10 STEP

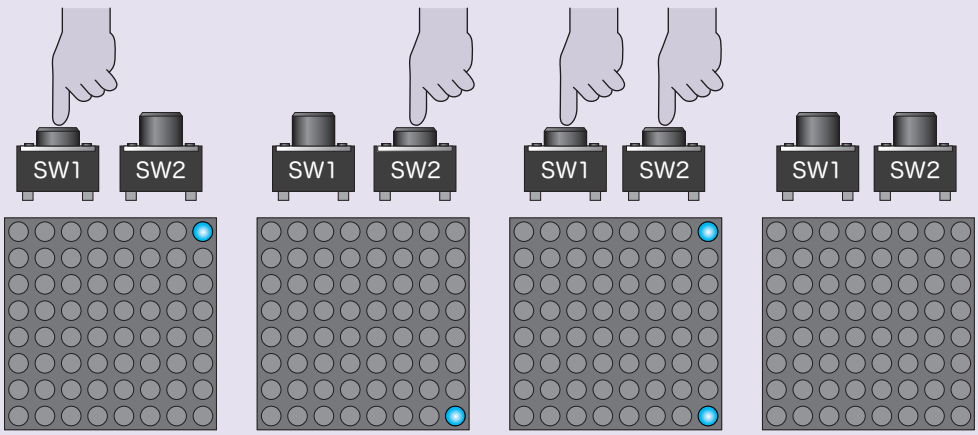
SW を 2 つ使おう

STEP09 では SW を 1 つしか使いませんでしたが、この STEP では SW1、SW2 の 2 つの SW を使ってプログラムを組みます。

課題　10-1

STEP09 の SW1 の動作に追加して、SW2 を押したら右下の LED が点灯するようにしましょう。

SW1 だけを押している間は右上の LED が点灯
SW2 だけを押している間は右下の LED が点灯
SW1、SW2 を押している間は右上と右下の LED が点灯
SW1、SW2 を押していない状態では LED は消灯



STEP 10 SW を使ってドットマトリクス LED を点灯させる

10.1 STEP09 の復習

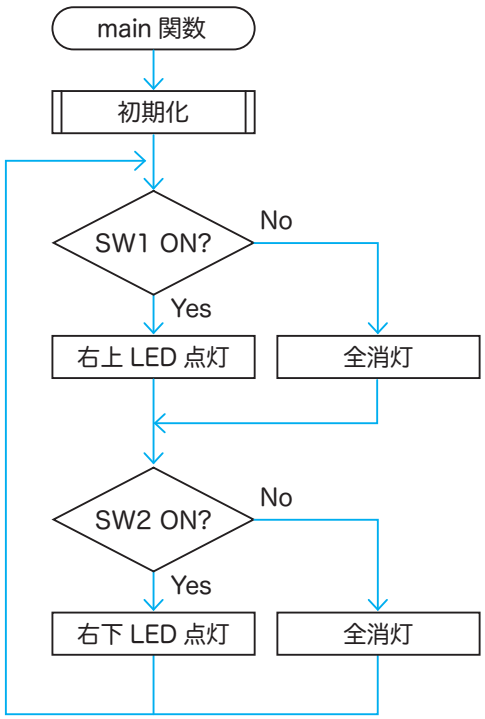


図 10-1 簡単なフローチャート

SW2 の条件分岐を SW1 の後に追加しただけのフローチャートです。プログラムを組んでみると、うまく動作しているように見えます。LED は高速で点滅を繰り返しているのですが、人の目には点灯して見えるのです。待ち時間を入れて動作を遅くしてみれば点滅を確認することができます。

しかし、このフローチャートでは、SW1 と SW2 の両方を押したとき別の点灯パターンにしたいときは対応できません。

10.2　入れ子を使ったフローチャート (パターン 1)

課題を解決する方法の 1 つに、「入れ子」を使ったフローチャートが考えられます。
入れ子とは、1 つの処理の中に別の処理を入れた構造のことです。ロシアのマトリョーシカ人形のような構造だと思ってください。今まで、待ち時間処理は for 文を 2 重にして組んできましたが、これも入れ子になっていると言えます。
図 10-1 は if 文の中にさらに if 文を入れ子にしたフローチャートです。前のフローチャートでは実現不可能だった SW を両方押したときに全点灯するようにしましょう。

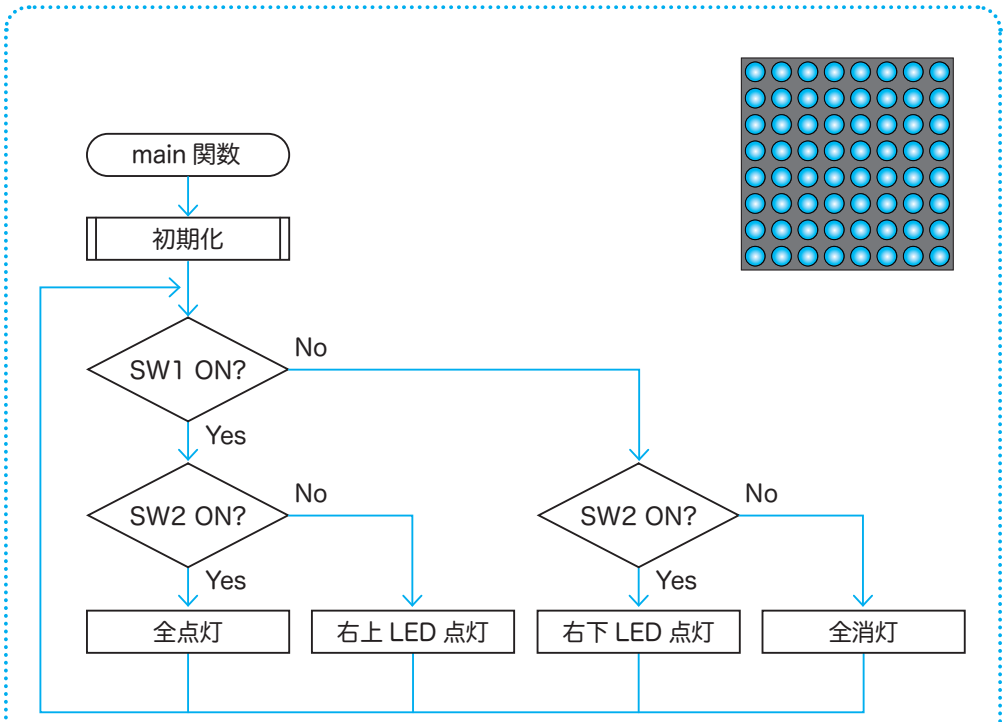


図 10-2 「入れ子」を使ったプログラムのフローチャート

では、このフローチャートを基にプログラム 10-2 を組みましょう。

STEP 10 SW を使ってドットマトリクス LED を点灯させる

入れ子を使ったプログラム 10-2 を組みましょう。main 関数の空欄を埋めてプログラムを完成させてください。

プログラム　10-2

```
/*
 *  main 関数
 */
int main(void)
{
    initIO(); // 初期化関数の呼び出し

    // 永久ループ
    while (1)
    {
        if (                          )    // SW1 が ON で
        {
            if (                          )    // SW2 が ON ならば
            {
                P4.DR.BYTE = 0xFF; // 全点灯
                PB.DR.BYTE = 0x00; // 0000 0000
            }
            else                // SW2 が OFF ならば
            {
                P4.DR.BYTE = 0x80; // 右上点灯
                PB.DR.BYTE = 0xFE; // 1111 1110
            }
        }
        else                // SW1 が OFF で
        {
            if (                          )    // SW2 が ON ならば
            {
                P4.DR.BYTE = 0x01; // 右下点灯
                PB.DR.BYTE = 0xFE; // 1111 1110
            }
            else                // SW2 が OFF ならば
            {
                P4.DR.BYTE = 0x00; // 全消灯
                PB.DR.BYTE = 0xFF; // 1111 1111
            }
        }
    }

    return 0;
}
```

解答例はページ下

```
if (P8.DR.BIT.B0 == 0) // SW1 が ON で
if (P8.DR.BIT.B1 == 0) // SW2 が ON ならば
if (P8.DR.BIT.B1 == 0) // SW2 が OFF ならば
```

10.3　論理演算子を用いたフローチャート (パターン 2)

図 10-3 は論理演算子を用いた場合のフローチャートになります。
図中のひし形の中を見てください。例えば、一番左の条件分岐では、条件が「SW1 が ON かつ SW2 が OFF」と、２つあります。このように、複数の処理を扱う時は論理演算子を使います。

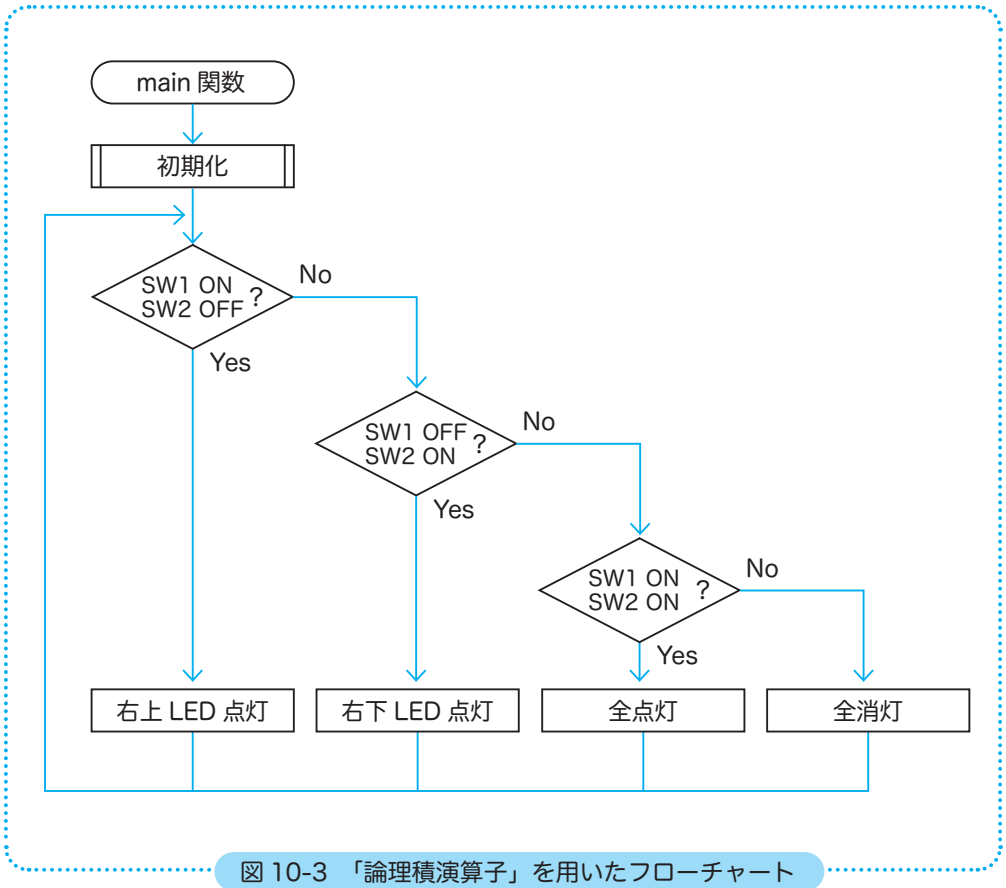


図 10-3 「論理積演算子」を用いたフローチャート

STEP 10 SW を使ってドットマトリクス LED を点灯させる

【 論理演算子 】

論理演算子には以下のような種類があります。STEP05 で学習した否定演算子も論理演算子の 1 種です。今回のフローチャートでは「論理積」が使えるでしょう。

名前	記号	演算子	読み	記述例	意味
論理積	AND	&&	かつ	a && b	a と b が共に真の場合に真
論理和	OR	||	または	a || b	a か b の少なくとも 1 つが真の場合に真
否定	NOT	!	～でない	! a	a が真の時に偽、偽の時に真

```
if( 条件式 1 && 条件式 2)
    {
        処理
    }
```

条件式 1 かつ条件式 2 が満たされた時、ブロック内の処理を行います。

例 1）i の値が 0 以上かつ 100 以下のときブロック内の処理をする。
※ C 言語では、以上 (≦) を「<=」、以下 (≧) を「>=」と表記します。

```
if( i >= 0 && i <= 100 )
{
        処理
}
```

例 2）i の値が 0 かつ j の値が 100 のときブロック内の処理をする。

```
if( i == 0 && j == 100 )
{
        処理
}
```

論理積演算子を用いたプログラム 10-3 を組みましょう。main 関数の空欄を埋めてプログラムを完成させてください。

プログラム　10-3

```
/*
 *  main 関数
 */
int main(void)
{
    initIO(); // 初期化関数の呼び出し

    // 永久ループ
    while (1)
    {
        // SW1 が ON かつ SW2 が OFF ならば、右上点灯　－①

        if (                                    )
        {
            P4.DR.BYTE = 0x80;
            PB.DR.BYTE = 0xFE;
        }
        // SW1 が OFF かつ SW2 が ON ならば、右下点灯　－②

        else if (                               )
        {
            P4.DR.BYTE = 0x01;
            PB.DR.BYTE = 0xFE;
        }
        // SW1 が ON かつ SW2 が ON ならば、全点灯　－③

        else if (                               )
        {
            P4.DR.BYTE = 0xFF;
            PB.DR.BYTE = 0x00;
        }
        // その他ならば、全消灯　－④
        else
        {
            P4.DR.BYTE = 0x00;
            PB.DR.BYTE = 0xFF;
        }
    }

    return 0;
}
```

解答例はページ下

```
if (P8.DR.BIT.B0 == 0 && P8.DR.BIT.B1 == 1)  // SW1 が ON かつ SW2 が OFF ならば、右上点灯－①
else if (P8.DR.BIT.B0 == 1 && P8.DR.BIT.B1 == 0)  // SW1 が OFF かつ SW2 が ON ならば、右下点灯－②
else if (P8.DR.BIT.B0 == 0 && P8.DR.BIT.B1 == 0)  // SW1 が ON かつ SW2 が ON ならば、全点灯　－③
```

STEP 10 SW を使ってドットマトリクス LED を点灯させる

プログラム 10-2、10-3 を組んでいると、SW1 と SW2 のどちらのことを組もうとしているのか、ごちゃごちゃになってきませんでしたか？
SW1 が ON なら P8.DR.BIT.B0 == 0、SW2 が ON なら P8.DR.BIT.B1 == 0 としていると、ぱっと見た時どちらの SW のことなのか分かりません。たとえプログラムを組んだ当人が分かっていたとしても、他人から見て分かりにくいプログラムというのは好ましくありません。

そこで、マクロを用いてプログラムを組み替えたいと思います。
まずは、マクロがどういうものなのか学習していきましょう。

【 マクロ 】

プログラム中の文字列を、あらかじめ定義された規則に従って置換することをマクロと言います。このマクロを用いれば、文字列を好きな言葉 (マクロ名) に置き換えることができます。

```
#define  マクロ名  値
```

※ 行末に「 ; 」は不要です。注意してください。

値には式を入れることもできます。その場合は (　) を付けましょう。(　) を付けるのは演算子に優先順位があり、(　) を付けないと思わぬ結果になる可能性があるからです。演算子の優先順位とは、「＋ － × ÷」に優先度があるのと同じです。算数の授業を思い出して見てください。

5 ＋ 1 × 3

この数式の答えが 18 ではなく 8 になるのは、演算子に優先度があるからですね。答えを 18 にしたいのなら、

(5 ＋ 1) × 3

と、表記します。これと同じことです。

では、プログラム 10-3 をマクロを用いて組んだ記述例を示します。if 文の条件式が分かり易くなりますね。

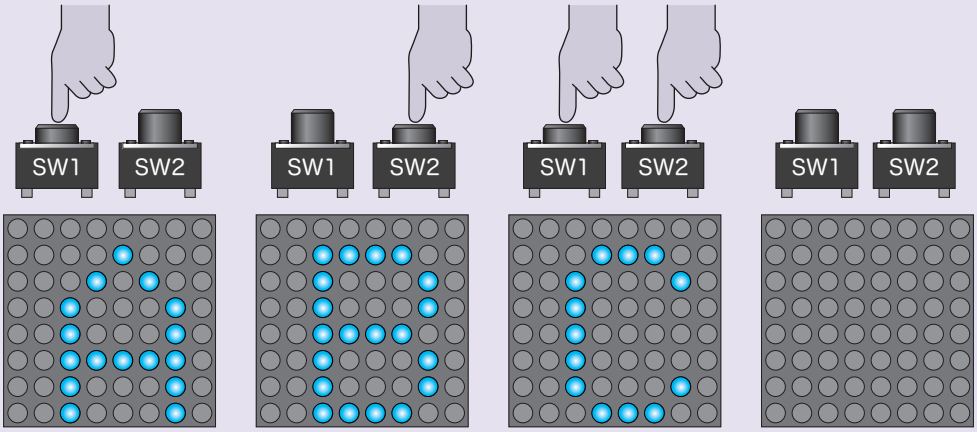
プログラム例　10-4

```
05  #include <3052f.h>                    // 3052F 固有の定数

06  #define SW1_ON (P8.DR.BIT.B0 == 0)    // SW1 が ON の時
07  #define SW1_OFF (P8.DR.BIT.B0 == 1)   // SW1 が OFF の時
08  #define SW2_ON (P8.DR.BIT.B1 == 0)    // SW2 が ON の時
09  #define SW2_OFF (P8.DR.BIT.B1 == 1)   // SW2 が OFF の時

        中略

42  /*
43   *  main 関数
44   */
45  int main(void)
46  {
47      initIO(); // 初期化関数の呼び出し
48
49      int index = 0; // 配列に使う添え字の変数
50
51      // 永久ループ
52      while (1)
53      {
54          // SW1 が ON かつ SW2 が OFF ならば、右上点灯　－①
55          if (SW1_ON && SW2_OFF)
56          {
57              P4.DR.BYTE = 0x80;
58              PB.DR.BYTE = 0xFE;
59          }
60          // SW1 が OFF かつ SW2 が ON ならば、右下点灯　－②
61          else if (SW1_OFF && SW2_ON)
62          {
63              P4.DR.BYTE = 0x01;
64              PB.DR.BYTE = 0xFE;
65          }
66          // SW1 が ON かつ SW2 が ON ならば、全点灯　－③
67          else if (SW1_ON && SW2_ON)
68          {
69              P4.DR.BYTE = 0xFF;
70              PB.DR.BYTE = 0x00;
71          }
72          // その他ならば、全消灯　－④
73          else
74          {
75              P4.DR.BYTE = 0x00;
76              PB.DR.BYTE = 0xFF;
77          }
78      }
79
80      return 0;
81  }
```

SW を使ってドットマトリクス LED を点灯させる

課題　10-6

以下の条件を満たすプログラム 10-6 を組みなさい。
SW1 を押している間は「A」と点灯する。
SW2 を押している間は「B」と点灯する。
SW1、SW2 を押している間は「C」と点灯する。
SW1、SW2 を押していない状態では消灯する。



フローチャートは図 10-3 とほとんど変わりません。LED の点灯パターンを変えるだけです。
しかし、各点灯パターンはダイナミック点灯でプログラムする必要があります。
ダイナミック点灯は STEP07 で学習しましたね。

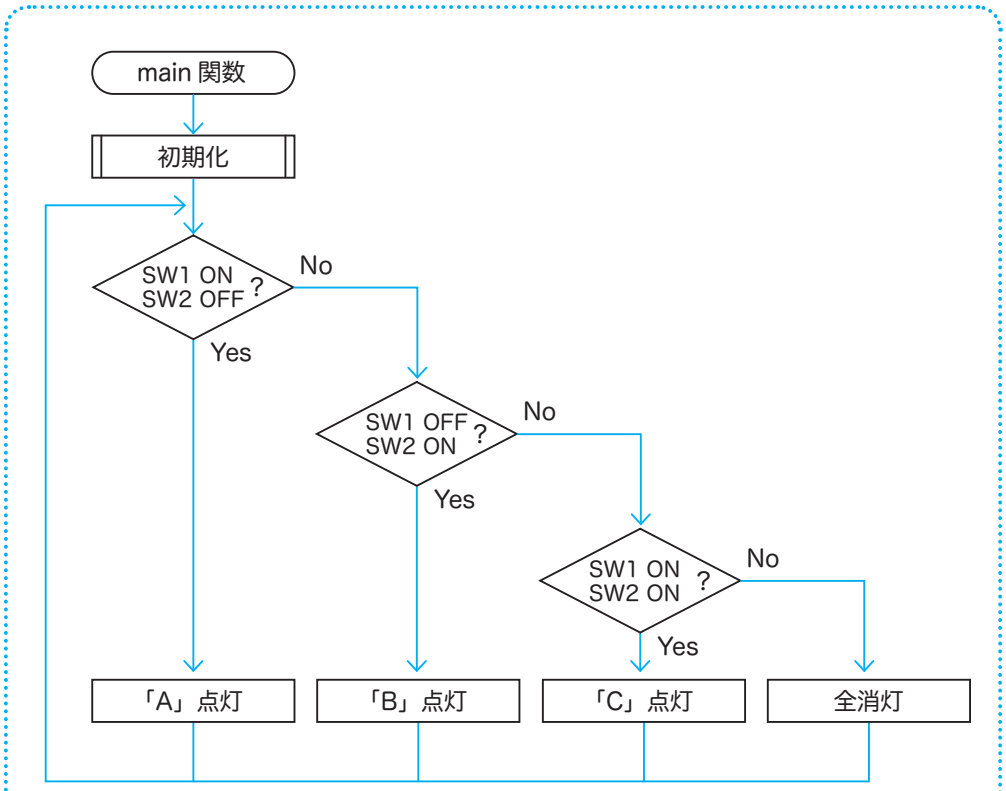


図 10-4　ダイナミック点灯文字フローチャート

STEP 10 SW を使ってドットマトリクス LED を点灯させる

プログラム 10-6 を組みましょう。
「プログラム 7-2」のダイナミック点灯に関する部分を抜粋して掲載しておきます。

プログラム　7-2　一部抜粋

```
// 横点灯行の配列
int a_p4[8] = {0x80, 0x40, 0x20, 0x10, 0x08, 0x04, 0x02, 0x01};

// 点灯パターン A の配列
int a_pb_A[8] = {0xFF, 0xF7, 0xEB, 0xDD, 0xDD, 0xC1, 0xDD, 0xDD};


/*
 *  main 関数
 */
int main(void)
{
    initIO(); // 初期化関数の呼び出し

    // 永久ループ
    while (1)
    {
        int index = 0; // 両配列に使う添え字の変数

        for (index = 0; index < 8; index++)
        {
            P4.DR.BYTE = a_p4[index];
            PB.DR.BYTE = a_pb[index];

            waitMs(1); // 待ち時間関数の呼び出し
        }
    }

    return 0;
}
```

点灯パターンの配列例です。参考にしてください。

・「A」の配列　　{ 0xFF, 0xF7, 0xEB, 0xDD, 0xDD, 0xC1, 0xDD, 0xDD };
・「B」の配列　　{ 0xFF, 0xC3, 0xDD, 0xDD, 0xC3, 0xDD, 0xDD, 0xC3 };
・「C」の配列　　{ 0xFF, 0xE3, 0xDD, 0xDF, 0xDF, 0xDF, 0xDD, 0xE3 };