第3章

OSやコンパイラの違いを吸収する仕組みから吸収できない環境の違いまで

C言語が環境の違いを抽象化するしくみ

内田 公太

リスト1　入力されるソースコードが同じでも対象となるOSが異なるとコンパイラの出力が異なる

```
#include <stdio.h>

int add(int a, int b) {
  a += b;
  return a;
}

int main() {
  printf("%d¥n", add(1, 2));
}
```

（a）コンパイラに入力するC言語ソースコード

```
add:
    lea     eax, [rdi + rsi]
    ret
main:
    push    rax
    mov     edi, 1
    mov     esi, 2
    call    add
    lea     rdi, [rip + .L.str]
    mov     esi, eax
    xor     eax, eax
    call    printf@PLT
    xor     eax, eax
    pop     rcx
    ret
```

（b）Linux環境のClangが出力したアセンブリ言語コード

```
add:
    lea     eax, [rcx + rdx]
    ret
main:
    sub     rsp, 0x28
    mov     edx, 2
    mov     ecx, 1
    call    add
    mov     edx, eax
    lea     rcx, [rip + disp32]
    call    printf
    xor     eax, eax
    add     rsp, 0x28
    ret
```

（c）Windos環境のVisual C++が出力したアセンブリ言語コード

● CPUが同じでも機械語が異なる場合がある

同じC言語ソースコードで，CPUも同じだったとしても，同じ機械語になるとは限りません．さまざまな要素がコンパイル結果に影響を与えます．その1つにOSがあります．

OSが変わると，標準的に採用される呼び出し規約が変わります．呼び出し規約とは，関数を呼び出すときの機械語レベルでの決め事です．関数の引数の設定方法や，戻り値の受け取り方，関数を呼び出した側と呼ばれた側の責務などを規定します．

● ClangとVisual C++の結果を比較してみる

リスト1を例として説明します．**リスト1**（**a**）のプログラムをLinux環境のClang［**リスト1**（**b**）］とWindows環境のVisual C++［**リスト1**（**c**）］でそれぞれコンパイルした結果を比べてみます．Clangの出力と比較しやすくするために，Visual C++の出力においてアセンブリ言語命令の表記を変えています．

どちらもCPUはx86-64で同じですが，CPU以外の環境（OSやコンパイラ，コンパイル時のオプションなど）が異なるので，得られる機械語が変わります．

▶OSの影響…呼び出し規約の違いによる

LinuxやmacOSなどのUnix系OSでは，System V AMD64 ABI[1]（正確にはSystem V Application Binary InterfaceのAMD64 Architecture Processor Supplement）が標準となっているため，その中で定められている呼び出し規約が使用されます．Windowsでは，マイクロソフトのx64呼び出し規約[2]が標準的に採用されます．

両者は関数を呼び出すときのレジスタの使い方などが異なるので，同じCPUであっても異なる機械語列になります．

▶コンパイラの影響…具体的な処理の方法が変わる

Visual C++とClangでは，引数をレジスタに設定する順序や，スタック調整のための命令選択といった部分に違いが見られます．加えて，Clangの方は`printf`を呼ぶ前にRAXをクリアしていますが，Visual C++ではしていないようです．

このように，C言語のプログラムが意図通りに動く範囲であれば，具体的な処理の方法は言語処理系が自由に決められます．これがC言語をはじめとする高水準言語の大きなメリットです．同じソースコードが処理方法を変えながらさまざまな環境で動きます．