

新しい言語の作り方

2026/05/06(水)

川合 秀実

アセンブラ自作

- アセンブラとは、機械語の命令と対応している非常に低級な言語（アセンブリ言語）のコンパイラのことです。
- Cコンパイラは、C言語のソースコードをアセンブリ言語に変換していて、それをアセンブラに渡すことで目的を達成しています。だからアセンブラは知らないうちに使っているツールです。
- アセンブラを自作したいという人はほとんどいません。自分が直接使わないものを作る動機は特にはないですね。
- でもあえて、これを自作したいとして、そのために簡単な方法を探ってみようと思います。

x86のアセンブリ言語の例

- 右に典型的なアセンブリ言語の例を示します。
- これを対応するバイナリデータに変換するのがアセンブラの仕事です。
- アセンブラに限りませんが、一般にプログラミング言語を作るのなら、**字句解析**をして、**構文解析**をして、**意味解析**をして、**最適化**をして、**結果を出力**します。
- でも、そんなの**面倒**だと私は思うのです。

```
MOV    EAX, 123
PUSH   ECX
MOV    [EBP+48], EDX
RET
など
```

アセンブラの構文を変えれば . . .

- アセンブリ言語の文法を少し変えてみます。
 - 右のように → → → → →
- こうすれば、右下のようなヘッダファイルを使うことで、機械語に変換できます。
 - DBというのはバイナリデータを直接出力する命令です。

```
MOV (EAX, 123);  
PUSH (ECX);  
MOV ([EBP+48], EDX);  
RET ();  
など
```

```
#define EAX 0  
#define ECX 1  
#define EDX 2  
#define RET () DB (0xC3)  
#define PUSH (r) DB (0x50+r)  
#define POP (r) DB (0x58+r)  
など
```

このやり方での問題点と解決案

- たとえば PUSH 命令は、レジスタ番号を指定する場合と、整数定数を指定する場合と、メモリ上の整数値を指定する場合があります。
- これを全部 PUSH と書かれてしまうと、うまく区別できません。
 - (これらの機械語はそれぞれ違います。)
- → **解決案** : PUSH_Reg(), PUSH_Const(), PUSH_Mem() と書き分けます。

- アセンブリ言語は単純な置換の他にもラベルによるアドレス計算という大事な役割があり、それはヘッダファイルだけでは解決しません。
- → **解決案** : 仕方ないのでアドレス計算機能だけは実装します。他の機能はなくしてヘッダファイルによる置換に任せます。

やってみました

- プリプロセッサは以前自作していたので、それにアドレス計算機能を付けて、さらに事前に決めておいたヘッダファイルを勝手にインクルードするようにしてみました。
 - (プリプロセッサ = #defineなどを解釈するプログラム)
- (a)自作プリプロセッサのみ → **23.5KB**
- (b)上記にアドレス計算処理を足して、アセンブラ化したもの → **26.0KB**
 - 実行ファイル (.exeファイル) のサイズで比較しました。
 - つまりアセンブラを作るために要した記述は **2.5KB** 分のみ！ (1日でできる)
- ヘッダファイルを書き足すだけで、すぐに新命令に対応できます。
- ヘッダファイルを書き換えるだけで、全然違うCPUにも対応できます。

C言語もこんな感じで作れたらいいのに

- 「a=b+c;」ではマクロ置換ができません。
- だから「ADD(a, b,c);」にする必要があります。
- しかし「a=b+c;」と書けないなら、それはもうC言語ではなく別の言語です。
- →**解決案**：仕方ないので「a=b+c;」とかをADD()とかに変換する関数だけ作ります。ifやforなどもうまいこと変換します。
 - ・ ・ ・ ADD()形式になってしまえば、あとはヘッダファイルとプリプロセッサだけで、どんどん置換していけます。

やってみました (パート2)

- (a)自作プリプロセッサのみ → **23.5KB**
- (b)上記に構文形式の変換関数を追加して、簡易Cコンパイラとして機能するようにしてみました → **28.0KB**
 - つまり簡易Cコンパイラを作るために要した記述は **4.5KB** 分のみ！
 - これはいいぞー
- 簡易ではなくもっとちゃんとサポートしたら、その差は10.0KBとかになる可能性はあります。でもきちんとC言語を作るよりはずっと楽ではあるでしょう、たぶん。

もっとやってみました

- この自作簡易Cコンパイラの出力を、先の自作アセンブラに渡して、それをそのまま実行してみました。つまりC言語のJITコンパイラです。→ **30.0KB**
 - あれ？JITコンパイラってこんなに簡単にできていいのかな？？（笑）
- さらにグラフィック関係の処理もできるようにして、簡単なプログラムをコンパイルせずにスクリプト言語みたいに実行して遊んでいます。
- **（誤解されやすいポイント）**
- **30KB**の自作ツールの他にgccとかclangとかが必要？ → 一切**不要**です！
- 特別なDLLファイルとかが必要なんじゃないの？ → 一切**不要**です！
- ヘッダファイルが巨大？ → 100行未満です。

30KBのJITコンパイラでもこれくらいは動く

- 素数を計算するプログラム
 - (今は配列変数が使えないので、あまり賢くはないアルゴリズムです)

```
int i, j;
for (i = 2; i < 100; i++) {
    for (j = 2; j * j <= i; j++) {
        if (i % j == 0) goto skip;
    }
    printf("%d ", i);
skip: ;
}
printf("¥n");
```

- 実行結果：

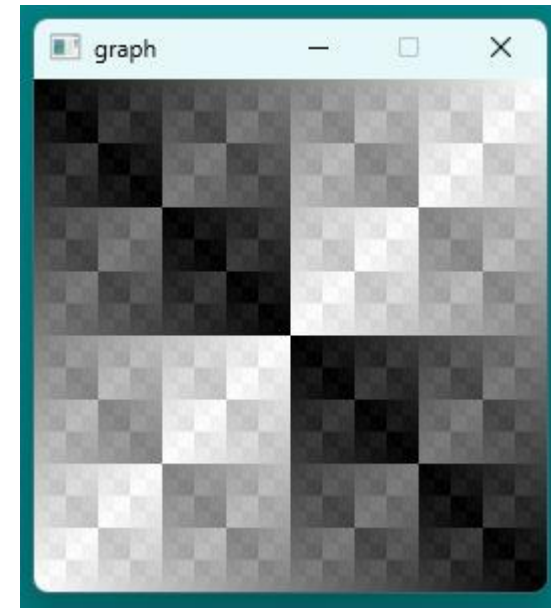
```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

32KBのJITコンパイラでもこれくらいは動く

```
int x, y, c; c = 0;
openWin(256, 256);
for (y = 0; y < 256; y++) {
    for (x = 0; x < 256; x++) {
        setPix(x, y, c);
        c = c + 0x100;
    }
}
waitInf();
```



```
int x, y;
openWin(256, 256);
for (y = 0; y < 256; y++)
    for (x = 0; x < 256; x++)
        setPix(x, y, (x ^ y) * 0x10101);
waitInf();
```



まとめ

- プリプロセッサの #define を使えば、難しい命令を簡単な命令の組み合わせに展開していくことができます。
- これによりアセンブラやCコンパイラを簡単に作れそうです。
- C言語では、言語の基本構文がプリプロセッサマクロの構文とかけ離れているので、最初にそこを変換する関数は必要そうです。
- 今後は、配列やポインタや関数定義などの機能を増やしていったって、それでも「簡単に作れる」が維持できるのか確かめたいです。
- 今回は説明を省略していますが、変換処理をより柔軟にするために、自作プリプロセッサには型付きdefine という機能もつけてあります。