# 02 STEP

## ドットマトリクス LED を点灯させよう

入出力端子の入出力設定、電圧レベル設定について学習していきます。
そして、この STEP では実際にプログラムを組んでいきましょう。

### 2.1　STEP01 の復習

STEP01 では、ドットマトリクス LED の仕組みと入出力端子について学習しました。
ここで、STEP01 の復習問題で正しく理解できているか確認しておきましょう。
問題 2-1、2-2 を解いてください。

#### 問題　2-1

ドットマトリクス LED のアノードとカソードに配線したマイコンの入出力端子 ( ポート 4、ポート B) を入力端子として使用するか、出力端子として使用するか決めなければなりません。
カソードとアノードはどちらの端子に設定すればよいのでしょうか？
入力、または出力に丸を付けてください。

ポート 4 は ( 入力 / 出力 ) 端子に設定する。

ポート B は ( 入力 / 出力 ) 端子に設定する。

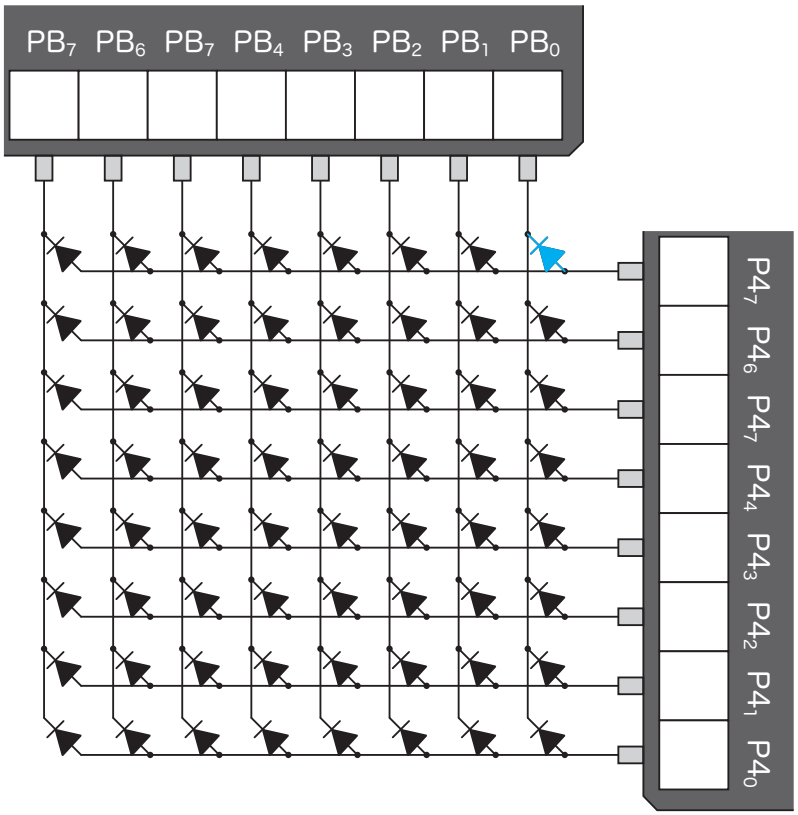
答えは p.41

STEP 02

# ドットマトリクス LED を点灯させよう

## 問題　2-2

ドットマトリクス LED の右上 1 つだけを点灯させるには、ポート 4、ポート B の各入出力端子の電圧レベルを H、L どちらに設定すればいいでしょうか？
下図の□を埋めましょう。




答えは p.41

問題 2-1、2-2 が分からなかった人、間違えた人は STEP01 を見直しておきましょう。
では、実際に入出力端子の入出力設定や、電圧レベル設定はどのように行うのでしょうか？

## 2.2　入出力設定

入出力設定は、データディレクションレジスタ（以後 DDR と表記します）という 8 ビットのレジスタに値を書き込むことで行います。レジスタの値を 0 にすると入力端子になり、1 にすると出力端子に設定されます。レジスタの値を 0 にすることをクリア、1 にすることをセットとも言います。

| ビット： | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | $P4_7DDR$ | $P4_6DDR$ | $P4_5DDR$ | $P4_4DDR$ | $P4_3DDR$ | $P4_2DDR$ | $P4_1DDR$ | $P4_0DDR$ |
| 初期値： | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W： | W | W | W | W | W | W | W | W |

図 2-1　P4DDR と初期値

| ビット： | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | $PB_7DDR$ | $PB_6DDR$ | $PB_5DDR$ | $PB_4DDR$ | $PB_3DDR$ | $PB_2DDR$ | $PB_1DDR$ | $PB_0DDR$ |
| 初期値： | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W： | W | W | W | W | W | W | W | W |

図 2-2　PBDDR と初期値

ポート 4 とポート B の DDR は、上図でも分かるように、初期値は 0 で入力端子になっています。また、DDR に値を書き込み（Write）はできますが、読み込み（Read）は無効です。入力端子に設定されていても読み出されるのは 0 ではなく常に 1 です。

### レジスタ

周辺装置のメモリではなく、マイコンチップに内蔵された記憶回路のことです。主に演算や実行状態の保持などに用いられます。メモリと比較すると、レジスタはきわめて高速に動作し、容量は小さめです。

### ビットとバイト

ビットとはコンピュータが扱うデータの最小単位で、2 進数の 1 桁のことです。1 ビットで 2 通りの状態 (0 か 1) を表現することができます。電子回路の場合、1 ビットで ON と OFF の 2 通りの状態を表現できるわけです。また、8 ビットをまとめて 1 バイトといいます。

## 2.3 電圧レベル設定

電圧レベル設定は、データレジスタ（以後 DR と表記します）という 8 ビットのレジスタに値を書き込むことで行います。DDR で出力に設定された端子に、レジスタの値を 0 にすると L レベルになり、1 にすると H レベルに設定されます。

| ビット： | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | $P4_7$ | $P4_6$ | $P4_5$ | $P4_4$ | $P4_3$ | $P4_2$ | $P4_1$ | $P4_0$ |
| 初期値： | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W： | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

図 2-3 P4DR と初期値

| ビット： | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | $PB_7$ | $PB_6$ | $PB_5$ | $PB_4$ | $PB_3$ | $PB_2$ | $PB_1$ | $PB_0$ |
| 初期値： | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W： | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

図 2-4 PBDR と初期値

ポート 4 とポート B の DR は、上図でも分かるように、初期値は L レベルになっています。DR は読み書き可能なレジスタです。入力端子であれば入力電圧レベルを読み出せ、出力端子であれば、現在の出力電圧レベルを読み出せます。

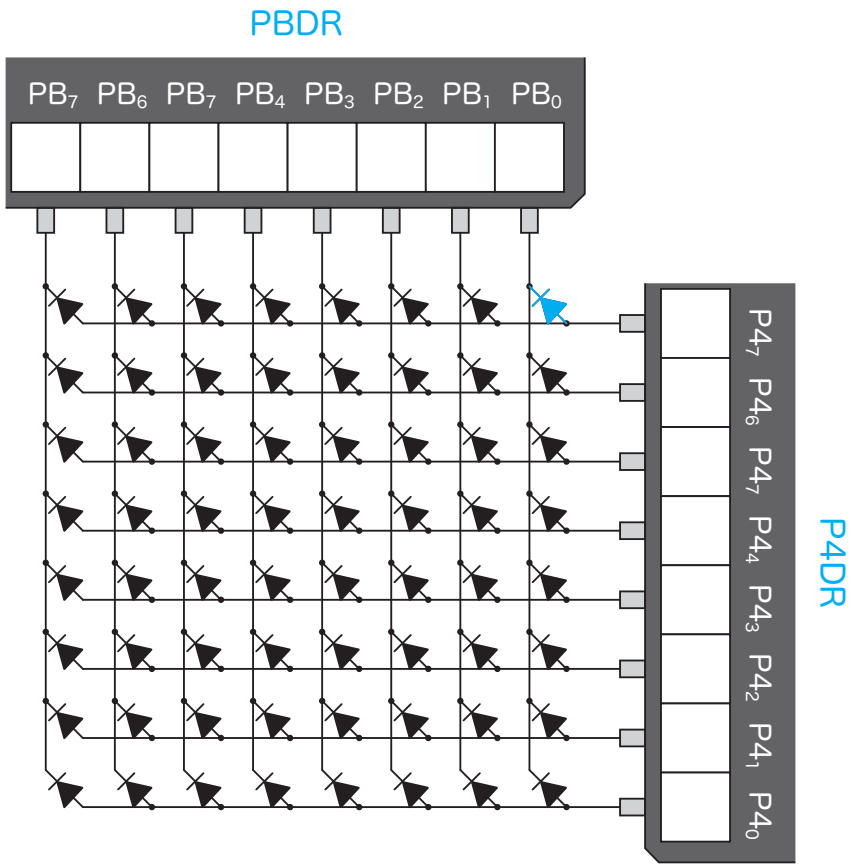
**初期値**

電源投入後のレジスタの値は、初期値としてマイコンのデータシート（仕様書）に記載されています。しかし、プログラムで明示的に初期化するようにすれば、初期値を調べる必要がなく、マイコンが変わっても初期値の違いによるトラブルを未然に防ぐことができます。

実際のプログラムでどのように記述するかは、次の「プログラムの作成」で説明していきます。

問題　2-3

下図のように右上の LED を点灯させるには、P4DR と PBDR の各ビットに 0、1 のどちらを書き込めばいいでしょうか？　下図の□を埋めましょう。



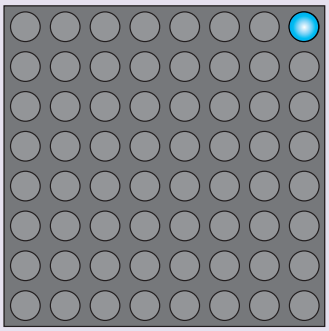

答えは p.42

# STEP 02 ドットマトリクス LED を点灯させよう

入出力設定、電圧レベル設定について理解できたと思います。
では、実際にプログラムを組んでいきましょう。

**課題　2-1**

ドットマトリクス LED の右上の LED1 つだけを点灯させる。



## 2.4　フローチャート

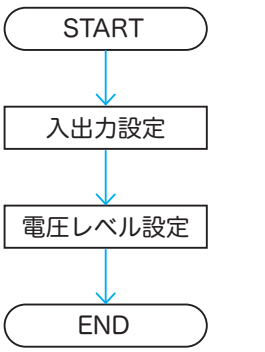



図 2-5　フローチャート

課題に取り組む際は、いきなりプログラムを組むのではなく、まずフローチャートを描くようにしましょう。
図 2-5 のフローチャートを見てください。これは、右上の LED だけを点灯させるフローチャートです。これまで説明してきたように、まずポート 4、ポート B の各入出力端子を出力端子に設定する「入出力設定」を行い、次に右上の LED だけを点灯するように各出力端子の「電圧レベル設定」をします。

これで課題は解決しているのですが、マイコンの場合、これだけでは問題があるのです。

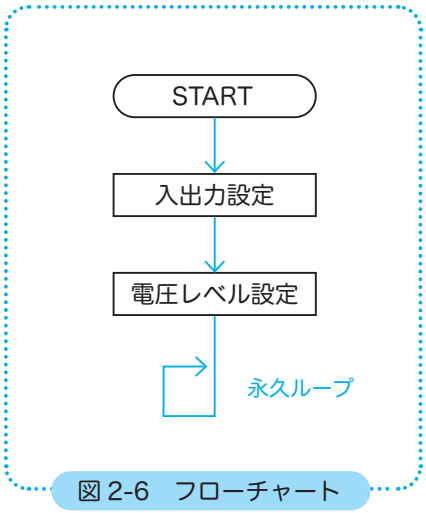
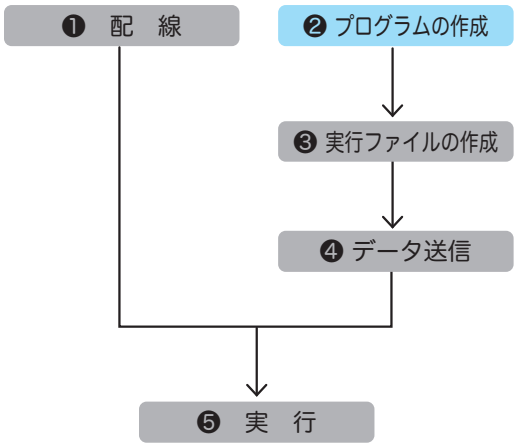



図 2-6　フローチャート

前ページの図 2-5 のように、プログラムを終わらせてマイコンを停止させるような処理を行うべきではありません。マイコンは動き続けていないと、予測不可能な動作をする恐れがあるからです。
そこで、プログラムに永久ループを追加し、繰り返し動き続けるようにします。これによりマイコンの暴走を防ぐことができるのです。

それでは、実際にプログラミングを行っていきましょう。

## 2.5　プログラムの作成




本テキストでは、C 言語と呼ばれるプログラミング言語を使用してプログラムを記述していきます。プログラムはソースコードエディタで記述していきます。

プログラムを書く際、以下のことに注意しましょう。
・基本的に半角英数字、記号を使用する。
・大文字と小文字は区別される。
・空白は Space( 半角 ) や Tab キーを使い、改行は Enter キー使う。
・今回使用するコンパイラの場合、10 進数か 16 進数を使う必要がある。

## 16 進数

2 進数はコンピュータには扱いやすい表記法ですが、数字が大きくなると桁数が増えて、人には判読しにくくなってしまいます。
そこで、一般的に使われるのが 16 進数です。16 進数は、0 ～ 15 までの数を 0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F で表します。
16 進数の 1 桁は 2 進数の 4 桁（4 ビット）に対応します。ですから、桁数が多い場合は 2 進数を 4 桁ずつ区切って 16 進数に置き換えることができます。なお、A ～ F は大文字、小文字のどちらでも構いません。

16 進数を表す時は、数値の前に 0x（ゼロエックス）を付けます。

例） 10 → 0x0A　　18 → 0x12　　175 → 0xAF

| 10 進数 | 2 進数 | 16 進数 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 2 | 10 | 2 |
| 3 | 11 | 3 |
| 4 | 100 | 4 |
| 5 | 101 | 5 |
| 6 | 110 | 6 |
| 7 | 111 | 7 |
| 8 | 1000 | 8 |
| 9 | 1001 | 9 |
| 10 | 1010 | A |
| 11 | 1011 | B |
| 12 | 1100 | C |
| 13 | 1101 | D |
| 14 | 1110 | E |
| 15 | 1111 | F |
| 16 | 1 0000 | 10 |
| 17 | 1 0001 | 11 |
| … | … | … |
| 100 | 110 0100 | 64 |
| 200 | 1100 1000 | C8 |
| … | … | … |
| 255 | 1111 1111 | FF |

## 計算機

Windows の標準アプリ「計算機」の「プログラマー」モードを使うと、進数変換が簡単にできます。

| | |
|---|---|
| HEX（Hexadecimal） | 16 進数 |
| DEC（Decimal） | 10 進数 |
| OCT（Octal） | 8 進数 |
| BIN（Binary） | 2 進数 |

ソースコードエディタを開いたら、プログラム 2-1 のように記述します。
ソースファイルは、HD にコピーした「exercise」フォルダ内の STEP02-1 > step2-1.c をお使いいただいても構いません。
プログラム 2-1 は、C 言語のプログラムの骨格となる部分です。
※ プログラムの左端の 2 桁の数字は行番号です。実際のプログラムには記述不要です。

プログラム 2-1

```
01  /************************************************************
02      製作者      アドウィン
03      解説        ドットマトリクス LED の右上点灯
04  ************************************************************/
05  #include <3052f.h> // 3052f.h の読み込み
06
07  /*
08   *  main 関数
09   */
10  int main(void)
11  {
12
13
14
15      ここに実行したい処理を上から順番に記述していきます
16
17
18
19
20      /* 永久ループ */
21      while (1)
22          ;
23
24      return 0;
25  }
```

前ページの注意書きで「基本的に半角英数字、記号を使用する」とあるにも関わらず、プログラム 2-1 には日本語、つまり全角文字が使用されています。実は、コメントには日本語を記述することができます。

では、コメントも含め、プログラム 2-1 について上から順番に説明していきます。

## 【 コメント文 】

コメント文とは、プログラマのメモ書きにあたるもので、C 言語の場合、範囲コメントと行コメントがあります。
コメントは、プログラムではないので全角の日本語でも書くことができます。
プログラムを分かりやすくするために、こまめにコメントを書くようにしましょう。

範囲コメント

```
/*
    コメント
*/
```

/* から */ の間が全てコメントになります

行コメント

```
// コメント
```

// から改行までがコメントになります

プログラム 2-1 の 20 行目のように、1 行を範囲コメントで記述することもできます。

## 【 ヘッダファイルの読み込み 】

```
#include  <3052f.h>        3052f.h ヘッダファイルを読み込みます
```

プログラム 2-1 の 6 行目は、#include という命令で「3052f.h」というヘッダファイルの読み込みを行っています。「3052f.h」には「レジスタ定義」が記述されており、H8/3052F の各レジスタの番地と名前を対応させるものです。このファイルを使うと、分かり易い名前を使ってレジスタにアクセスすることができます。

このファイルを使わず番地を直接指定してアクセスすることもできるのですが、番地を間違えると、プログラムが動作しない、マイコンが暴走してしまうといったことが起こる可能性が高くなってしまいます。また、番地を正確に覚えなければならないのでプログラムを組むのも大変です。

「3052f.h」は exercise > include フォルダ内にあるので、ソースコードエディタで開いて見てみるのもいいでしょう。なお、このヘッダファイルを編集すると、作成したプログラムが機能しなくなるので、内容が理解できないうちは変更しないでください。

## 【 main 関数 】

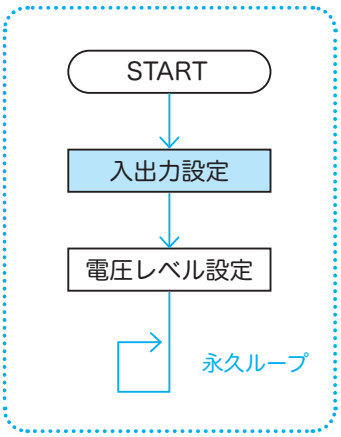
```
int main(void)
{
    ここに実行したい処理を上から順番に記述していきます

    return 0;
}
```

プログラム 2.1 の 9 行目から 15 行目は main 関数といい、文字通り C 言語のメインとなる部分です。
プログラムを実行すると { } 内に記述した処理を上から 1 行ずつ実行していきます。
また、return 0; が実行されるとプログラムが終了します。つまり、行いたい処理は return 0; の前、上枠内の□の中に、上から順番に記述します。
先程も言いましたが、マイコンはプログラムが終了しないように永久ループをプログラムの最後に付けなくてはいけません。ですから永久ループは、return 0; の前に記述することになります。

## 2.6　プログラミング

それでは、main 関数に処理を記述していきましょう。
フローチャートによると、はじめにポート 4 とポート B の入出力設定を行います。
「2.2 入出力設定」で説明したように、入出力設定は、P4DDR と PBDDR に値を書き込む必要があります。




### 【 レジスタに値を書き込む 】

```
レジスタ　＝　値 ;
```

「レジスタ」、「＝」、「値」、「；」には、以下のような意味があります。

・「レジスタ」には、「3052f.h」で定義されているレジスタの名前を記述します。

```
P4DDR は　P4.DDR
PBDDR は　PB.DDR
```

・C 言語の「=」は代入という意味です。等しいという意味ではないので注意してください。

・「値」には、書き込みたい値を記述します。先ほども述べたように、表記法は 10 進数と 16 進数のみです。DDR はバイト単位で値を書き込みます。

・最後は、「; ( セミコロン )」を付けます。セミコロンは処理の終わりを表します。何かしら処理を記述すると、必ず最後にセミコロンを付けます。

例 1） P4DDR に 10 進数の 5 を書き込む

```
P4.DDR = 5 ;
```

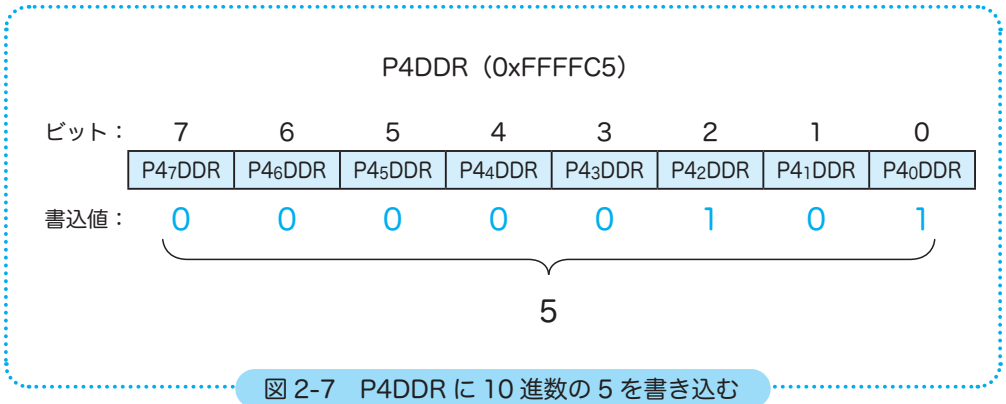



図 2-7　P4DDR に 10 進数の 5 を書き込む

例 2） PBDDR に 16 進数の A を書き込む

```
PB.DDR = 0x0A ;
```

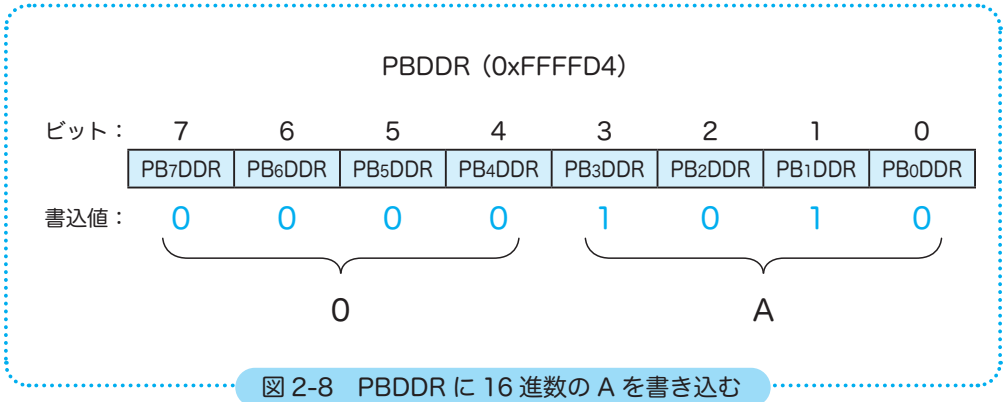
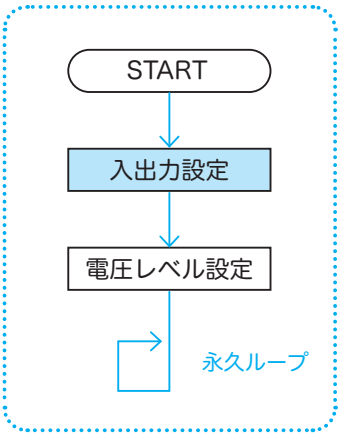



図 2-8　PBDDR に 16 進数の A を書き込む

次は、電圧レベル設定を行います。

「2.3 電圧レベル設定」で説明したように、電圧レベル設定は、P4DR と PBDR に値を書き込む必要があります。




DR はバイト単位、ビット単位のどちらでもアクセスできますが、ここではバイト単位で書き込むことにします。ビット単位でアクセスする方法については STEP04 で解説します。

P4DR、PBDR にバイト単位でアクセスする場合は、次のような名前を記述します。

P4DR は　P4.DR.BYTE

PBDR は　PB.DR.BYTE

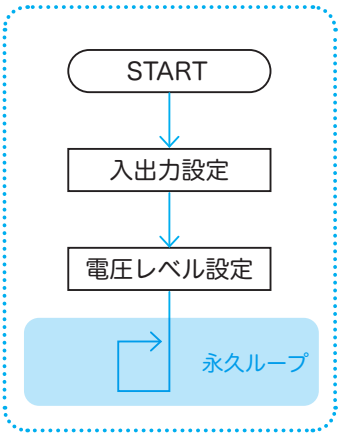
例 1）　P4DR に 10 進数の 10 を 16 進数で書き込む

```
P4.DR.BYTE = 0x0A;
```

例 2）　PBDR に 2 進数の 1111 1111 を 16 進数で書き込む

```
PB.DR.BYTE = 0xFF;
```

最後に永久ループです。
永久ループの記述には、繰り返し構文である while 文を使います。




## 【 while 文 】

```
while ( 継続条件式 )
    処理 ;
```

継続条件式が「真 ( 条件が成り立っている )」ならば、直後の 1 行の処理を繰り返し実行します。継続条件式が「偽 ( 条件が成り立っていない )」ならば、処理は実行されません。

また、複数の処理を繰り返したい場合は、下記のように処理を { } で囲みます。

```
while ( 継続条件式 )
{
    処理 1
    処理 2
    …
}
```

例）永久ループ

```
while(1)
    ;
```

上記 while 文の条件式は「1」、繰り返したい処理は;（セミコロン）だけですから、「処理なし」となります。C 言語では、0 は偽、0 以外 ( 一般的には 1) は真という意味があります。つまり、while(1) と記述すると、常に条件式が真の状態になるので、永久に処理を繰り返し続けるようになるのです。

# STEP 02 ドットマトリクス LED を点灯させよう

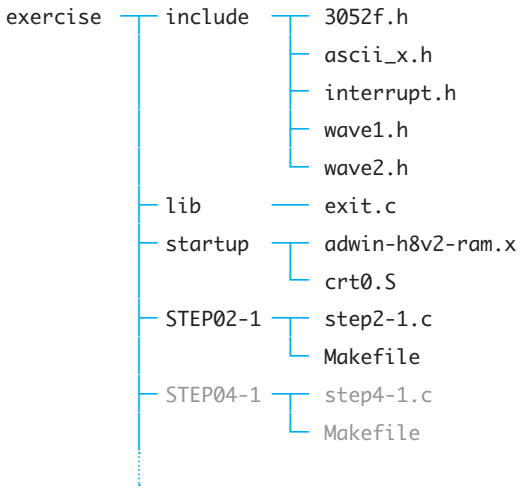
## プログラム　2-1

課題 2.1 を実現するプログラムを完成させましょう。

解答例は p.43

付属 CD に収録された exercise フォルダを、パソコンにコピーしてください。
C ドライブ直下にコピーすれば WSL（Ubuntu）からアクセスし易いですが、任意のディレクトリでも構いません。

exercise ＞ STEP02-1 ＞ step2-1.c をソースコードエディタで開いて編集してください。プログラムが完成したらそのまま上書き保存してください。C 言語のソースコードなので、拡張子は「.c」をつけておく必要があります。




※ STEP04-1 以降のフォルダやファイルは、実習時に自作してください。

図 2-9　exercise フォルダのディレクトリ構成

## 解答例　2-1

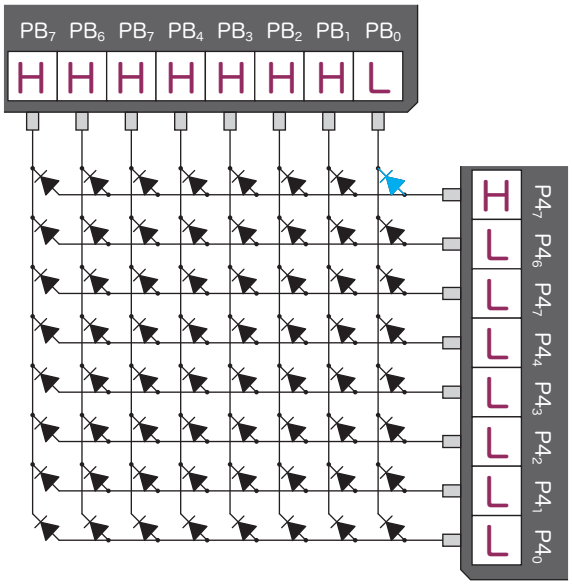
ポート 4 は ( 入力 / 出力 ) 端子に設定する。

ポート B は ( 入力 / 出力 ) 端子に設定する。

ドットマトリクス LED のアノード側のポート 4 も、カソード側のポート B も出力端子です。「カソード側は電流が流れ込むので入力端子かな？」と誤解しそうですが、入力端子に設定するのは、入力装置が接続され、端子の状態を読み込んで使う場合です。

## 解答例　2-2

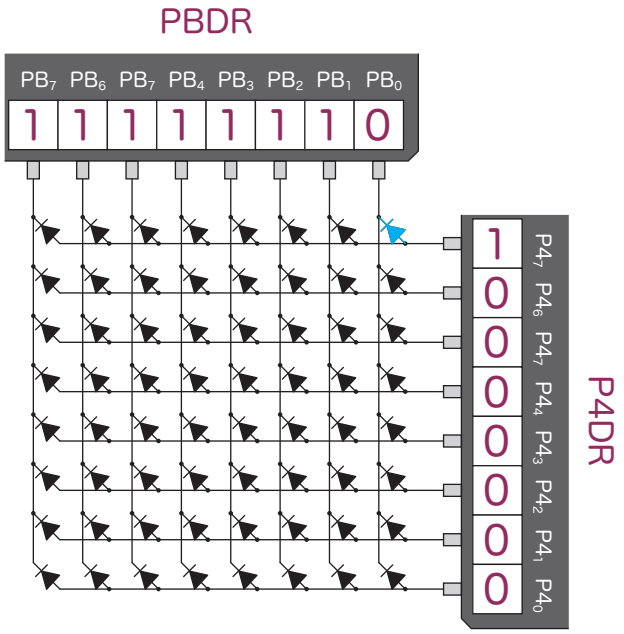
ドットマトリクス LED の右上 1 つだけを点灯させる




点灯させたい LED だけアノードを H、カソードを L にしなければならないので、正解は上記のパターンのみです。

## 解答例　2-3

下図のように右上の LED を点灯させるには、P4DR と PBDR の各ビットに 0、1 のどちらを書き込めばいいでしょうか？　下図の□を埋めましょう。




この解答例も前の「解答例 2-2」と同じで、アノード・カソードの両端子をＬ（0）にして消灯します。

プログラム例　2-1

```
05  #include <3052f.h> // 3052f.h の読み込み
06
07  /*
08   *  main 関数
09   */
10  int main(void)
11  {
12      // 入出力設定
13      P4.DDR = 0xFF; // 出力 LED 横行
14      PB.DDR = 0xFF; // 出力 LED 縦行
15
16      // 出力レベル設定
17      P4.DR.BYTE = 0x80; // 1000 0000 アノード
18      PB.DR.BYTE = 0xFE; // 1111 1110 カソード
19
20      // 永久ループ
21      while (1)
22          ;
23
24      return 0;
25  }
```

DDR の記述

DR の記述

永久ループの記述