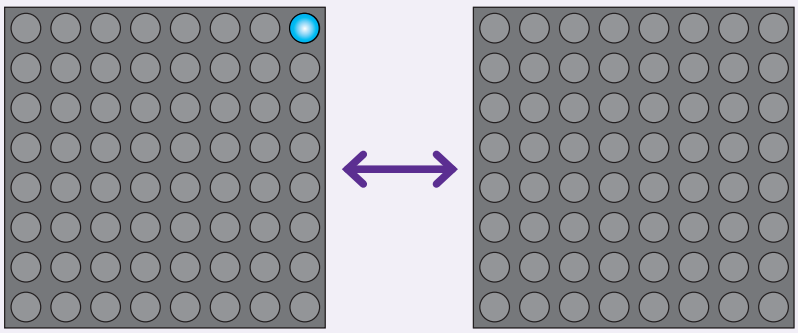
04 STEP

ドットマトリクス LED を点滅させよう

STEP02 で組んだドットマトリクス LED 点灯プログラムの流用で、ここでは LED を点滅させてみましょう。
ドットマトリクス LED を点滅させるには、マイコンの処理速度を考慮してプログラムを組まなければなりません。

課題　4-1

ドットマトリクス LED の右上の LED1 つだけを点滅させる。



STEP 04 ドットマトリクス LED を点滅させよう

課題 4-1 の図を見れば分かると思いますが、点滅とは、点灯と消灯を繰り返している状態です。LED の点灯、消灯、繰り返しは既に学習済みです。LED の点灯、消灯は STEP02 で学習したように、電圧レベルを設定することでできます。そして、繰り返し処理も STEP02 で学習済みです。STEP02 では、プログラムを終了させないために、何もしない処理を繰り返す永久ループを使用しました。この永久ループを利用して繰り返し処理を行います。
まずは、フローチャートを描いて、どのような流れになるのかを考えてみましょう。
問題 4-1 を解いてください。

問題　4-1

ドットマトリクス LED の右上の LED を点滅させるフローチャートを考えてみましょう。
ヒント：ドットマトリクス LED の点灯と消灯を永久ループの中に組み込んでしまえば、点灯と消灯が永久に繰り返されることになります。

答えは p.59

ドットマトリクス LED の点滅は永久ループの中に点灯と消灯を組み込んで、点灯と消灯を永久に繰り返すようにすればいいので、図 4-1 のようなフローチャートが考えられます。

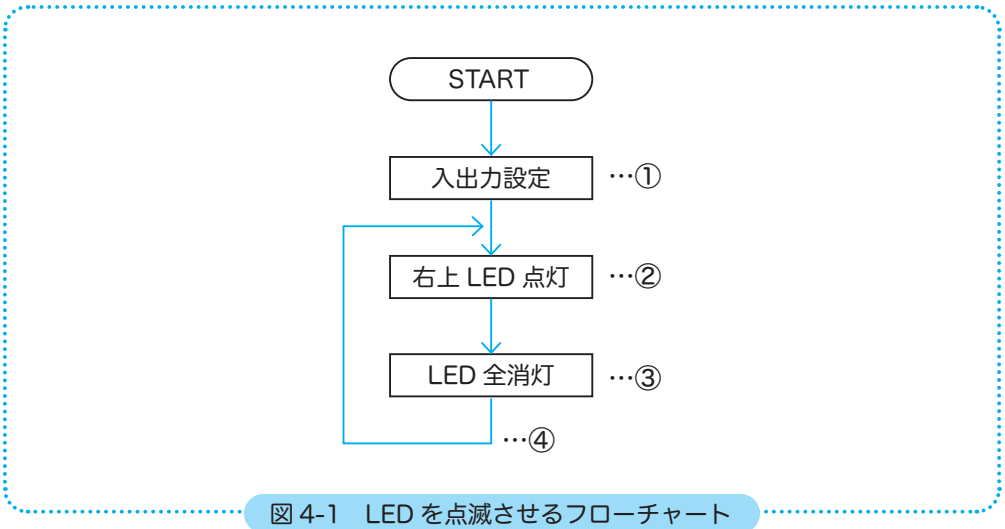


図 4-1　LED を点滅させるフローチャート

図 4-1 の①～④について解説します。

① 入出力設定を行います。使用する装置はドットマトリクス LED なので、出力に設定します。
② ドットマトリクス LED の右上の LED を点灯させるの処理は STEP02 と同じです。
③ 右上の LED を消灯させます。この段階で点滅が 1 回行われます。
④ 永久ループにより、再び右上の LED が点灯する処理に戻ります。

図 4-1 を見ていただければ分かるように、永久ループ内で LED の点灯と消灯を繰り返しています。こうすることで、LED の点灯と消灯を永久的に繰り返すようになります。また、マイコン制御の場合、プログラムが終了しないようにしなければなりませんでした。LED の点滅を永久ループの中に入れることにより、プログラムが終了するはありません。

では、このフローチャートを基にプログラムを組んでみましょう。
プログラム 4-1 を解いてください。

STEP 04 ドットマトリクス LED を点滅させよう

問題 4-1 で描いたフローチャートを基にプログラムを組んでみましょう。
「STEP02-1」フォルダを複製して「STEP04」とするといいでしょう。Makefile 内で指定したソースファイル名と実際のソースファイル名を一致させることを忘れないでください。

プログラム　4-1

課題 4-1 を実現するプログラムを完成させましょう。

```
#include <3052f.h> // 3052F 固有の定数

/*
 *  main 関数
 */
int main(void)
{
    // 入出力設定


    // 出力レベル設定


    // 永久ループ
    while (1)
    {


    }

    return 0;
}
```

解答例は p.62

4.1　LED を点滅させる

p.18 でも紹介しましたが、ドットマトリクス LED は LED のアノード、カソードの両方をマイコン端子に接続するので、表 4-2 のように 4 パターンの組み合わせがあります。その中で LED が点灯するのはアノードが H、カソードが L の時だけです。その他の組み合わせでは消灯になります。
つまり点滅は、アノードを H で固定しカソードの H/L で点滅（a ⇔ b）させるか、カソードを L で固定しアノードの H/L で点滅（b ⇔ d）させる 2 通りの方法が使えるということです。

ですから、PBDR を固定にして P4DR を変化させるか。P4DR を固定にして PBDR を変化させるかのどちらかでいいでしょう。解答例では、PBDR を固定にして P4DR を変化させています。

	アノード	カソード	LED
a	H	H	 消灯
b	H	L	 点灯
c	L	H	 消灯
d	L	L	 消灯

表 4-2　LED 点滅組み合わせ

STEP 04 ドットマトリクス LED を点滅させよう

プログラム例　4-1

```
#include <3052f.h> // 3052F 固有の定数

/*
 *  main 関数
 */
int main(void)
{
    // 入出力設定
    P4.DDR = 0xFF; // 出力 LED 横行
    PB.DDR = 0xFF; // 出力 LED 縦行

    // 出力レベル設定
    PB.DR.BYTE = 0xFE; // 1111 1110 カソード

    // 永久ループ
    while (1)
    {
        P4.DR.BYTE = 0x80; // 右上の LED を点灯
        P4.DR.BYTE = 0x00; // 全消灯
    }

    return 0;
}
```

プログラムが完成したら、WSL（Ubuntu）で make を行い、Tera Term で送信、実行しましょう。忘れてしまった方は、STEP03 を読み返してください。

さて、ドットマトリクス LED は点滅しましたか？
ドットマトリクス LED は点灯しているだけのように見えたと思います。なぜでしょうか？

実は、高速波形を観測できるオシロスコープで見ると、ちゃんと点滅しています。ただ、あまりにも高速で点滅しているので、人間の目では追いきれず点灯しているようにしか見えないのです。

プログラムはマイコンが処理をします。ということは、マイコンの処理速度がプログラムの実行速度ということになります。では、マイコンの処理速度はどのくらいなのでしょうか？
マイコンの処理速度は、Hz（ヘルツ）という単位で表されるクロック周波数によって決まります。
マイコンはクロックという信号に合わせて処理を行います。このクロックが 1 秒間に発生する回数をクロック周波数と言います。ですから、一般的にクロック周波数の値が大きいほど、コンピュータの処理速度が速いと言えます。

例えば、クロック周波数が 10MHz のマイコンの場合、1 秒間に 10 × 10^6 回クロックが発生します。つまり、下式で求めるように、100n 秒に 1 回処理が行われます。

1[秒] ÷ 10[MHz] ＝ 100[n 秒]

本キットの H8/3052F モジュールは、25MHz の外部水晶発振子で動作しており、水晶発振子からクロック信号が発生するたびに、処理が行われます。

では、H8/3052F は何秒に 1 回処理を行っているのでしょうか？
計算をして求めてみましょう。

問題　4-3

クロック周波数が 25MHz の時、何秒に 1 回処理を行っているのか求めましょう。

接頭辞	記号	10^n
テラ	T	10^9
メガ	M	10^6
キロ	k	10^3
ミリ	m	10^{-3}
マイクロ	μ	10^{-6}
ナノ	n	10^{-9}

[　　秒]

答えは下記解説内

計算で求めたように、H8/3052F は 40n 秒に 1 回処理を行っています。
ここで注意しておかなければならないのは、「C 言語のコマンド 1 行が 1 クロックで処理されるわけではない」ということです。実際にプログラム 4-1 を実行して、オシロスコープで点滅速度を観測してみると、880[n 秒] 程度の周期で点滅していました。点灯→消灯→繰り返しの 3 コマンドで 880[n 秒] ですから、1 コマンドで数クロックかかっている計算になります。それでも、これだけ速いと人間の目には LED が点灯しているように見えてしまいます。

そこで、LED の点灯と消灯の間に待ち時間を入れれば、LED が点滅しているように見せることができます。

4.2 待ち時間

待ち時間は、「空(から)ループ」という簡単な方法でつくってみましょう。空ループでマイコンにカウント作業をさせるのです。つまり、繰り返す回数を変えることによって待ち時間をある程度調節することができます。

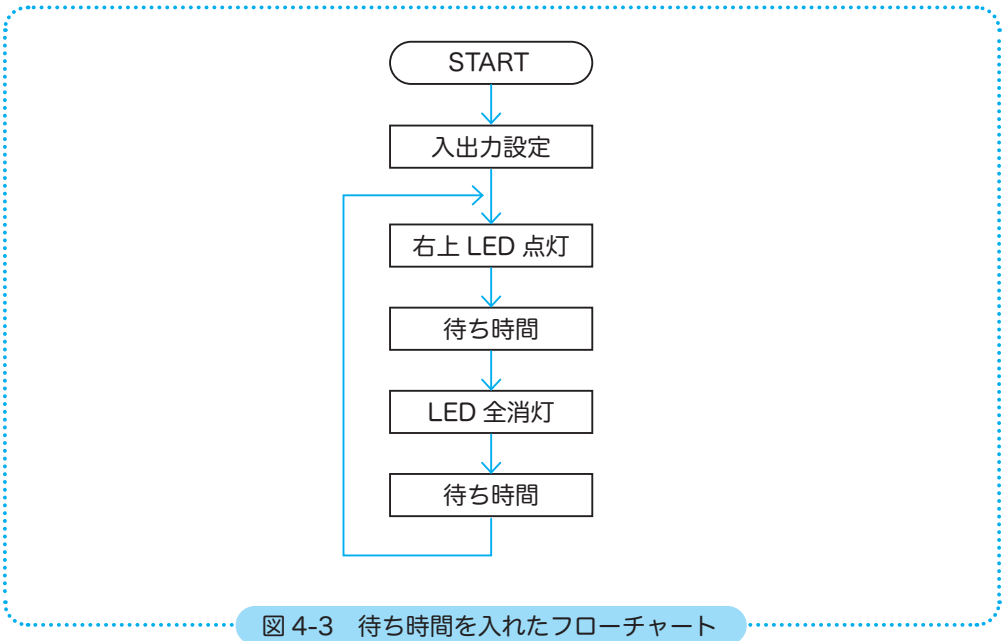
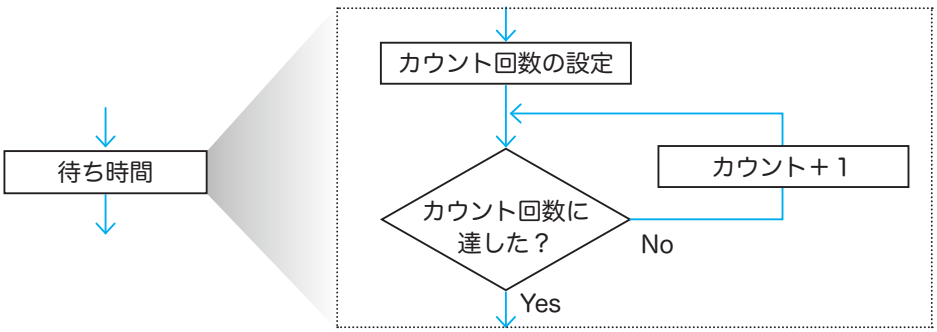


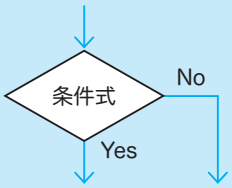
図 4-3 待ち時間を入れたフローチャート

「待ち時間」の中で行われる処理を、さらに詳細なフローチャートにすると、以下のように表せます。



条件分岐

フローチャートで条件分岐は右図のように描きます。条件が満たされた時は Yes に、満たされなかった時は No に処理を続けて記していきます。



実際にプログラムではどのように記述するのか考えていきましょう。例えば、カウント回数を 2 万回に設定した待ち時間処理は以下のように記述します。

```
int i;                          // 変数の宣言
for ( i=0; i<20000; i++ )       // 待ち時間
    ;
```

では、待ち時間処理で使われている各要素を解説していきます。

【 変数 】

変数は文字や値を入れる箱のようなものです。今回は、何回カウントしたかを記憶しなければならないので、変数を使っています。
箱を使うには、まず箱を作り、箱の中に文字や値を入れていきます。箱を作ることを変数の宣言と言い、箱に文字や値を入れることを値の代入と言います。

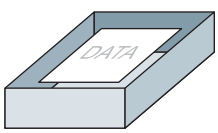


図 4-4　変数のイメージ

C 言語では、次のように記述します。

変数の宣言

```
データ型 変数名 ;
```

・変数名は、1 文字がアルファベットであり、予約語 (コンピュータで予め設定されている単語) を使用していないものに限ります。
・データ型とは、変数に入れるデータの種類のことです。整数型、文字型などがあります。
・複数の変数を 1 度に宣言することができます。その場合は、変数名を ,(コンマ) で区切ります。
・変数宣言は、ブロックの先頭で行わなければなりません。
※ブロックとは、{ } で囲まれた範囲のことを言います。p.38 の while 文の説明では、複数の処理を繰り返したい場合、{ } で処理を囲むと書いてありますが、これは、処理をブロックで囲むということです。while 文に限らず、これから学習する for 文や if 文も複数の処理を実行する場合、処理をブロックで囲みます。

値の代入

```
変数名 = 値 ;
```

変数の宣言と値の代入を同時に行うこともできます。

```
データ型 変数名 = 値 ;
```

```
int i;          // 変数の宣言
```

この例では、データ型は「int 型」、変数名が「i」の変数を宣言しています。
H8/3052F は 16 ビットマイコンなので、扱える最大数は 16 ビット＝ 65535 となります。「int 型」は、－ 37268 ～ 37267 の範囲の 65535 個の整数を扱えます。つまり、変数「i」は 37268 ～ 37267 の値を入れることができます。

【 for 文 】

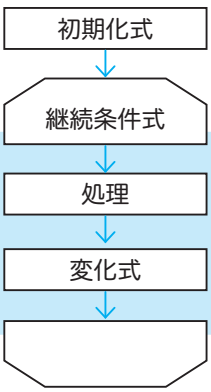
for 文とは、while 文と同じ繰り返し構文です。指定した回数分だけ処理を繰り返します。C 言語では以下のように記述します。

```
for( 初期化式 ; 継続条件式 ; 変化式 )
{
    処理
}
```

変数の初期値が変化式によって変化した後、継続条件式が真ならば直後の 1 行か、ブロックの処理を実行します。継続条件式が偽ならば、処理は実行されません。

for 文のフローチャート

for 文をフローチャートで表すには右図のような描き方もあります。継続条件式が真の間はループを繰り返します。継続条件式が偽になると、ループを抜けて次の処理に移ります。



for 文と while 文の使い分け

while 文も for 文と同じく繰り返し構文の一種です。while 文は指定した条件を満たさなくなるまで処理を繰り返します。一方、for 文は、数を数える場合によく使用されます。

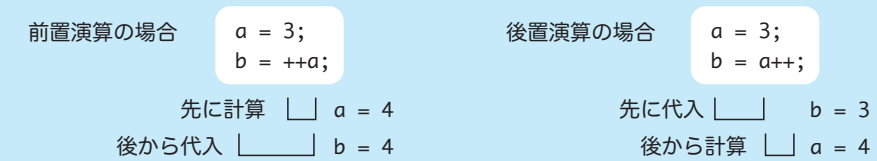
【 インクリメント演算子 】

インクリメント演算子は、整数型の変数の値をインクリメント (1 だけ増加) する際に用いられる演算子です。C 言語では、以下のように記述します。

＋＋変数　:変数に 1 を加える (前置演算)

変数＋＋　:変数に 1 を加える (後置演算)

前置演算と後置演算の違い



インクリメント演算子は、ループ文と組み合わせて使用することが多く、使用頻度も高いです。では、ここで「待ち時間処理」をもう一度見てみましょう。

```
int i;                          // 変数の宣言
for ( i=0; i<20000; i++ )       // 待ち時間
    ;
```

変数の宣言をした「i」は数える回数を記憶します。i の初期値は「0」、継続条件式は「i が 20000 未満」なので、i は 0 か 20000 までカウントされます。変化式は「i++」となっているので、i は 1 ずつカウントされますね。

待ち時間処理の記述について理解できたでしょうか？
では、これらのことを踏まえて、ドットマトリクス LED を点滅させるプログラム 4-2 を組んでいきましょう。

プログラム例　4-2

```
#include <3052f.h> // 3052F 固有の定数

/*
 *  main 関数
 */
int main(void)
{
    int i; // 待ち時間処理で使う変数の宣言

    // 入出力設定
    P4.DDR = 0xFF; // 出力 LED 横行
    PB.DDR = 0xFF; // 出力 LED 縦行

    // 出力レベル設定
    PB.DR.BYTE = 0xFE; // 1111 1110 カソード

    // 永久ループ
    while (1)
    {
        P4.DR.BYTE = 0x80;          // 右上の LED を点灯
        for (i = 0; i < 20000; i++) // 待ち時間 )
            ;

        P4.DR.BYTE = 0x00;          // 全消灯
        for (i = 0; i < 20000; i++) // 待ち時間 )
            ;
    }

    return 0;
}
```

ドットマトリクス LED は、点滅しているように見えましたか？
残念ながら、まだ点滅しているようには見えないですね。繰り返し回数を int 型の最大値の 32767 回繰り返してもマイコンの実行速度が速すぎて目視では点滅が確認できません。
32767 以上の回数にすると、カウントできずにオシロスコープで見ても点滅しなくなってしまいます。どうすればいいでしょうか。

STEP 04 ドットマトリクス LED を点滅させよう

もう一つ変数を用意してループを二重にすれば待ち時間を長くすることができます。

以下のプログラム例では、i 変数のループで、何もしない処理を 32767 回繰り返してわずかな待ち時間をつくり、更にそれを j 変数のループで 100 回繰り返しています。つまり、3,276,700 回ループさせているのと同じです。このプログラムを試してみるときは j の変数宣言を忘れないようにしてください。

```
// 点滅の永久ループ
while (1)
{
    P4.DR.BYTE = 0x80; // 右上の LED を点灯

    for (j = 0; j < 100; j++) // 待ち時間 (a × 500)
    {
        for (i = 0; i < 32767; i++) // 待ち時間 (a)
            ;
    }

    P4.DR.BYTE = 0x00; // 全消灯

    for (j = 0; j < 100; j++) // 待ち時間 (a × 500)
    {
        for (i = 0; i < 32767; i++) // 待ち時間 (a)
            ;
    }
}
```

図 4-5　ループの二重化

「空ループ」検証

「空(から)ループ」は分かり易くて簡単な待ち時間処理ですが、マイコンは空ループの間、他の処理をすることができないため、全体としての処理効率が良くありません。また、以下のような問題もあります。

C 言語ソースファイルは make コマンドによって、mot ファイルに変換処理されますが、その際、コンパイラにより空ループの扱いが異なるのです。コンパイラによっては、空ループを無駄な処理として省略してしまう場合もあります。同じコンパイラでも最適化レベルによって空ループの処理時間が変わる場合もあります。

オシロスコープで出力波形を実測して、空ループで 1ms の待ち時間をつくるカウント数を調べてみました。for ループの処理が全くの空(から)だと実習環境のコンパイラでは待ち時間とループ回数が比例しません。そこで、ループ中の処理に k++ を入れるとループ時間が安定しました。マイコンは H8/3052F、25MHz です。

■ Ubuntu ＋コンパイラ：hms-8300

```
for (i = 0; i < 1560; i++)
    k++;
```

■ HEW

```
for (i = 0; i < 2070; i++)
    k++;
```

HEW の場合、最適化レベルや k++ の有無によって処理速度は変わりませんでした。

プログラム例　4-3

```
#include <3052f.h> // 3052F 固有の定数

/*
 *  main 関数
 */
int main(void)
{
    int i, j, k; // 待ち時間処理で使う変数の宣言

    // 入出力設定
    P4.DDR = 0xFF; // 出力 LED 横行
    PB.DDR = 0xFF; // 出力 LED 縦行

    // 出力レベル設定
    PB.DR.BYTE = 0xFE; // 1111 1110 カソード

    // 永久ループ
    while (1)
    {
        P4.DR.BYTE = 0x80; // 右上の LED を点灯

        for (j = 0; j < 500; j++) // 待ち時間 (1[m 秒 ] × 500)
        {
            for (i = 0; i < 1560; i++) // 待ち時間 (1[m 秒 ])
                k++;
        }

        P4.DR.BYTE = 0x00; // 全消灯

        for (j = 0; j < 500; j++) // 待ち時間 (1[m 秒 ] × 500)
        {
            for (i = 0; i < 1560; i++) // 待ち時間 (1[m 秒 ])
                k++;
        }
    }

    return 0;
}
```

「ループの多重化」と「空ループの検証結果」を反映して、上記のようなプログラムにすれば、LED が点滅して見えるようになります。