

3章 一生世界に挨拶してそう

タイトルから喧嘩を売っているが、本章では次のことを学ぶ。

- 文字の入力、出力（合わせて 入出力 と呼ぶ）を使いこなす
- 数値の入力と、その扱い方

実際に複数の命令を組み合わせて、アセンブリ言語でのプログラムを体験する。

- [3章 一生世界に挨拶してそう](#)
 - [3.1 「Hello, World!」](#)
 - [3.2 どれくらいの挨拶を](#)
 - [3.3 出力する回数を指定しよう](#)
 - [おまけ問題](#)

スペースが余ったので、タイトルの小話を書いておく。

プログラミングを学ぶと、どの言語でも基本的には 初めに「Hello, World!の出力」を学ぶ。

そのため、「たくさんのプログラミング言語を知ってるけど、ろくなコードを書けないプログラマ」

「色々な言語に手を出して"さわり"だけ勉強するけど、すぐ別の言語に浮気する実にならない人」を

「Hello World（こんにちは世界）しか出来ないのか？一生世界に挨拶してろ」と揶揄する人がいるらしい。

殺伐としてますね。

こんな文化は新規参入を遠ざけるだけなので、看護師・実習生への嫌がらせ悪循環ぐらい減びるべき。

揶揄されないためにも、この教材を通して「アセンブリ完全に理解した」と言えるように勉強しよう。

「完全に理解した」（ダニング=クルーガー効果）については各自調べてほしいが、

まあ、完全に理解したといえる土俵にすら立てないよりは全然良い。慢心できるくらい理解しよう！

3.1 「Hello, World!」

プログラミング言語を学習すると、大体初めにこれを行う。

まあ、文字を出力できなければ「こいつ何か出来るんか？まず動いてる？」案件になってしまうので、最初に入出力を学ぶんですかね。

ということで、世界に挨拶しよう。「Hello, World!」という文字列を出力する。

文字列を出力するには、OUT 命令を使う。使い方は OUT 出力文字列領域, 文字長領域 だ。

詳しくは 2章 を見返してほしい。

出力文字列領域は「文字列」を意味する character string から string と名付けてみる。

文字長領域は「長さ」length から len とでもしておこうか。

```
MAIN    START
        OUT    string, len
        RET
        END
```

そしたら、string と len に出力したい文字列とその長さを指定しよう。

これを高級言語では「変数宣言」と呼んだりする。

文字列は ' (シングルクォーテーション) で囲う。 'Hello, World!' としよう。

len には文字列の長さを指定する。文字列 Hello, World! は 13文字 だから、13 を指定する。

```
MAIN    START
        OUT    string, len
        RET
string  DC    'Hello, World!'
len     DC    13
        END
```

これで「世界に挨拶」出来そうだ。実行して Output 欄を見てみよう。

このコードが理解出来たら、本節は以上になる。理解できるまで見返そう。ポイントは、

- 文字列の出力には OUT 命令を使う
- OUT 命令は「出力文字列領域, 文字長領域」を指定する。
- それぞれラベルで宣言して、DC 命令でメモリに格納する。
- 文字列は ' で囲う必要があり、文字長には適切な文字数を指定する必要がある。

といったところだろう。

おまけ

「なんで文字列を string なの？紐じゃん」と疑問に思った方向けに説明を書く。

まず、1文字を意味する character という単語がある。ここから、プログラミングの世界では（省略した）char を「文字」の意味で使う。

文字列の正式名称は、先に挙げた character string だ。文字（character）が紐のよう（string）に連なって、列になっているから文字列である。既に character の要素は使われてしまったので、列であることを持ってきて string や str を「文字列」という意味にした。

3.2 くらいの挨拶を

文字を一回だけでなく、たくさん表示したい。他人が見たらドン引きするくらいの変人ムーブをしよう！
一つ思いつくのは、「表示したい数だけ OUT 命令を書く」だろう。

```
MAIN    START
        OUT    string, len ; この行を何個も書く
        OUT    string, len
        OUT    string, len
        OUT    string, len
        OUT    string, len
        RET
string  DC    'Hello, World!'
len     DC    13
        END
```

確かにこれでもたくさん表示できる。しかし、「10000回表示してくれ」となったらどうだろう。
コピー&ペーストでも手打ちでも、10000回ぶん律儀に OUT string, len する？
流石に大変が過ぎるから、別の手を考えよう。

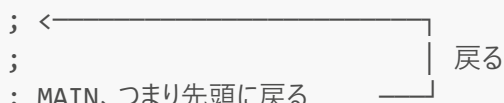
同じことを何回もする。つまり「繰り返す」のだ。

命令には「分岐命令」と呼ばれる分類があった。JUMP とかが入っている。

これを上手く使って、「前に実行したことをもう一度実行する」が出来ないだろうか。

命令は上から順番に 逐次実行 される。「OUTして、前に戻ってもう一回OUTする」と考えてみよう。

```
MAIN    START
        OUT    string, len ;
        JUMP   MAIN        ; MAIN、つまり先頭に戻る
        RET
string  DC    'Hello, World!'
len     DC    13
        END
```



これを実行すると、無条件で先頭へ戻り続けるので、無限に出力され続けてしまう。終わらない。
[||] ボタンを押して止めるか、右上の × を押してアプリを消してください。
ということで、指定された回数だけ戻るように改修しよう。

考え方は、「繰り返した回数が、n回 より小さければ、もっと繰り返す」である。

繰り返しに応じて、繰り返した回数 カウンタ が増えるように、カウンタを作る。

カウンタと 指定する繰り返し回数 を比較して、カウンタの方が小さければ、ジャンプして戻る。

次のページで、流れを追って実装しよう。

まず、繰り返し回数を宣言しよう。とりあえず 5回 繰り返すことを想定する。

`count` と名付けて、`DC` 命令で宣言して 5 を入れてみよう。

今回は `GR0` をカウンタとして運用してみる。繰り返すたびに、`GR0` の値が 1 増えるように設計しよう。

足し算を行うには、`ADDA` 命令がある。`ADDA レジスタ1, リテラル` の形で 1 を指定して使ってみる。

これ以外にも方法があるので、好きに書き換えてみてほしい。

値 1 を格納したアドレスをラベルで宣言して、そのアドレスを指定しても良い。

例えば、`GR1` に 1 を入れて、`ADDA GR0, GR1` のようにレジスタ同士の加算でも良い。

`ADDA` 命令の代わりに `ADDL` 命令を使ってもいい。

そしたら、カウンタ`GR0` と、繰り返し回数 `count` を比較する。

比較を行うには、`CPA` 命令がある。`CPA レジスタ, アドレス` の形で使うと、

レジスタ - アドレス を計算して、`FR`の値を書き換えてくれる。

今回は、実際に繰り返した回数（いま何週目？） - 繰り返すべき回数 `count` なので、

繰り返すときには、引き算の結果がどういう値になるだろうか。

条件に応じてジャンプしよう。`JPL` かな、`JMI` だろうか、`JZE` かも、`JOV` だったり.....？

無条件でジャンプする `JUMP` 命令を、上で考えた条件に合わせて書き換える。

以上を組み合わせ、指定回数だけ繰り返す **forループ** を作ろう。

2章の分岐命令の項目にも、forループのサンプルプログラムが存在する。そちらを参考にしても良い。

```
MAIN      START
          LAD      GR0, 0          ; GR0 をカウンタとして使う。初期値0 を代入

FOR        OUT      string, len    ; 繰り返したい部分の先頭を FOR とラベル付け  <----
          ADDA     GR0, =1          ; カウンタを 1 増やす
          CPL      GR0, count      ; カウンタ と 繰り返し回数 を比較
          JMI      FOR              ; カウンタ < 繰り返し回数 なら FOR に戻る  ----

          RET

string     DC      'Hello, World!'
len        DC      13
count      DC      5              ; 繰り返し回数
          END
```

まとめ

- 繰り返しには 分岐命令 を使って、自分より前（上）に飛ぶようにする。`JUMP` とか `JMI` とか。
- 条件を適切に考えることで、「〇〇回だけ繰り返す」のような処理が行える。
- リテラルを使って宣言を省くと、コードを短く書くことが出来る。

3.3 出力する回数を指定しよう

今のプログラムでは、命令の出力回数を変えるために、`count` の数をいちいち書き直して Assemble し直さなければならない。非常に手間である。この環境だからまだいいが、実際に組み込み機器を作るとなると、「アセンブルして CPU に書き込んで実機で動かす」と、テストするにも時間がかかりすぎる。そこで、「プログラムは変えずに、ユーザーが好きなように回数を変えられる」ように変更したい。具体的には、繰り返す回数を入力にて指定する。

まず、入力を受け取るには `IN` 命令を用いる。`IN` 入力文字列領域、文字長領域 の形だ。入力を受け取って、それを `count` に格納すれば、繰り返す回数を好きなように弄れそうだ。なので、入力文字列領域 には `count` を指定して、文字長領域 には `=1` でも入れておく。入力する文字数を、ラベルを使って `inlen DC 1` など宣言しても良い。

```
MAIN      START
          IN      count, =1      ; 繰り返し回数を標準入力

          LAD     GR0, 0         ; GR0 をカウンタとして使う。初期値0 を代入

FOR        OUT     string, len   ; 繰り返したい部分の先頭を FOR とラベル付け
          ADDA    GR0, =1        ; カウンタを 1 増やす
          CPL     GR0, count     ; カウンタ と 繰り返し回数 を比較
          JMI     FOR           ; カウンタ < 繰り返し回数 なら FOR に戻る

          RET

string    DC      'Hello, World!'
len       DC      13
count     DS      1             ; 繰り返し回数
          END
```

これを実行して、Input欄に 5 を入力する。5回繰り返されるはずだ。

.....

終わらない。ぜんっぜん終わらない。

ずっと待つと、終了時には GR0 が 53 になっている。53回も繰り返したらしい。どうして？

原因は「数字」と「数値」の違いにある。「文字としての数字」と「実際の数」が違うのだ。

1章を思い出してほしい。長い座学の中に、「文字コード」「asciiコード」というものがあつた。何なら 1.2.2 を見てほしい。

文字は、コンピュータ上では2進数として扱われる。それは asciiコード として変換される。

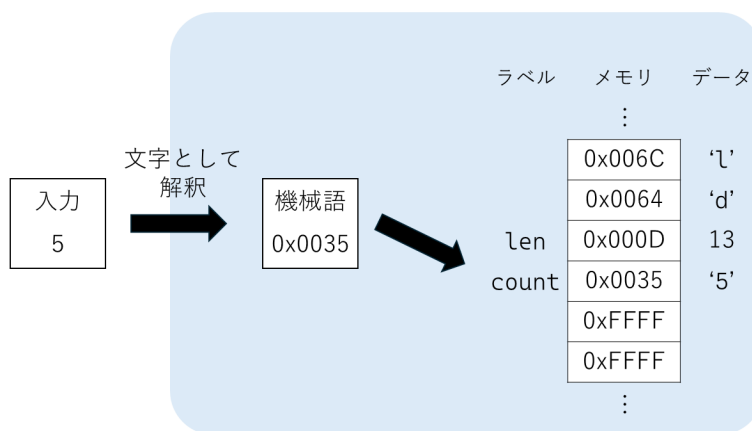
例えば A は 0x41 (10進数で 65) に、a は 0x61 (10進数で 97) に変換される。

数字に対しても変換が行われて、0 は 0x30 (48)、1 は 0x31 (49)、...、9 は 0x39 (57) となる。

IN 命令で受け取った文字列は、「文字の列」であるから、文字として処理される。

5 と入力したなら、コンピュータは 0x35、つまり 53 という数値として処理してしまう。

これを count に格納するから、上のコードは「53回繰り返す」ことになる。



入力した数が、しっかりと「その数値」として解釈してもらえるように、プログラムを改修しよう。

ヒントはasciiコードの並びにある。0 は 0x30、1 は 0x31、...、9 は 0x39 となる。

よく見てみると、数字は 0x3 がついている。数値 8 に 0x30 を足すと、数字 0x38 になる。

つまり、0x30 を引いてあげれば、数字→数値の変換が出来そうだ。

一度、これで変換が出来るか、数値入力のテストコードを書いてみよう。

```
TEST    START
        IN      count, =1
        LD      GR0, count ; count の中身 (つまり入力した数字) を GR0 にロード
        SUBA    GR0, =#0030 ; GR0の値 から、0x30 を引く。10進数で 48 を指定しても良い
        ST      GR0, count ; 計算結果を count に再格納
        LD      GR1, count ; 確認するために、count の中身を GR1 に呼ぶ
        RET
count    DS      1
        END
```

実行したら、0から9までの適当な数字を入力しよう。

GR1 の値が、入力した数値と同じになっていたら成功だ。

これを、先程まで作っていたプログラムに移植する。

```
MAIN    START
        IN      count, =1      ; 繰り返し回数を標準入力

        LD      GR0, count     ; 移植部分
        SUBA    GR0, =#0030
        ST      GR0, count

        LAD     GR0, 0         ; GR0 をカウンタとして使う。初期値0 を代入

FOR      OUT     string, len    ; 繰り返したい部分の先頭を FOR とラベル付け
        ADDA    GR0, =1        ; カウンタを 1 増やす
        CPL     GR0, count     ; カウンタ と 繰り返し回数 を比較
        JMI     FOR           ; カウンタ < 繰り返し回数 なら FOR に戻る

        RET

string   DC      'Hello, World!'
len      DC      13
count    DS      1             ; 繰り返し回数
        END
```

5 を入力したら、しっかり 5回 だけで実行が終わることを確認しよう。

余談だが、「テスト」は大事な考え方になる。

初めから、一気にたくさんの要素を詰め込もうとすると、いざミスがあったときに「どこが間違っているかわからない」状態になってしまうことがある。

そこで、実際に機能を詰め込む前に、「その機能だけ」あっているか確かめるといい。

この検証プログラムを「テストコード」と呼び、小さい機能単体をテストするような場合を「単体テスト」という。

単体テストを行って、いけそうなら実際のプログラムに組み込む。

さらに、組み込んだことで、ほかの部分と噛み合わずバグが起きないか、などもテストする。少し大規模なテストだ。

このように、テストを繰り返しながら 安全にプログラムを作成すると、一見 遠回りに見えても完成が早かったりする。致命的なミスを予防できるからね。

おまけ問題

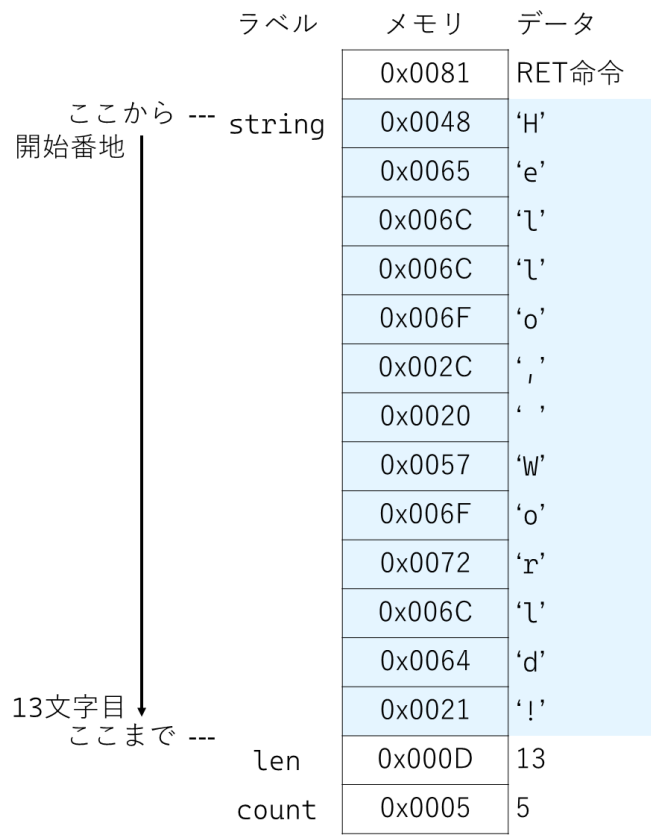
今のプログラムは、何行ぶん出力したのかが分かりにくい。
そこで、1: Hello, World! のように、先頭に何個目の Hello, World! なのか分かるようにしたい。

ヒント

出力する文字列は、「出力文字列領域」と「文字長」で指定する。また、メモリは 命令を上から**連続させて**書き込む。
上手く工夫して、「何行目」「: Hello, World!」を連続させて、文字長を適切に指定しよう。

また、出力する文字列は「文字」である。「数値」のままだと上手くいかないぞ！
では、数値から「数字」にするには、どうしたらいいだろう？

OUT string, len の処理



ヒント2

次のコードの、一部分を書き換えよう。

```
MAIN      START
          IN      count, =1      ; 繰り返し回数を標準入力

          LD      GR0, count      ; 移植部分
          SUBA    GR0, =#0030
          ST      GR0, count

          LAD     GR0, 0          ; GR0 をカウンタとして使う。初期値0 を代入

FOR        ADDA   GR0, =1        ; カウンタを 1 増やす

; ここに、数値→数字の変換を記述
; row に、変換した数字を格納
; 文字列を出力

          CPL     GR0, count      ; カウンタ と 繰り返し回数 を比較
          JMI     FOR            ; カウンタ < 繰り返し回数 なら FOR に戻る

          RET

row        DS      1
string     DC      ': Hello, World!'
len        DC      16            ; 行目が 1文字、': 'が 2文字、全体で 1 + 2 + 13 = 16文字
count      DS      1            ; 繰り返し回数
          END
```