第 3 章　プログラムが動く仕組みを知ろう

スタックと割り込み

大山 将城

ここでは，C言語で開発したプログラムがシステム上で動く際に必要な，「スタック」と「割り込み」と呼ばれる仕組みについて説明する．「スタック」は関数の作業領域としてメモリの一部を利用する仕組みである．「割り込み」は実行の流れを切り替えるための仕組みである．どちらもC言語の文法ではないが，プログラムが動く仕組みを知っておくと，特にデバッグで役に立つ．（筆者）

本稿では，スタックと割り込みの概念や動作原理について説明します．スタックと割り込みは，Cプログラムの文法上は見えないにもかかわらず，プログラム実行の上で重要な役割を果たしています．

1．プログラムの実行の流れを知ろう

ここでは，順次実行，分岐，関数呼び出しという3種類のC言語プログラムを例に，プログラムの実行の流れを見ていきます．まずは，順次実行と分岐について，C言語ソース・レベル，およびメモリ上の動作レベルで見ていきましょう．

● 順次実行

リスト1にmain関数のみで作成したプログラムを示します．内容は，処理Aと処理Bで構成されています．このプログラムを実行したとき，実行の流れは，処理A→処理Bです．

● 分岐

リスト2にif文を含むプログラムを示します．ここでは，引き数chの値によってローカル変数resultに代入する値を変え，最後はresultを戻り値として関数を終了します．このプログラムを実行したとき，実行の流れは以下のようになります．

- if文の条件「ch=='A'」が真なら，“result = 1;”→“return result;”
- if文の条件「ch=='A'」が偽なら，“result = 2;”→“return result;”

for文，while文，do～while文は，繰り返しが加わる以外はif文と同じです．switch文も，if～elseの組み合わせと考えれば，if文と同じです．

● 実行の鍵を握るプログラム・カウンタ

以上が，C言語のソース・レベルでのプログラム実行の流れです．では，CPUはこれをどのように実行するのでしょうか．

C言語で記述したプログラムは，コンパイラによって

▶ リスト2
if文を含むプログラム

```
int FuncA( char ch )
{
    int result;

    if(ch == 'A'){
        result = 1;
    }
    else{
        result = 2;
    }

    return result;
}
```

リスト1　main関数のみのプログラム

```
void    main()
{
    /* 処理A */
    /* 処理B */
}
```

リスト3　リスト2をコンパイルした結果の機械語プログラム

```
Address  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0x0000 :                           40 0E 00 00 21 2E 00 00
0x0010 : 40 0E 00 01 21 26 0C 61 C5 21 40 0E 00 01 21 1E
0x0020 : 0C E3 40 0E 00 01 21 F6 00 E0 20 A6 FF 00 40 0E
0x0030 : 01 00 21 AE FF FF 40 0E 00 01 21 6E 0C E1 40 0E
0x0040 : 00 01 21 66 0C E1 EC 69 E9 05 6D 07 01 00 44 6A
0x0050 : EC 69 C1 FD 40 0E 00 01 21 6E 0C E1 40 0E 00 01
0x0060 : 21 66 0C E3 EC 69 E9 05 6D 07 01 00 44 6A EC 69
0x0070 : C1 FD 44 0E FF FF 21 37 F5 7F 44 0E FF FF 21 3E
0x0080 : F8 7F 80 FF 4A 00 E0 07 20 01 00 00 C5 1D 63 37
0x0090 : 09 00 03 57 08 00 20 0E 41 00 E1 51 DA 05 01 5A
0x00a0 : 63 5F 01 00 C5 05 02 62 63 67 01 00 23 6F 01 00
0x00b0 : 63 6F 05 00 95 05 23 57 05 00 23 FF 0D 00 03 1E
0x00c0 : 10 00 7F 00 50 1A 63 FF 0D 00 A5 E5 95 0D 20 36
0x00d0 : 41 00 BF FF BA FF 23 FF 01 00 44 1A 7F 00 5C 1A
0x00e0 : 63 FF 01 00 D5 F5
```


KEYWORD ── プログラム・カウンタ，スタック・ポインタ，スタック，レジスタ，関数呼び出し，関数リターン，プッシュ，ポップ，LIFO，関数フレーム，割り込み，割り込みハンドラ

リスト4　リスト3と等価のアセンブリ言語（V850の例）

```
_FuncA:
    add     -.S2, sp
    st.w    lp, -4+.F2[sp]
    st.w    r6, .R2[sp]
    ld.b    .R2[sp], r10
    cmp     65, r10
    jne     .L12
    mov     1, r11
    st.w    r11, -8+.A2[sp]
    jbr     .L17
.L12:
    mov     2, r12
    st.w    r12, -8+.A2[sp]
.L17:
    ld.w    -8+.A2[sp], r13
    st.w    r13, -4+.A2[sp]
    ld.w    -4+.A2[sp], r10
    ld.w    -4+.F2[sp], lp
    add     .S2, sp
    jmp     [lp]        --1
```

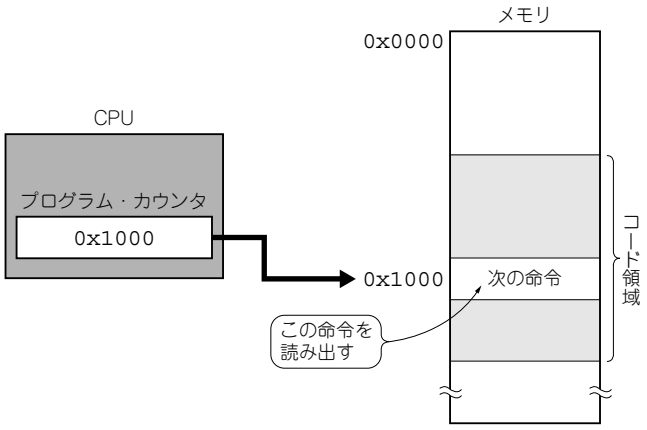

図1　プログラム・カウンタとメモリの関係

プログラム・カウンタの値が0x1000のとき，CPUはメモリ上に確保されたコード領域内のアドレス0x1000から命令を読み出す．命令を読み出したら，次の命令を読み出すためにプログラム・カウンタの値を増やす．

CPUが理解できる機械語に翻訳されます（**リスト3**）．機械語をもう少し人間に分かりやすく表現したものがアセンブリ言語です（**リスト4**）．

CPUは機械語の命令を一つずつメモリから読み出して実行します．さて，このときメモリ上のどの位置（どのアドレス）にある命令を読み出せばよいのかをどうやって知るのでしょうか？ ここで出てくるのが，「プログラム・カウンタ」という特別なレジスタ[注1]です．

プログラム・カウンタは，命令が格納されている領域（一般に「コード領域」と呼ばれる）内の，一つのアドレスを指しています．CPUはプログラム・カウンタが指定するアドレスのメモリから命令を読み出し，次の命令を読み出すためにプログラム・カウンタの値を増やします（**図1**）．

命令を読み出すたびにプログラム・カウンタの値を増やすことで，CPUは命令の順次実行を実現します．分岐の場合は，分岐命令の実行結果によってプログラム・カウンタを更新する値が変わります．

図2に，C言語プログラム（**リスト2**）とアセンブリ言語プログラム（**リスト4**）の対応関係を示します．このアセン

注1：レジスタとは，CPUにとって特に重要な情報を格納するための記憶領域であり，CPU内部に存在する．

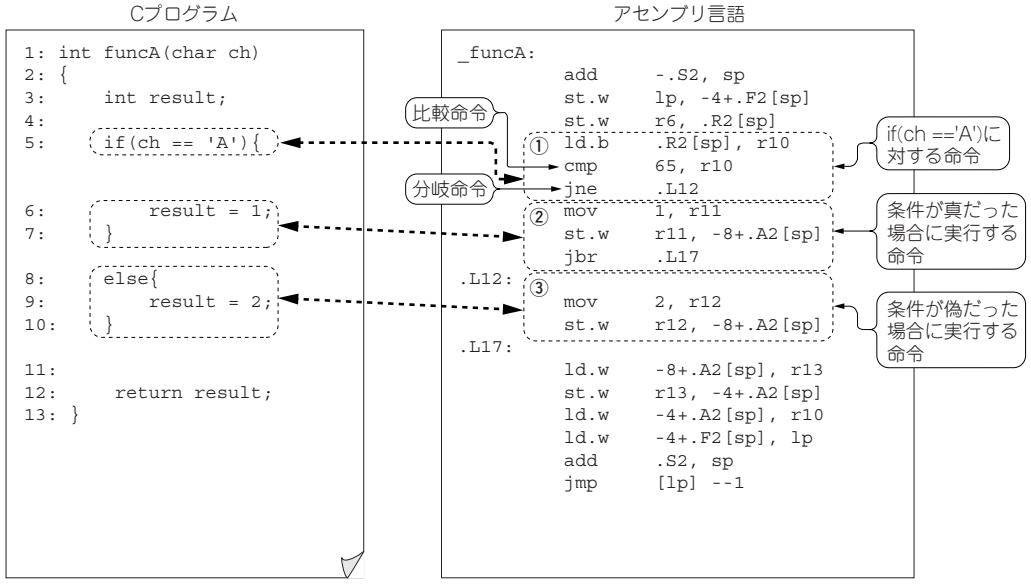

図2
C言語ソース・プログラム（リスト2）**とアセンブリ言語プログラム**（リスト4）**の対応関係**

比較命令cmpで65（'A'）とr10（chの中身）を比較し，分岐命令jneで前行の比較結果がイコールでない場合に.L12に分岐している．