，No. 129，pp. 25-30，2005.
[4] 高橋正子：計算論—計算可能性とラムダ計算—，近代出版社，1991.
[5] “最凶言語 Malbolge - ロベールの小部屋”, http://d.hatena.ne.jp/Robe/20060824，2006.
[6] http://www.sakabe.i.is.nagoya-u.ac.jp/malbolge/，2010.