```
;==========================================================================
;  PICマイコンとマトリックスLEDを使った視覚支援タイマー（MPASM用）
;                                  version 2.54 release candidate (2009/2/11)
;                                                (C) Copyright by SoraPapa
;                                                soramame@kids.email.ne.jp
;
;  20ピンPICマイコンと秋月電子の8x8ドットマトリックスLEDを活用して、時間管理
;  を視覚支援する電子タイマーを製作する。
;  PICマイコンには5Vの標準電源を供給し、I/Oピン18本のうち2本をタクトSWと圧電
;  ブザーに接続、それ以外の16ピンをマトリックスLEDに直結して駆動する。
;
;  起動すると、最初に起動音とともに設定時間が表示され(1.5秒間)、以後は時間の
;  経過とともにLEDが消灯、全部消えると「0」が表示されてアラームが鳴る仕組み。
;  タイマー設定時間はボタンを押すことで変更でき、1分～5分は1分刻み、、以後は
;  5分刻みで最大120分まで設定できる。
;
;  標準の設定では、3分までは3秒ごとにLEDが１つ消える表示、4分から40分までは
;  LEDの左5列を使って1分ごとにLEDが１つ消える表示、45分から60分まではLEDを8列
;  すべて使って1分ごとにLEDが１つ消える表示、60分超についてはその部分だけ15分
;  ごとにLEDが１つ消える表示方法となる（表示設定は変更可能）。
;
;  タクトスイッチを押しながら起動すると、動作モード設定画面になる。
;  動作モード設定画面起動時には、左４列でプログラムのバージョンが表示され、
;  右３列でタイマーを作動させた累積時間が2進24ビット表記で表示される。
;
;  最初の設定画面はLEDの表示方法。左から順に、4分以下の表示、4分超40分以下の
;  表示、40分超の表示の方法を視覚的に表現している。
;  (1) 4分以下の表示：LED点灯数を増やして3秒に1LEDが消える設定（設定画面では
;      4列のLEDが全部点灯）または分数どおりのLEDが点灯する設定（設定画面では
;      3個のLEDが点灯）を選択できる。
;  (2) 4分超40分以下の表示：マトリックスLEDの左の5列だけを使って表示する設定
;      （設定画面では5個のLEDが点灯）または8列全部を使って表示する設定（設定
;      画面では8個のLEDが点灯）を選択できる。
;  (3) 40分超の表示：マトリックスLEDの左の5列だけを使って表示し、2分または3分
;      につき１つのLEDが点灯する設定（設定画面では5個のLEDが点灯）または8列全
;      部で1分につき１つのLEDが点灯、ただし60分超の部分は15分につき１つのLED
;      が点灯する設定（設定画面では8個のLEDが点灯）を選択できる。
;  (4) フルLEDモード：常にLED全点灯（64個）から始まるモード。1LEDにつき、分数
;      と等しい秒数が割り当てられる（例：35分なら1LED＝35秒）。すると実際は60
;      LEDで済むので、LEDの7・23・39・55個めはダミー点灯となっていて、例えば
;      56個めのLEDが消えると、55個めは即座に消えて54個めが点滅するといった挙
;      動をとる（設定画面では全LEDが点灯）。
;  (5) フルLED・リバースモード：常に全消灯から始まり、時間ぴったりで全点灯と
;      なる、フルLEDモードの逆挙動をとるモード（設定画面では、Rの反転表示）。
;  (6) 砂時計モード：上半分は8列すべて使った通常点灯だが、点灯LEDが砂時計のよ
;      うに落下し、下にたまっていくモード。1LEDあたりの時間が1分（設定時間40
;      分以下の場合）、2分（設定時間45～75分の場合）、3分（設定時間90分以上の
;      場合）の変動式となるモード（設定画面では砂時計の表示）と、常に1LEDあた
;      り3分で固定されるモード（設定画面では砂時計に「F」が表示）がある。
;  (7) デジタルモード：分数が数字で表示されるモード。設定時間からカウントダウ
;      ンする通常モード（設定画面では「15」の表示）と、設定時間までカウントア
;      ップするリバースモード（設定画面では「15」の反転表示）がある。
;  (8) アナログタイマーモード：マトリックスLEDの外周部分のLEDが時計回りに消灯
;      していくモード。1LEDあたり3分。60分以内での使用を推奨。75分以上の設定
;      では内周も使うので見づらくなる（設定画面では、○表示）
;  (9) トーテムポールモード：４本のポールが伸び縮みして残り時間を表示する。左
;      ２本が分、右２本が秒をあらわしており、それぞれ８進数表現となっている。
;      （設定画面では、４本ポールで表示）
;
;  最初の設定画面でボタン操作をせずに5秒待つとOKの文字が表示され、次の設定画
;  面に移る。ここではアラーム音とクリック音の有無、LEDの点滅等を設定する。
;  (10) 左上の「C」がクリック音の設定で、表示されていれば音あり。
;  (11) 左下の「A」がアラーム音の設定で、表示されていれば音あり。
;  アラーム音は起動時と設定時間経過時に、クリック音はボタン操作時と1分ごとの
;  経過時に発せられる。
;  (12) 右上の「D」はデジタル時間表示の設定（１分おきに残り時間を表示するかど
;       うか）で、「D」が表示されていればデジタル時間を表示する。
;  (13) 右下の「b」はLED点滅の設定を表しており、「b」の文字が点滅していれば点
;       滅あり、点灯したままなら点滅なしの設定となる。
;
;  この画面で5秒操作しないでおくとOKの文字が表示され、第３の設定画面に移る。
;  第３の設定は、起動時のタイマー設定標準時間の設定となっている。
;  (14) 数字が表示されるので、希望のタイマー標準時間（分）に設定する。
;
;  この画面で5秒操作しないでおくとOKの文字が表示され、第４の設定画面に移る。
;  第４の設定は、内蔵タイマーの速さの微調整が行なえる。
;  (15) 0,プラスまたはマイナスとLEDによるバーが表示される。プラスはタイマーを
;       早め、マイナスは遅くする。LED1個につき1分あたり約0.5秒調整される。
;
;  さらにこの画面で5秒ボタン操作しないと、OKの文字が表示され、(1)～(15)の設定
;  がこの時点で書き込まれる。（それまでに電源を切ると設定されない）
;
;  設定時間経過後、設定モード終了後はボタン待機状態になり、ボタンを押すとタイ
;  マーが再開する。待機状態では若干の電力を消費するので、使用後は電源を切る。
;
;  入出力設定は以下のとおり。（すべてデジタル入出力扱い）
;      タクトスイッチ                  : A(3)  （プルアップ接続によりON=0）
;      圧電ブザー                      : A(5)  （他励式）
;      マトリックスLED（8x8, カソードコモン）
;          DATAポート(0-7,アノード)    : C(0) - C(7)
;          SCANポート(0-7,カソード)    : A(0) - A(2), A(4), B(4) - B(7)
;==========================================================================
;----------------------------------------------------------
;  コンフィギュレーション
;----------------------------------------------------------
    list    p = PIC16F689   ; PIC16F689を使う。多分685、690でも動作するはず
    include "p16f689.inc"
    __config _FCMEN_OFF & _IESO_OFF & _BOR_OFF & _MCLRE_OFF & _PWRTE_ON & _WDT_OFF
 & _INTRC_OSC_NOCLKOUT
    errorlevel -302         ; BANK切り替えについてのWarningを抑制

;----------------------------------------------------------
;  グローバル変数
;----------------------------------------------------------
LED_DATA0       equ 020h    ; LED表示データアクセス用（直接＆間接）
LED_DATA1       equ 021h
LED_DATA2       equ 022h
LED_DATA3       equ 023h
LED_DATA4       equ 024h
LED_DATA5       equ 025h
LED_DATA6       equ 026h
LED_DATA7       equ 027h
LED_BUF0        equ 028h    ; LED表示データ退避用
LED_BUF1        equ 029h
LED_BUF2        equ 02ah
LED_BUF3        equ 02bh
LED_BUF4        equ 02ch
LED_BUF5        equ 02dh
LED_BUF6        equ 02eh
LED_BUF7        equ 02fh

MINUTE          equ 030h    ; 残り時間（分）
SECOND2         equ 031h    ; 残り時間（秒×２）0.5秒でデクリメント
SEC2_CHANGE     equ 032h    ; (0)：1/2秒が経過したことを示すフラグ
                            ; (1)：セットすると1回だけフラグセットスキップ
BTN_STAT        equ 033h    ; (0)：スイッチのリアルタイム状態
                            ; (1)：タクトスイッチ押された情報
GRA_PTR         equ 034h    ; グラフィック表示用ポインタ
MIN_ZONE        equ 035h    ; (2) 設定した時間が80分超であるというフラグ
                            ; (1) 設定した時間が3分以下であるというフラグ
                            ; (0) 設定した時間が40分超であるというフラグ
MIN_LED         equ 036h    ; フルLEDモードのときのLED点灯数
MIN_COUNT       equ 037h    ; フルLEDモードのときのカウントダウンタイマー
MIN_COUNT_D     equ 038h    ; フルLEDモードのときのカウントダウン初期値
MIN_INI         equ 039h    ; セットされた分数
DIV_OPERAND     equ 03ah    ; 割り算の「割る数」
DIV_MOD         equ 03bh    ; 割り算の「余り」
WK1             equ 03ch    ; 汎用ワークレジスタ
WK2             equ 03dh    ; 汎用ワークレジスタ
WK3             equ 03eh    ; 汎用ワークレジスタ
WK4             equ 03fh    ; 汎用ワークレジスタ

                            ; 以下の変数はBANKを切り替えてもアクセスできる
OPTION_MODE     equ 070h    ; (7) デジタル時間表示フラグ(1:表示する)
                            ; (6) LED点滅しないフラグ(1:点滅しない)
                            ; (5) アラーム音消しフラグ(1:消す)
                            ; (4) クリック音消しフラグ(1:消す)
                            ; (3)～(0) 点灯モード詳細
                            ; (3)がオフの場合は以下のとおり
                            ;   (2) 40分超モード（1:５進表示）
                            ;   (1) 5～40分モード（1:８進表示）
                            ;   (0) 3分以内モード（1:点灯数増やさない）
                            ; (3)がオンの場合は以下のとおり
                            ;   [000] フルLEDモード
                            ;   [001] フルLED・リバースモード
                            ;   [010] 砂時計モード・表示可変
                            ;   [011] 砂時計モード・表示３分固定
                            ;   [100] デジタルモード
                            ;   [101] デジタル・リバースモード
                            ;   [110] アナログタイマーモード
                            ;   [111] トーテムポールモード
MIN_DEFAULT     equ 071h    ; デフォルトタイマー時間（0-19で保存）
OSC_ADJUST      equ 072h    ; タイマー周波数（OSCTUNE）微調整
HIST_CNT1       equ 073h    ; 累積アラーム回数カウンタ 下位バイト
HIST_CNT2       equ 074h    ; 累積アラーム回数カウンタ 中位バイト
HIST_CNT3       equ 075h    ; 累積アラーム回数カウンタ 上位バイト

;----------------------------------------------------------
;  割り込みルーチン用変数
;----------------------------------------------------------
BKUP_W          equ 040h    ; 割り込み時のバックアップ（Wレジスタ）
BKUP_STATUS     equ 041h    ; 割り込み時のバックアップ（STATUSレジスタ）
BKUP_FSR        equ 042h    ; 割り込み時のバックアップ（FSRレジスタ）
BKUP_PCLATH     equ 043h    ; 割り込み時のバックアップ（PCLATHレジスタ）
SECTIMER        equ 044h    ; 0.5秒カウント用変数
SECTIMER_LEAP   equ 045h    ; うるう秒カウント用変数
CHATTER_CNT     equ 046h    ; チャタリング回避カウンタ
BUZZER_STAT     equ 047h    ; ブザー（ポートA5）のON/OFF状態
ROW_COUNT       equ 048h    ; ダイナミック表示で何行目を表示しているか
SCAN_PORT       equ 049h    ; スキャンポートの開きかた
WK_SCAN         equ 04ah    ; スキャンポート計算用

;----------------------------------------------------------
;  その他サブルーチン用変数
;----------------------------------------------------------
BEEP_CNT        equ 04bh    ; BEEPルーチン作業用カウンタ
WAIT_CNT1       equ 04ch    ; WAITルーチン作業用カウンタ１
WAIT_CNT2       equ 04dh    ; WAITルーチン作業用カウンタ２
WK_CALCDIV1     equ 04eh    ; CALC_DIVルーチン作業用変数１
WK_CALCDIV2     equ 04fh    ; CALC_DIVルーチン作業用変数２
WK_SHOWNUM      equ 050h    ; 数字表示ルーチン作業用変数
SHOWNUM_CNT     equ 051h    ; 数字表示ルーチン作業用カウンタ
SHOWNUM_PTR     equ 052h    ; 数字表示ルーチン用ポインタ
SHOWGRA_CNT     equ 053h    ; SHOW_GRAPHICルーチン作業用カウンタ

SET_MODE_WAIT   equ 058h    ; SET_MODEルーチン時間待ちカウンタ
MINUTE_LED      equ 059h    ; LED表現上の分数
LED_BASE        equ 05ah    ; LED表示パターン（端数以外の部分）
LED_MOD         equ 05bh    ; LED表示パターン（端数部分）
LED_PTR         equ 05ch    ; LED表示パターン（表示位置）
HG_DIVOPE       equ 060h    ; 砂時計モード用「MINUTEを割る数」
HG_LEDBASE      equ 061h    ; 砂時計モード用「設定分数ベースパターン」
HG_MOD          equ 062h    ; 砂時計モード用「設定分数LED端数」
HG_MOD_ADDR     equ 063h    ; 砂時計モード用「砂落下スタートアドレス」
HG_GAP          equ 064h    ; 砂時計モード用「砂落下間隔ベース値」
HG_MINDIV       equ 065h    ; 砂時計モード用「経過済LED行数」
HG_MINMOD       equ 066h    ; 砂時計モード用「経過済LED端数」
WK_HG1          equ 067h    ; 砂時計モード作業用変数１
WK_HG2          equ 068h    ; 砂時計モード作業用変数２
WK_HG3          equ 069h    ; 砂時計モード作業用変数３

;----------------------------------------------------------
;  マクロ定義・定数定義
;----------------------------------------------------------
DECMIN_CHECK    macro
    decf    MINUTE_LED, f
    btfsc   STATUS, Z
    goto    ANALOG_FINISH
    endm

    #define RELEASE         ; デバッグ中はこの行をコメントアウト

    ifdef   RELEASE
SECTIMER_CONST  equ d'244'  ; 割り込みタイマーのカウンタ数
    else
SECTIMER_CONST  equ d'80'   ; カウンタ数（デバッグ用６倍速）
    endif

;----------------------------------------------------------
;  EEPROMデータ
;----------------------------------------------------------
    org     h'2100'
    ifdef   RELEASE         ; リリース用のEEPROM設定
        de  b'00000000'     ; OPTION_MODE：すべて標準設定
        de  d'1'            ; MIN_DEFAULT：１分
        de  h'00'           ; OSC_ADJUST：調整なし
        de  h'00'           ; HIST_CNT1：ゼロ
        de  h'00'           ; HIST_CNT2：ゼロ
        de  h'00'           ; HIST_CNT3：ゼロ
    else                    ; デバッグ用のEEPROM設定
        de  b'00001111'     ; OPTION_MODE
        de  d'1'            ; MIN_DEFAULT
        de  h'00'           ; OSC_ADJUST
        de  h'0fe'          ; HIST_CNT1：254
        de  h'0fe'          ; HIST_CNT2：254
        de  h'00'           ; HIST_CNT3：ゼロ
    endif

;----------------------------------------------------------
;  割り込みジャンプ先設定
;----------------------------------------------------------
    org     00h
    goto    INIT            ; 初期設定→メインループへ
    org     04h
    goto    INTERRUPT       ; 割り込み用ルーチンの設定

;----------------------------------------------------------
;  初期設定
;----------------------------------------------------------
INIT
    bcf     INTCON, GIE     ; 割込み禁止
    bcf     STATUS, RP0     ; BANK2に切り替え
    bsf     STATUS, RP1
    clrf    ANSEL           ; アナログ入出力禁止
    clrf    ANSELH
    bsf     STATUS, RP0     ; BANK1に切り替え
    bcf     STATUS, RP1
    movlw   b'11001000'
    movwf   TRISA           ; ポートA3のみ入力、あとは出力
    clrf    TRISB           ; ポートBはすべて出力
    clrf    TRISC           ; ポートCはすべて出力
    bsf     IOCA, IOCA3     ; タクトスイッチ(RA3)にInterrupt on change設定
                            ; (RABIEをセットしなければ実際には起動しない)
    movlw   b'10000010'     ; Timer0のプリスケーラの設定（1:8)=488.28125Hz
    movwf   OPTION_REG      ; うるう秒を設定することで488.25Hzとして扱う。
                            ; 誤差0.0064%(120分で0.5秒)はクロック誤差未満
    bcf     STATUS, RP0     ; BANK0に切り替え

;----------------------------
INIT2                       ; SLEEP復帰時はここに飛んでくる
    clrf    PORTC           ; PORT 初期設定
    clrf    PORTA
```

```
    clrf    PORTB
    clrf    ROW_COUNT       ; LED ROWカウンタクリア
    call    CLEAR_LED_DATA  ; LEDデータを空にする
    clrf    MIN_ZONE
    clrf    SECOND2
    call    RESET_TIMER     ; 割り込みタイマ用の変数をリセット
    clrf    CHATTER_CNT
    clrf    BTN_STAT
    bsf     INTCON, TOIE    ; Timer0の割り込みを許可する
    bsf     INTCON, GIE     ; 割り込み許可

    btfss   PORTA, 3        ; タクトスイッチがON (RA(3)=Low) の場合、
    goto    SET_MODE        ; 設定モードに移行

;-----------------------------------------------------------
;   メインループ
;-----------------------------------------------------------
    call    READ_EEPROM     ; EEPROMに格納された値を読む
    call    SET_OSCTUNE     ; OSCTUNEを設定
    movf    MIN_DEFAULT, W  ; MIN_DEFAULTを読み出し
    movwf   MINUTE          ; その値をMINUTEにセット
    movwf   MIN_INI         ; MIN_INIにも代入する
    call    CHECK_TIMEZONE  ; 設定時間のレンジのフラグセット
    movf    MIN_DEFAULT, W
    call    SHOW_NUMBER     ; 残り時間表示
    bsf     SEC2_CHANGE, 1  ; 残り時間を１秒間表示する
    call    BEEP            ; ピッと音を出す（約0.1秒）

MAINLOOP
    btfsc   SEC2_CHANGE, 0  ; 0.5秒経過をチェック
    goto    CHANGE_SECOND

    btfss   BTN_STAT, 1     ; タクトスイッチ押された？
    goto    MAINLOOP        ; 押されていなければループ

    call    SET_TIME        ; 押されていたら時間設定ルーチンへ
    movf    MIN_INI, W      ; MIN_INIの値をWにセットして
    call    SHOW_NUMBER     ; 残り時間表示
    bsf     SEC2_CHANGE, 1  ; 残り時間を１秒間表示する
    bcf     BTN_STAT, 1     ; タクトスイッチ押されたフラグを消す
    call    TICK_SOUND
    goto    MAINLOOP        ; 表示をアップデートしない

;-------------------------
CHANGE_SECOND               ; 秒のデクリメント
    bcf     SEC2_CHANGE, 0
    movf    SECOND2, f
    btfsc   STATUS, Z
    goto    CHANGE_MINUTE   ; 0秒だったときは分のデクリメントへ

    decf    SECOND2, f      ; １秒デクリメント
    movf    SECOND2, f      ; ゼロ秒になったかチェック
    btfss   STATUS, Z
    goto    LED_UPDATE

    movf    MINUTE, f       ; 最後の１回はカチ音出さない
    btfss   STATUS, Z
    call    TICK_SOUND      ; カチ音を出す

    goto    LED_UPDATE

CHANGE_MINUTE               ; 分のデクリメント
    movf    MINUTE, f
    btfsc   STATUS, Z
    goto    FINISH          ; 0分だったときは終了処理へ

    decf    MINUTE, f       ; そうでない通常のときは分をデクリメントして
    movlw   d'119'
    movwf   SECOND2         ; SECOND2には119を設定
    btfss   OPTION_MODE, 7  ; 設定をチェック
    goto    LED_UPDATE      ; 時間表示フラグがオフなら先へ

    movlw   b'00001011'     ; 時間表示フラグがオンの場合
    andwf   OPTION_MODE, W  ; OPTION_MODEの(3)(1)(0)ビットだけを抽出
    xorlw   b'00001001'     ; (3)(1)(0)が1,0,1かどうかをチェック(xxxx1x01)
    btfss   STATUS, Z       ; ゼロならリバースモード（フルLED／デジタル）
    goto    CHANGE_MINUTE2

    movf    MINUTE, W
    addlw   d'1'
    subwf   MIN_INI, W      ; リバースモードなら W ← MIN_INI - (MINUTE+1)
    goto    DIGI_UPDATE

CHANGE_MINUTE2
    movf    MINUTE, W
    addlw   d'1'            ; リバースモード以外なら W ←MINUTE + 1

DIGI_UPDATE
    call    SHOW_NUMBER     ; デジタル表示する
    call    CHECK_FULL_LED
    goto    MAINLOOP        ; 戻る

LED_UPDATE
    call    CHECK_FULL_LED
    call    SET_LED_DATA    ; LED表示パターンをセット
    goto    MAINLOOP        ; 戻る

;-------------------------
FINISH
    movf    MIN_INI, W
    addwf   HIST_CNT1, f    ; タイマー分数を加算
    movlw   d'1'
    btfsc   STATUS, C       ; 繰り上がりをチェック
    addwf   HIST_CNT2, f    ; 繰り上がりがあればHIST_CNT2をインクリメント

    btfsc   STATUS, C       ; 繰り上がりをチェック
    incf    HIST_CNT3, f    ; 繰り上がりがあればHIST_CNT3をインクリメント

    call    WRITE_EEPROM    ; 累積タイマー時間をEEPROMに書き込む
    call    ALARM           ; アラームを鳴らす

FINISH2
    bcf     INTCON, TOIE    ; Timer0の割り込みを禁止する
    bcf     INTCON, GIE     ; 割込み禁止
    movlw   b'11111111'     ; PORTを安定状態へ
    movwf   PORTB
    movf    PORTA, W
    iorlw   b'11010111'
    movwf   PORTA
    clrf    PORTC
    btfss   PORTA, 3        ; タクトスイッチが離されるのを待つ
    goto    $-1

    movlw   d'50'
    call    WAIT_NMS        ; チャタリング分（50ms）待ち
    movf    PORTA, W        ; タクトスイッチの最終状態を読む
    bsf     INTCON, RABIE   ; Interrupt on changeを許可
    bcf     INTCON, RABIF   ; Interrupt on change割り込みフラグクリア
    sleep                   ; スリープモードへ

; ここから下はタクトスイッチでWake Upした後の処理
    bcf     INTCON, RABIE   ; Interrupt on changeを禁止
    bcf     INTCON, RABIF   ; Interrupt on change割り込みフラグクリア
    btfss   PORTA, 3        ; タクトスイッチが離されるのを待つ
    goto    $-1

    movlw   d'40'
    call    WAIT_NMS        ; チャタリング分（40ms）待ち
    goto    INIT2           ; INIT2から再度初期設定

;-------------------------
CHECK_FULL_LED              ; 秒変化時のフルLEDモードのときの追加処理
    movlw   b'00001110'
    andwf   OPTION_MODE, W  ; OPTION_MODEの[321]ビットを抽出
    xorlw   b'00001000'     ; [321]=b100でXORをとってみる
    btfss   STATUS, Z       ; これでゼロならフルLEDモード
    return                  ; フルLEDモード以外のときは戻る

    decf    MIN_COUNT, f    ; MIN_COUNTをデクリメント
    btfss   STATUS, Z
    return                  ; ゼロでない場合は戻る

    movf    MIN_COUNT_D, W
    movwf   MIN_COUNT       ; カウンタに数字を再セット
    movf    MIN_LED, f      ; MIN_LEDがゼロかどうかチェック
    btfsc   STATUS, Z
    return                  ; MIN_LEDがゼロなら何もせず戻る

    decf    MIN_LED, f      ; 表示LED数を減らす
    movlw   b'00001111'
    andwf   MIN_LED, W      ; まずMIN_LEDの下位4ビットを取り出して
    xorlw   b'00000111'     ; 07h(00000111_b)と比較する
    btfsc   STATUS, Z       ; 下位4ビットが07hならゼロになる
    decf    MIN_LED, f      ; MIN_LEDをさらにマイナス
    return

;-------------------------
SET_OSCTUNE                 ; OSCTUNEの値を設定
    bcf     INTCON, GIE     ; 割込み禁止 (BANK切り替え中の割り込み防止)
    bsf     STATUS, RP0     ; BANK1に切り替え
    bcf     STATUS, RP1
    movf    OSC_ADJUST, W   ; OSC_ADJUSTの値を読み込んで、
    movwf   OSCTUNE         ; OSCTUNEに書き込む
    bcf     STATUS, RP0     ; BANK0に切り替え
    bsf     INTCON, GIE     ; 割り込み許可
    return

;-----------------------------------------------------------
;  設定モード
;-----------------------------------------------------------
SET_MODE
    call    READ_EEPROM     ; EEPROMに格納された値を読む
    call    SET_OSCTUNE;
    movlw   h'08'
    movwf   GRA_PTR
    call    SHOW_GRAPHIC    ; バージョンNOを表示
    movf    HIST_CNT3, W    ; 右３ラインは累積タイマー時間を表示
    movwf   LED_DATA5
    movf    HIST_CNT2, W
    movwf   LED_DATA6
    movf    HIST_CNT1, W
    movwf   LED_DATA7
    call    SET_MODE_RESET  ; キー離すの待ち＆各種の値をリセット
    call    MODE_DISPLAY1

SET_MODE_LOOP1
    btfsc   SEC2_CHANGE, 0  ; 0.5秒経過をチェック
    call    MODE_SEC_CHANGED    ; 経過していたら処理する
    btfsc   SET_MODE_WAIT, 0    ; 5秒放置したかをチェック
    goto    SET_MODE_ALARM  ; 放置されたら次の設定

    btfss   BTN_STAT, 1     ; タクトスイッチ押された？
    goto    SET_MODE_LOOP1  ; 押されていなければループ

    bcf     BTN_STAT, 1     ; ボタン入力ありフラグをクリア
    clrf    SECOND2         ; SECOND2もクリア

    movlw   b'11110000'     ; OPTION_MODEの下位4ビット内でインクリメント
    andwf   OPTION_MODE, W  ; OPTION_MODEの上位4ビットを抽出
    iorlw   b'00001111'     ; 下位4ビットは全部1にして
    incf    OPTION_MODE, f  ; OPTION_MODEをインクリメントしてから
    andwf   OPTION_MODE, f  ; 上位4ビットを元に戻す
    call    MODE_DISPLAY1
    goto    SET_MODE_LOOP1  ; ループに戻る

SET_MODE_ALARM
    call    MODE_SHOW_OK    ; OKの文字を表示
    call    SET_MODE_RESET  ; 各種の値をリセット
    call    MODE_DISPLAY2

SET_MODE_LOOP2
    btfss   SEC2_CHANGE, 0  ; 0.5秒経過をチェック
    goto    SET_MODE_LOOP2b ; 経過していなければスイッチチェック

    call    MODE_SEC_CHANGED    ; 経過していたら、必要な処理をして
    call    MODE_DISPLAY2   ; LEDの点滅があるので表示も更新
    btfsc   SET_MODE_WAIT, 0    ; 5秒放置したかをチェック
    goto    SET_MODE_TIME   ; 放置されたら次の設定

SET_MODE_LOOP2b
    btfss   BTN_STAT, 1     ; タクトスイッチ押された？
    goto    SET_MODE_LOOP2  ; 押されていなければループ

    bcf     BTN_STAT, 1     ; ボタン入力ありフラグをクリア
    clrf    SECOND2         ; SECOND2もクリア
    movlw   b'00010000'
    addwf   OPTION_MODE, f  ; OPTION_MODEの上位ビットに１を足す
    call    MODE_DISPLAY2
    goto    SET_MODE_LOOP2  ; ループに戻る

SET_MODE_TIME
    call    MODE_SHOW_OK    ; OKの文字を表示
    call    SET_MODE_RESET  ; 各種の値をリセット
    movf    MIN_DEFAULT, W  ; MIN_DEFAULTをWに入れて
    call    SHOW_NUMBER     ; 分数を表示

SET_MODE_LOOP3
    btfsc   SEC2_CHANGE, 0  ; 0.5秒経過をチェック
    call    MODE_SEC_CHANGED    ; 経過していたら処理する
    btfsc   SET_MODE_WAIT, 0    ; 5秒放置したかをチェック
    goto    SET_MODE_OSCADJ ; 放置されたら次の設定へ

    btfss   BTN_STAT, 1     ; タクトスイッチ押された？
    goto    SET_MODE_LOOP3  ; 押されていなければループ

    bcf     BTN_STAT, 1     ; ボタン入力ありフラグをクリア
    clrf    SECOND2         ; SECOND2もクリア
    movlw   d'120'
    subwf   MIN_DEFAULT, W  ; MIN_DEFAULT - 120
    btfsc   STATUS, C       ; C=0、つまりボローが発生したらスキップ
    clrf    MIN_DEFAULT     ; ボローなし、つまりMIN_DEFAULT=120なら0にする

    movf    MIN_DEFAULT, W
    movwf   WK1             ; WK1にMIN_DEFAULTをコピー
    movlw   d'5'
    subwf   WK1, f          ; MIN_DEFAULT - 5を計算してみる（Wは５のまま）
    btfss   STATUS, C       ; C=1、つまりボローがなければスキップ
    movlw   d'1'            ; ボローあり、つまりMIN_DEFAULTが4以下ならWを1にする

    addwf   MIN_DEFAULT, f  ; MIN_DEFAULTに５または１を足す
    movf    MIN_DEFAULT, W  ; MIN_DEFAULTをWに入れて
    call    SHOW_NUMBER     ; 分数を表示
    goto    SET_MODE_LOOP3  ; ループに戻る

SET_MODE_OSCADJ
    call    MODE_SHOW_OK    ; OKの文字を表示
    call    SET_MODE_RESET  ; 各種の値をリセット
    call    MODE_DISPLAY4

SET_MODE_LOOP4
    btfsc   SEC2_CHANGE, 0  ; 0.5秒経過をチェック
    call    MODE_SEC_CHANGED    ; 経過していたら処理する
```

```
    btfsc   SET_MODE_WAIT, 0    ; 5秒放置したかをチェック
    goto    SET_MODE_FINISH ; 放置されたら終了処理へ

    btfss   BTN_STAT, 1     ; タクトスイッチ押された？
    goto    SET_MODE_LOOP4  ; 押されていなければループ

    bcf     BTN_STAT, 1     ; ボタン入力ありフラグをクリア
    clrf    SECOND2         ; SECOND2もクリア
    incf    OSC_ADJUST, W   ; OSC_ADJUSTに1を足したうえで、
    andlw   b'00011111'
    movwf   OSC_ADJUST      ; 6ビットに繰り上がったらゼロに戻す
    call    MODE_DISPLAY4
    goto    SET_MODE_LOOP4  ; ループに戻る

SET_MODE_FINISH
    ifdef   RELEASE
        call    WRITE_EEPROM    ; リリース時は最後だけEEPROMに書き込む
    endif
    call    MODE_SHOW_OK    ; OKの文字を表示
    goto    FINISH2         ; SLEEPモードへ

;-----------------------------
SET_MODE_RESET
    btfss   PORTA, 3        ; タクトスイッチが離されるのを待つ
    goto    $-1

    movlw   d'50'
    call    WAIT_NMS        ; チャタリング分（50ms）待ち
    call    RESET_TIMER     ; 割り込みタイマ用の変数をリセット
    clrf    SECOND2
    clrf    SET_MODE_WAIT
    bcf     BTN_STAT, 1     ; スイッチフラグをクリアしておく
    return

;-----------------------------
MODE_SEC_CHANGED
    bcf     SEC2_CHANGE, 0  ; 0.5秒たった場合の処理
    incf    SECOND2, f
    movlw   d'10'
    subwf   SECOND2, W      ; SECOND2 - 10
    btfsc   STATUS, C       ; C=0、つまりボローが発生したらスキップ
    bsf     SET_MODE_WAIT, 0    ; 5秒たったフラグをONにする

    return

;-----------------------------
MODE_SHOW_OK
    ifndef  RELEASE
        call    WRITE_EEPROM    ; デバッグ時のみ毎回EEPROMに書き込む
    endif
    call    BEEP            ; ピッと音を出す（約0.1秒）
    movlw   h'010'
    movwf   GRA_PTR         ; GRA_PTR = 010h（OK）
    call    SHOW_GRAPHIC    ; OKの文字を表示
    movlw   d'250'
    call    WAIT_NMS        ; 250ms時間待ち
    movlw   d'250'
    call    WAIT_NMS        ; 250ms時間待ち
    movlw   d'250'
    call    WAIT_NMS        ; 250ms時間待ち
    movlw   d'250'
    call    WAIT_NMS        ; 250ms時間待ち
    return

;-----------------------------
MODE_DISPLAY1
    btfss   OPTION_MODE, 3  ; 特殊表示モードかどうか判定
    goto    MODE_U3_CHECK   ; 通常表示モードなら先に飛ぶ

    movlw   b'00000111'
    andwf   OPTION_MODE, W  ; OPTION_MODEの下3ビットを抽出
    movwf   WK1             ; ワークレジスタにコピー
    bcf     STATUS, C       ; Cフラグクリア
    rlf     WK1, f
    rlf     WK1, f
    rlf     WK1, f          ; ８倍する
    movlw   h'018'
    addwf   WK1, W          ; ８倍したWK1 + 018h → W
    movwf   GRA_PTR         ; それをGRA_PTRにコピーして
    call    SHOW_GRAPHIC    ; LEDの表示設定
    return

MODE_U3_CHECK               ; 3分以下の表示設定
    call    CLEAR_LED_DATA  ; LEDデータを空にする
    movlw   b'11100000'
    movwf   LED_DATA3
    btfsc   OPTION_MODE, 0  ; ３分以下表示モードのチェック
    goto    MODE_U40_CHECK

    movlw   b'11111111'     ; 標準モードの場合、LEDをたくさん点灯
    movwf   LED_DATA0
    movwf   LED_DATA1
    movwf   LED_DATA2
    movwf   LED_DATA3

MODE_U40_CHECK              ; 5～40分の表示設定
    movlw   b'11111000'     ; 標準モード＝5進モード
    btfsc   OPTION_MODE, 1  ; 設定チェック
    movlw   b'11111111'     ; 8進モード
    movwf   LED_DATA5

MODE_O40_CHECK              ; 40分超の表示設定
    movlw   b'11111111'     ; 標準モード＝8進モード
    btfsc   OPTION_MODE, 2  ; 40分超の設定チェック
    movlw   b'11111000'     ; 5進モード
    movwf   LED_DATA7
    return

;-----------------------------
MODE_DISPLAY2               ; アラーム設定画面の表示
    movlw   h'058'
    movwf   GRA_PTR
    call    SHOW_GRAPHIC    ; まずはAbCDすべての文字を表示

    btfss   OPTION_MODE, 4  ; クリック音の設定チェック
    goto    MODE_ALM_CHECK

    movlw   b'11111000'     ; クリック音なしの場合「C」を消す
    andwf   LED_DATA0, f
    andwf   LED_DATA1, f
    andwf   LED_DATA2, f

MODE_ALM_CHECK
    btfss   OPTION_MODE, 5  ; アラーム音の設定チェック
    goto    MODE_DIGI_CHECK

    movlw   b'00001111'     ; アラーム音なしの場合「A」を消す
    andwf   LED_DATA0, f
    andwf   LED_DATA1, f
    andwf   LED_DATA2, f

MODE_DIGI_CHECK
    btfsc   OPTION_MODE, 7  ; デジタル表示の設定チェック
    goto    MODE_BLINK_CHECK

    movlw   b'11111000'     ; デジタル表示なしの場合「D」を消す
    andwf   LED_DATA4, f
    andwf   LED_DATA5, f
    andwf   LED_DATA6, f

MODE_BLINK_CHECK
    btfsc   OPTION_MODE, 6  ; 点滅の設定をチェック
    return                  ; 点滅なしの場合は戻る

    btfsc   SECOND2, 0      ; SECOND2が偶数かどうかチェック
    return                  ; 奇数の場合は点灯したまま戻る

    movlw   b'00001111'     ; 点滅あり、かつSECOND2が奇数なら「ｂ」を消す
    andwf   LED_DATA4, f
    andwf   LED_DATA5, f
    andwf   LED_DATA6, f
    return

;-----------------------------
MODE_DISPLAY4               ; 内部発信周波数微調整画面の表示
    call    CLEAR_LED_DATA  ; LEDデータを空にする
    movf    OSC_ADJUST, f
    btfss   STATUS, Z       ; OSC_ADJUSTがゼロかどうかチェック
    goto    MODE_DISP4_2    ; ゼロでない場合

    movlw   b'00011100'     ; ゼロの場合「０」を表示するだけ
    movwf   LED_DATA0
    movwf   LED_DATA3
    movlw   b'00100010'
    movwf   LED_DATA1
    movwf   LED_DATA2
    return

MODE_DISP4_2
    movlw   b'00001000'     ; 描けるところだけ先に描いておく
    movwf   LED_DATA0
    movwf   LED_DATA2
    btfsc   OSC_ADJUST, 4   ; 符号をチェック
    goto    MODE_DISP4_MINUS    ; マイナスの場合は先の処理へ

    movlw   b'00011100'     ; プラスの場合
    movwf   LED_DATA1       ; プラスを描く
    movf    OSC_ADJUST, W
    movwf   WK1             ; WK1にOSC_ADJUSTをコピー
    goto    MODE_DISP4_BAR  ; 表示処理へ

MODE_DISP4_MINUS
    movlw   b'00001000'     ; マイナスの場合の処理
    movwf   LED_DATA1       ; マイナスを描く
    movf    OSC_ADJUST, W
    xorlw   b'00011111'     ; OSC_ADJUSTの下5ビットの符号を逆転
    movwf   WK1             ; それをWK1に代入して
    incf    WK1, f          ; 1を足して２の補数にする

MODE_DISP4_BAR
    movlw   LED_DATA4
    movwf   FSR             ; 間接アクセスポインタをLED_DATA4にセット
    movlw   d'8'
    subwf   WK1, W          ; WK1 - 8を計算
    btfss   STATUS, C       ; C=1, つまりボローがなければスキップ
    goto    MODE_DISP4_BAR2 ; ボローあり＝8未満の場合

    movlw   b'11111111'     ; 8以上の場合
    movwf   INDF            ; 8個分のLEDを点灯
    incf    FSR, f          ; 表示位置を１つ右にシフト
    movlw   d'8'
    subwf   WK1, f          ; WK1から８を引く

MODE_DISP4_BAR2
    clrf    WK2             ; WK2には端数分のバー表示がセットされる
    movf    WK1, f
    btfsc   STATUS, Z       ; WK1がゼロかをチェック
    return                  ; 余りがゼロなら終了

MODE_DISP4_LOOP
    bsf     STATUS, C       ; キャリーフラグセット
    rrf     WK2, f          ; 右シフト
    decfsz  WK1, f          ; WK1をデクリメントしてゼロならスキップ
    goto    MODE_DISP4_LOOP ; WK1がゼロになるまでくり返す

    movf    WK2, W          ; これで端数分のバー表示が作成できたので
    movwf   INDF            ; 必要な数のLEDを点灯
    return

;-------------------------------------------------------
;   割り込みルーチン（Timer0の割り込みで飛んでくる）
;-------------------------------------------------------
INTERRUPT
    movwf   BKUP_W          ; レジスタのバックアップ
    swapf   STATUS, W       ; (STATUSを変化させずにバックアップする
    movwf   BKUP_STATUS     ;  ためにswapfを使っている)
    movf    FSR, W
    movwf   BKUP_FSR
    movf    PCLATH, W
    movwf   BKUP_PCLATH

;タイマールーチン
    decf    SECTIMER, f     ; タイマーカウントダウン
    btfss   STATUS, Z       ; カウントダウンが終わったらスキップ
    goto    LED_OUTPUT      ; 0.5秒経過していない

    btfss   SEC2_CHANGE, 1  ; ビット1が立っているときは１回だけスキップ
    bsf     SEC2_CHANGE, 0  ; 0.5秒経過したのでフラグを立てる

    bcf     SEC2_CHANGE, 1  ; ビット1はクリア
    movlw   SECTIMER_CONST
    movwf   SECTIMER        ; タイマーリセット
    decf    SECTIMER_LEAP, f    ; うるう秒カウンタをデクリメント
    btfss   STATUS, Z       ; 次がうるう秒に当たる場合はスキップ
    goto    LED_OUTPUT      ; うるう秒でない場合は先に進む

    incf    SECTIMER, f     ; うるう秒の場合はカウンタを245にする
    movlw   d'8'            ; うるう秒タイマーをリセット
    movwf   SECTIMER_LEAP   ; 0.5秒カウンタなのでうるう秒8回に1回で0.25Hz分

LED_OUTPUT                  ; ここからダイナミック点灯処理
    movlw   b'11111111'
    btfss   PORTA, 5        ; A5(ブザー出力)の状態を取得
    movlw   b'11011111'

    movwf   BUZZER_STAT     ; その状態を保存しておく
    movf    ROW_COUNT, W    ; スキャンポートの開き方をあらかじめ計算する
    movwf   WK_SCAN         ; WK_SCANには開くべきビット位置が入る
    sublw   d'3'            ; 3 - W を計算
    btfsc   STATUS, Z       ; 対象ポートが(3)でなければスキップ
    incf    WK_SCAN         , f ; Scan(3)=A(4)、ここだけずれる

    movlw   b'1111111O'
    movwf   SCAN_PORT
    bsf     STATUS, C       ; 11111110(C=1) というビット状態にする
    movf    WK_SCAN, f      ; WK_SCANかゼロかどうかチェック
    btfsc   STATUS, Z
    goto    SET_PORT        ; 既にゼロなら先に進む

PORT_SHIFT_LOOP             ; ポートAまたはBの開き方を設定する
    rlf     SCAN_PORT, f    ; 左シフト
    decfsz  WK_SCAN, f      ; ゼロになるまでくり返す
    goto    PORT_SHIFT_LOOP

SET_PORT
    movlw   b'11111111'     ; まずはスキャンポートを閉じる
    movwf   PORTB
    movf    BUZZER_STAT, W
    movwf   PORTA
    movf    ROW_COUNT, W    ; 次にポートCを設定する
    addlw   LED_DATA0       ; W ← LED_DATA0アドレス＋ROW_COUNT
    movwf   FSR
```

```
    movf    INDF, W         ; 書き込むべきデータを取ってきて
    movwf   PORTC           ; PORTCに出力する
    movf    SCAN_PORT, W
    btfsc   ROW_COUNT, 2    ; ROW_COUNTが4～7ならPORTBを開く
    goto    OPEN_PORTB

    andwf   BUZZER_STAT, W  ; ROW_COUNTが0～3なら、ブザーの
    movwf   PORTA           ; 状態を保護しつつPORTAを開く
    goto    INC_ROW_COUNT

OPEN_PORTB
    movwf   PORTB           ; ROW_COUNTが4～7ならPORTBを開く

INC_ROW_COUNT
    incf    ROW_COUNT, W    ; ROW_COUNTを次の値にする
    andlw   d'7'            ; 0～7でぐるぐる回す
    movwf   ROW_COUNT

    movf    CHATTER_CNT, f  ; ここからタクトスイッチ入力判定ルーチン
    btfss   STATUS, Z
    goto    DEC_WK_CHAT     ; CHATTER_CNTがゼロ以外ならチャタリングカウント中

    btfss   PORTA, 3        ; タクトスイッチ入力チェック
    goto    BTN_ON_CHECK    ; 押されている場合

    btfss   BTN_STAT, 0     ; 押されていない場合の判定
    goto    RESTORING       ; もともと押されていない（変化なし）
    goto    SET_WK_CHAT     ; 離された（ON→OFF）（変化あり）

BTN_ON_CHECK
    btfsc   BTN_STAT, 0     ; 押されている場合の判定
    goto    RESTORING       ; もともと押されていない（変化なし）
                            ; 押された（OFF→ON）場合はそのまま次の処理へ

SET_WK_CHAT
    movlw   d'11'
    movwf   CHATTER_CNT     ; チャタリングカウンタを11にセット
    goto    RESTORING

DEC_WK_CHAT
    decf    CHATTER_CNT, f  ; チャタリングカウントをデクリメント
    decfsz  CHATTER_CNT, W  ; カウントが１になったら再判定（１－１＝０）
    goto    RESTORING       ; そうじゃないときはまだ再判定しない

;ここから再判定ルーチン
    clrf    CHATTER_CNT     ; CHATTER_CNTをゼロにする
    btfsc   PORTA, 3        ; 最終判定用のタクトスイッチ状態を取得
    goto    BTN_RELEASED

    bsf     BTN_STAT, 0     ; 押された場合のフラグ設定
    bsf     BTN_STAT, 1
    goto    RESTORING

BTN_RELEASED
    bcf     BTN_STAT, 0     ; 離された場合のフラグ設定

RESTORING
    bcf     INTCON, T0IF    ; 割り込みT0IFビットをクリア
    movf    BKUP_PCLATH, W  ; ここから各レジスタを元に戻していく
    movwf   PCLATH
    movf    BKUP_FSR, W
    movwf   FSR
    swapf   BKUP_STATUS, W
    movwf   STATUS
    swapf   BKUP_W, F       ; STATUSレジスタを変えないように
    swapf   BKUP_W, W       ; Wレジスタを元に戻す
    retfie                  ; 割り込み処理終了

;-------------------------------------------------------------
;   残り時間を示すLED表示パターンをセットする。縦横が逆なので面倒！
;-------------------------------------------------------------
SET_LED_DATA
    btfss   OPTION_MODE, 3  ; 特殊表示モードか？
    goto    SET_LED_NORMAL  ; そうでなければ通常モードへ

    btfsc   OPTION_MODE, 2  ; アナログ・デジタルタイマーモードか？
    goto    ANALOG_MODE     ; アナログ・デジタルタイマーモードへジャンプ

    btfsc   OPTION_MODE, 1  ; 砂時計モードか？
    goto    HOURGLASS       ; 砂時計モードへジャンプ

    movf    MIN_LED, W      ; フルLEDモードの処理
    btfsc   OPTION_MODE, 0  ; フルLED・リバースモードか？
    sublw   d'63'           ; リバースモードなら、63 - MIN_LED → W

    movwf   MINUTE_LED      ; MIN_LEDをMINUTE_LEDに代入して
    goto    SET_LED_MODE2b  ; 通常の8進表示モードへ

SET_LED_NORMAL              ; 通常表示モードの処理
    movf    MINUTE, W
    movwf   MINUTE_LED      ; MINUTE_LEDにMINUTEの値を代入

    btfss   MIN_ZONE, 1     ; 4分未満の時間がセットされていたか？
    goto    SET_LED_DATA2   ; 4分以上の場合、先に進む

    btfsc   OPTION_MODE, 0  ; 4分未満の表示設定を確認
    goto    SET_LED_MODE2b  ; 点灯数を増やさない設定ならそのまま8進表示へ

    clrf    WK1             ; WK1には分で加算すべきLED数が入る
    movf    MINUTE_LED, f   ; MINUTE_LEDがゼロかチェック
    btfsc   STATUS, Z
    goto    SET_LED_CALCSEC ; ゼロなら先の処理へ

    movlw   d'20'
    addwf   WK1, f          ; WK1に20(個)を足す
    decf    MINUTE_LED, f
    btfss   STATUS, Z       ; １を引いてからMINUTE_LEDがゼロかチェック
    addwf   WK1, f          ; ゼロでなければさらにWK1に20(個)を足す

SET_LED_CALCSEC
    movlw   d'6'
    movwf   DIV_OPERAND     ; 6で割る(3秒＝1LED)
    movf    SECOND2, W      ; 割られる数はSECOND2
    call    CALC_DIV        ; 割り算する
    addwf   WK1, W          ; 割り算の結果とWK1を加算して
    movwf   MINUTE_LED      ; MINUTE_LEDにセットする
    goto    SET_LED_MODE2b  ; 8進表示モードへ

SET_LED_DATA2               ; 4分以上の時間がセットされていた場合の処理
    btfss   MIN_ZONE, 0     ; 40分超の時間がセットされていたか？
    goto    SET_LED_UNDER40 ; 40分以内の場合

    btfss   OPTION_MODE, 2  ; 40分超を設定時の表示モードチェック
    goto    SET_LED_MODE2   ; 標準：8進表示モードへ
    goto    SET_LED_MODE1   ; 特別設定：5進表示モードへ

SET_LED_UNDER40
    btfss   OPTION_MODE, 1  ; 40分以内を設定時の表示モードチェック
    goto    SET_LED_MODE1b  ; 標準：5進表示モードへ
    goto    SET_LED_MODE2b  ; 特別設定：8進表示モードへ

;-------------------------
SET_LED_MODE1               ; 5進表示モード（40分超のときしか使われない）
    btfsc   MIN_ZONE, 2     ; 80分超フラグをチェック
    goto    MIN_OVER80      ; 80分超の場合、先に飛ぶ

    bcf     STATUS, C
    rrf     MINUTE_LED, f   ; MINUTE_LEDを２で割る
    goto    SET_LED_MODE1b

MIN_OVER80                  ; 80分超の場合のLED数の調整
    movlw   d'3'
    movwf   DIV_OPERAND     ; 3で割る
    movf    MINUTE_LED, W   ; 割られる数はMINUTE_LED
    call    CALC_DIV        ; 割り算する
    movwf   MINUTE_LED      ; 割り算の結果(W)をMINUTE_LEDにセットする
    movlw   d'40'           ; MINUTE_LEDが40以上になってないかチェック
    subwf   MINUTE_LED, W   ; W ← MINUTE_LED - 40
    btfss   STATUS, C       ; C=1、つまりボローがなければスキップ
    goto    SET_LED_MODE1b  ; MINUTE_LEDは40未満

    movlw   d'39'           ; MINUTE_LEDが40以上なら39に修正
    movwf   MINUTE_LED

SET_LED_MODE1b              ; 表示LED数の修正がいらないときはここから
    call    INC_MIN_LED     ; LEDの点灯・点滅を設定する
    movlw   d'5'            ; 5進数の商と余りを計算
    movwf   DIV_OPERAND     ; 5で割る
    movf    MINUTE_LED, W   ; 割られる数はMINUTE_LED
    call    CALC_DIV        ; 割り算する
    movwf   LED_BASE        ; 割り算の結果(W)をLED_BASEに暫定的にセットする
    movf    DIV_MOD, W
    movwf   LED_MOD         ; 割り算の余りをLED_MODにセットする
    call    DEMUX_LED_BASE  ; LED_BASEをデマルチプレクスする
    clrf    LED_PTR         ; LED_PTRに0をセット

CALC_PAT5_LOOP
    call    STORE_PATTERN   ; パターンをLED_DATAに書き込む
    movlw   d'4'
    subwf   LED_PTR, W      ; LED_PTRが4かをチェック
    btfsc   STATUS, Z       ; 4でなければ次をスキップ
    goto    ERASE_LAST3     ; 4なら残り3列を消す処理へ

    incf    LED_PTR, f      ; LED_PTRをインクリメント
    goto    CALC_PAT5_LOOP

ERASE_LAST3
    clrf    LED_DATA5       ; あとLED_DATAが3列残っているのでクリアしておく
    clrf    LED_DATA6
    clrf    LED_DATA7
    return

;-------------------------
SET_LED_MODE2               ; 8進表示モード
    movf    MINUTE_LED, W
    sublw   d'60'           ; 60 - MINUTE_LED を計算してみる
    btfsc   STATUS, C       ; C=0、つまりボローが発生したらスキップ
    goto    SET_LED_MODE2b  ; 60分以下の場合、MINUTE = MINUTE_LED

    movlw   d'15'           ; 60分超の場合の表示LED数の調整
    movwf   DIV_OPERAND     ; 割る数は１５
    movlw   d'60'
    subwf   MINUTE_LED, W   ; MINUTE_LED - 60 → W
    call    CALC_DIV        ; 割り算する
    addlw   d'60'           ; 割り算の結果(W)に60を加算して
    movwf   MINUTE_LED      ; MINUTE_LEDにコピー
    movlw   d'64'           ; MINUTE_LEDが64以上になってないかチェック
    subwf   MINUTE_LED, W   ; W ← MINUTE_LED - 64
    btfss   STATUS, C       ; C=1、つまりボローがなければスキップ
    goto    SET_LED_MODE2b  ; ボローがあるときは通常処理へ

    movlw   d'63'           ; MINUTE_LEDが64以上なら63に修正
    movwf   MINUTE_LED

SET_LED_MODE2b              ; 表示LED数の修正がいらないときはここから
    call    INC_MIN_LED     ; LEDの点灯・点滅を設定する
    movf    MINUTE_LED, W   ; 割られる数はMINUTE_LED
    call    CALC_DIV8       ; 8で割る
    movwf   LED_BASE        ; 割り算の結果(W)をLED_BASEに暫定的にセットする
    movf    DIV_MOD, W
    movwf   LED_MOD         ; 割り算の余りをLED_MODにセットする
    call    DEMUX_LED_BASE  ; LED_BASEをデマルチプレクスする
    clrf    LED_PTR         ; LED_PTRに0をセット

CALC_PAT8_LOOP
    call    STORE_PATTERN   ; パターンをLED_DATAに書き込む
    movlw   d'7'
    subwf   LED_PTR, W      ; LED_PTRが7かをチェック
    btfsc   STATUS, Z       ; 7でなければ次をスキップ
    return                  ; 7ならリターン

    incf    LED_PTR, f      ; LED_PTRをインクリメント
    goto    CALC_PAT8_LOOP

;-------------------------
INC_MIN_LED
    btfss   OPTION_MODE, 6  ; LED点灯モードをチェック
    goto    INC_MIN_LED2

    incf    MINUTE_LED, f   ; 点灯モードなら常に１つ増やす（点灯）
    return

INC_MIN_LED2                ; そうでない場合、
    movf    MINUTE_LED, W
    xorwf   SECOND2, W      ; MINUTE_LEDとSECOND2のXORをとる→Wレジへ
    andlw   b'00000001'     ; 下１けたのビットテスト
    btfsc   STATUS, Z       ; MINUTE_LEDとSECOND2のビット0が違ったらそのまま
    incf    MINUTE_LED, f   ; 同じ場合は１つ増やす（点滅）

    return

;-------------------------
DEMUX_LED_BASE
    movf    LED_BASE, W
    movwf   WK1             ; LED_BASEをWK1に退避
    clrf    LED_BASE        ; LED_BASEを00000000にする
    movf    WK1, f          ; WK1がゼロかどうかチェック
    btfsc   STATUS, Z
    goto    DEMUX_LB_FINISH ; ゼロなら処理をスキップ

DEMUX_LB_LOOP
    bsf     STATUS, C       ; キャリーフラグをオンにして
    rlf     LED_BASE, f     ; 左シフト（右からONのビットが入ってくる）
    decfsz  WK1, f
    goto    DEMUX_LB_LOOP   ; WK1の値だけ繰り返す

DEMUX_LB_FINISH
    movlw   LED_DATA0
    movwf   FSR             ; 間接アクセス用ポインタをLED_DATA先頭にセット
    return

;-------------------------
STORE_PATTERN               ; LED_BASE/MOD/PTRでLEDパターン書込み
    movf    LED_BASE, W
    movwf   WK1             ; WK1にはLED_BASEのコピーが入る
    movf    LED_MOD, W
    subwf   LED_PTR, W      ; LED_PTR(0→4or7) - LED_MOD
    btfsc   STATUS, C       ; C=0、つまりボローが発生したらスキップ
    goto    SET_LED_DAT     ; 追加点灯不用の場合は先に進む

    bsf     STATUS, C       ; ボローあり＝もう１個点灯すべき場合は
    rlf     WK1, f          ; もう１個追加点灯

SET_LED_DAT
    movf    WK1, W          ; 実際に書き込むべきパターンが計算できたので
    movwf   INDF            ; LED_DATAに点灯パターン書き込み
    incf    FSR, f          ; LED_DATAの次の列へ
    return

;-------------------------
HOURGLASS                   ; 砂時計モード
    movlw   LED_BUF0        ; まずはベースとなる設定分数のパターンを生成
```

```
    movwf   FSR             ; 間接アクセス用ポインタをLED_BUF先頭にセット
    clrf    WK_HG1          ; WK_HG1に0をセット

HG_FRAME_LOOP
    movf    HG_LEDBASE, W
    movwf   WK_HG2          ; WK_HG2にはHG_LEDBASEのコピーが入る
    movf    HG_MOD, W
    subwf   WK_HG1, W       ; WK_HG1(0→7) - HG_MOD
    btfsc   STATUS, C       ; C=0、つまりボローが発生したらスキップ
    goto    HG_FRAME_LOOP2  ; 追加点灯不用の場合は先に進む

    bsf     STATUS, C       ; ボローあり＝もう１個点灯すべき場合は
    rlf     WK_HG2, f       ; もう１個追加点灯

HG_FRAME_LOOP2
    movf    WK_HG2, W       ; 実際に書き込むべきパターンが計算できたので
    movwf   INDF            ; LED_BUFに点灯パターン書き込み
    incf    FSR, f          ; LED_BUFの次の列へ
    movlw   d'7'
    subwf   WK_HG1, W       ; WK_HG1が7かをチェック
    btfsc   STATUS, Z       ; 7でなければ次をスキップ
    goto    HG_PILE_BASE    ; 7なら終了、次の処理へ

    incf    WK_HG1, f       ; WK_HG1をインクリメント
    goto    HG_FRAME_LOOP   ; 処理を繰り返す

HG_PILE_BASE                ; 行単位の「堆積」
    movf    HG_DIVOPE, W
    movwf   DIV_OPERAND     ; 割る数は1～3
    movf    MINUTE, W
    addlw   d'1'
    subwf   MIN_INI, W      ; W ← MIN_INI - MINUTE - 1
    btfss   STATUS, C       ; C=1, つまりボローがなければスキップ
    clrw                    ; ボローがあった場合はWをゼロにする

    call    CALC_DIV        ; 「MIN_INI - MINUTE - 1」を割り算する
    call    CALC_DIV8       ; さらに8で割る
    movwf   HG_MINDIV       ; 割り算の商をHG_MINDIVにセットする
    movf    DIV_MOD, W
    movwf   HG_MINMOD       ; 割り算の余りをHG_MINMODにセットする
    movf    HG_MINDIV, f    ; HG_MINDIVがゼロかどうかチェック
    btfsc   STATUS, Z
    goto    HG_PILE_MOD     ; ゼロなら処理をスキップ

    movlw   LED_BUF0
    movwf   FSR             ; 間接アクセス用ポインタをLED_BUF先頭にセット

HG_PILE_BASE_LOOP
    movf    HG_MINDIV, W
    movwf   WK_HG1          ; HG_MINDIVの値をWK_HG1にセット
    movf    INDF, W         ; LED_BUFから点灯パターンを取得
    movwf   WK_HG2          ; そのパターンをWK_HG2にコピー

HG_PILE_BASE_LOOP2
    bsf     STATUS, C       ; キャリーフラグをオンにして
    rrf     WK_HG2, f       ; 右シフト（全体が右に回る）
    decfsz  WK_HG1, f
    goto    HG_PILE_BASE_LOOP2  ; WK_HG1の値だけ繰り返す

    movf    WK_HG2, W
    movwf   INDF            ; 回転後のデータを書き戻す
    movf    FSR, W
    sublw   LED_BUF7        ; W ← LED_BUF7 - FSR
    btfsc   STATUS, Z       ; FSRは既にLED_BUF7まできている？
    goto    HG_PILE_MOD     ; きている場合は先へ

    incf    FSR, f          ; そうでない場合はポインタをインクリメントして
    goto    HG_PILE_BASE_LOOP   ; 繰り返し処理を続ける

HG_PILE_MOD                 ; 端数部分の「堆積」
    movf    HG_MOD_ADDR, W  ; タイマースタート時の末端位置がここに入っている
    movwf   FSR             ; 間接アクセスポインタをその場所に移動
    movf    HG_MINMOD, f    ; HG_MINMODがゼロかをチェック
    btfsc   STATUS, Z
    goto    HG_ANIME        ; ゼロなら処理をスキップ

    movf    HG_MINMOD, W
    movwf   WK_HG1          ; HG_MINMODの値をWK_HG1にセット

HG_PILE_MOD_LOOP
    bsf     STATUS, C       ; キャリーフラグをオンにして
    rrf     INDF, f         ; LED_BUFを直接右シフト
    movf    FSR, W
    sublw   LED_BUF0        ; FSRは既にLED_BUF0まできている？
    btfsc   STATUS, Z
    goto    HG_PILE_MOD2    ; きている場合の処理は先におく

    decf    FSR, f          ; きていない場合はFSRをデクリメント
    goto    HG_PILE_MOD3

HG_PILE_MOD2
    movlw   LED_BUF7        ; きている場合はFSRをLED_BUF7へ
    movwf   FSR

HG_PILE_MOD3
    decf    WK_HG1, f       ; WK_HG1をデクリメント
    btfss   STATUS, Z
    goto    HG_PILE_MOD_LOOP    ; ゼロになるまで繰り返す

HG_ANIME                    ; 砂の落下アニメーションの設定
    movf    HG_GAP, W
    movwf   WK_HG1          ; 砂落下間隔初期値をWK_HG1にセット
    movf    FSR, W
    subwf   HG_MOD_ADDR, W  ; 当初の末端位置と今のポインタ位置を比較
    btfss   STATUS, C       ; C=1、つまりボローなしの場合はスキップ
    incf    WK_HG1, f       ; ボローありの場合は間隔を１つ増やす

    movf    WK_HG1, W
    addwf   HG_MINDIV, W    ; W ← 落下間隔 + 砂積みあがり行数
    movwf   WK_HG2          ; その値をWK_HG2に入れる
    decf    WK_HG2, f       ; １減らす（落下位置は必ず積みあがり行数－１）
    call    HG_ANIME_SUB    ; WK_HG2の数だけシフトする
    movf    WK_HG3, W
    xorwf   INDF, f         ; 落下部分の砂を消す

    movf    WK_HG1, W
    movwf   DIV_OPERAND     ; 割る数は落下間隔
    movf    SECOND2, W
    sublw   d'119'          ; 割られる数は119-SECOND2
    btfsc   OPTION_MODE, 6  ; LED点滅なしか？
    clrw                    ; 点滅なしなら割られる数を0固定に

    call    CALC_DIV        ; 割り算する
    movf    DIV_MOD, W
    subwf   WK_HG1, W       ; W ← 落下間隔 - 余り
    addwf   HG_MINDIV, W    ; W ← 落下間隔 - 余り + 砂積みあがり行数
    movwf   WK_HG2          ; その値をWK_HG2に入れる
    decf    WK_HG2, f       ; １減らす（落下位置は必ず積みあがり行数－１）
    call    HG_ANIME_SUB    ; WK_HG2の数だけシフトする
    movf    WK_HG3, W
    iorwf   INDF, f         ; 砂を表示
    call    COPY_BUF2DATA   ; BUFのパターンを表示エリアにコピー
    return

;---------------------------
HG_ANIME_SUB
    movlw   b'10000000'     ; 落下する砂を設定
    movwf   WK_HG3          ; 砂パターンはWK_HG3に作る
    movf    WK_HG2, f       ; シフト数はあらかじめWK_HG2に入れておく
    btfsc   STATUS, Z       ; 砂の位置は一番下？
    return                  ; その場合は戻る

HG_ANIME_SUB_LOOP
    bcf     STATUS, C
    rrf     WK_HG3, f       ; 右回転（砂を上げていく）
    decfsz  WK_HG2, f
    goto    HG_ANIME_SUB_LOOP

    return

;---------------------------
ANALOG_MODE                 ; アナログタイマーのように表示するモード
    btfss   OPTION_MODE, 1  ; デジタルタイマーモードか？
    goto    DIGITAL_MODE    ; デジタルタイマーモードへジャンプ

    btfsc   OPTION_MODE, 0  ; トーテムポールモードか？
    goto    TOTEM_POLE      ; トーテムポールモードへジャンプ

    movlw   d'3'            ; アナログモードでは1LED＝3分
    movwf   DIV_OPERAND     ; 割る数は3
    movf    MINUTE, W       ; 割られる数はMINUTE
    call    CALC_DIV        ; 割り算する
    movwf   MINUTE_LED      ; 割り算の結果をMINUTE_LEDに代入
    call    INC_MIN_LED     ; LEDの点灯・点滅を設定する
    call    CLEAR_LED_BUF   ; LEDバッファをクリア
    movf    MINUTE_LED, f   ; MINUTE_LEDがゼロかどうかをチェック
    btfsc   STATUS, Z
    goto    ANALOG_FINISH   ; MINUTE_LEDがゼロならすぐ最後の処理へ

    movlw   b'00000001'
    iorwf   LED_BUF3, f     ; 1個めを点灯
    DECMIN_CHECK
    iorwf   LED_BUF2, f     ; 2個めを点灯
    DECMIN_CHECK
    iorwf   LED_BUF1, f     ; 3個めを点灯
    DECMIN_CHECK

    movlw   b'00000010'
    iorwf   LED_BUF0, f     ; 4個めを点灯
    DECMIN_CHECK
    movlw   b'00000100'
    iorwf   LED_BUF0, f     ; 5個めを点灯
    DECMIN_CHECK
    movlw   b'00001000'
    iorwf   LED_BUF0, f     ; 6個めを点灯
    DECMIN_CHECK
    movlw   b'00010000'
    iorwf   LED_BUF0, f     ; 7個めを点灯
    DECMIN_CHECK
    movlw   b'00100000'
    iorwf   LED_BUF0, f     ; 8個めを点灯
    DECMIN_CHECK
    movlw   b'01000000'
    iorwf   LED_BUF0, f     ; 9個めを点灯
    DECMIN_CHECK

    movlw   b'10000000'
    iorwf   LED_BUF1, f     ; 10個めを点灯
    DECMIN_CHECK
    iorwf   LED_BUF2, f     ; 11個めを点灯
    DECMIN_CHECK
    iorwf   LED_BUF3, f     ; 12個めを点灯
    DECMIN_CHECK
    iorwf   LED_BUF4, f     ; 13個めを点灯
    DECMIN_CHECK
    iorwf   LED_BUF5, f     ; 14個めを点灯
    DECMIN_CHECK
    iorwf   LED_BUF6, f     ; 15個めを点灯
    DECMIN_CHECK

    movlw   b'01000000'
    iorwf   LED_BUF7, f     ; 16個めを点灯
    DECMIN_CHECK
    movlw   b'00100000'
    iorwf   LED_BUF7, f     ; 17個めを点灯
    DECMIN_CHECK
    movlw   b'00010000'
    iorwf   LED_BUF7, f     ; 18個めを点灯
    DECMIN_CHECK
    movlw   b'00001000'
    iorwf   LED_BUF7, f     ; 19個めを点灯
    DECMIN_CHECK
    movlw   b'00000100'
    iorwf   LED_BUF7, f     ; 20個めを点灯
    DECMIN_CHECK
    movlw   b'00000010'
    iorwf   LED_BUF7, f     ; 21個めを点灯
    DECMIN_CHECK

    movlw   b'00000001'
    iorwf   LED_BUF6, f     ; 22個めを点灯
    DECMIN_CHECK
    iorwf   LED_BUF5, f     ; 23個めを点灯
    DECMIN_CHECK
    iorwf   LED_BUF4, f     ; 24個めを点灯
    DECMIN_CHECK

    movlw   b'00000010'
    iorwf   LED_BUF3, f     ; 25個めを点灯
    DECMIN_CHECK
    iorwf   LED_BUF2, f     ; 26個めを点灯
    DECMIN_CHECK

    movlw   b'00000100'
    iorwf   LED_BUF1, f     ; 27個めを点灯
    DECMIN_CHECK
    movlw   b'00001000'
    iorwf   LED_BUF1, f     ; 28個めを点灯
    DECMIN_CHECK
    movlw   b'00010000'
    iorwf   LED_BUF1, f     ; 29個めを点灯
    DECMIN_CHECK
    movlw   b'00100000'
    iorwf   LED_BUF1, f     ; 30個めを点灯
    DECMIN_CHECK

    movlw   b'01000000'
    iorwf   LED_BUF2, f     ; 31個めを点灯
    DECMIN_CHECK
    iorwf   LED_BUF3, f     ; 32個めを点灯
    DECMIN_CHECK
    iorwf   LED_BUF4, f     ; 33個めを点灯
    DECMIN_CHECK
    iorwf   LED_BUF5, f     ; 34個めを点灯
    DECMIN_CHECK

    movlw   b'00100000'
    iorwf   LED_BUF6, f     ; 35個めを点灯
    DECMIN_CHECK
    movlw   b'00010000'
    iorwf   LED_BUF6, f     ; 36個めを点灯
    DECMIN_CHECK
    movlw   b'00001000'
    iorwf   LED_BUF6, f     ; 37個めを点灯
    DECMIN_CHECK
    movlw   b'00000100'
    iorwf   LED_BUF6, f     ; 38個めを点灯
    DECMIN_CHECK

    movlw   b'00000010'
    iorwf   LED_BUF5, f     ; 39個めを点灯
    DECMIN_CHECK
```

```
    iorwf   LED_BUF4, f     ; 40個めを点灯

ANALOG_FINISH
    call    COPY_BUF2DATA   ; LED_BUFからLED_DATAへ転送
    return

;-------------------------
DIGITAL_MODE                ; デジタルタイマーを表示するモード
    movf    MINUTE, W
    movwf   MINUTE_LED      ; MINUTEをMINUTE_LEDにコピーして
    incf    MINUTE_LED, f   ; １を足す
    btfss   OPTION_MODE, 0  ; リバースモードをチェック
    goto    DIGITAL2        ; リバースモードでない場合

    movf    MINUTE_LED, W   ; リバースモード用の処理
    subwf   MIN_INI, W      ; W ← MIN_INI - (MINUTE + 1)
    btfss   STATUS, C       ; C=1, つまりボローがなければスキップ
    clrw                    ; ボローがあった場合はWをゼロにする

    movwf   MINUTE_LED      ; WをMINUTE_LEDに入れなおす

DIGITAL2
    movf    MINUTE_LED, W
    call    SHOW_NUMBER     ; 現在時間をデジタル表示する
    btfsc   OPTION_MODE, 6  ; LED点滅の設定チェック
    return                  ; 点滅なしの場合すぐ戻る

    btfsc   SECOND2, 0      ; SECOND2が奇数かどうかチェック
    bsf     LED_DATA7, 7    ; 点滅あり、かつSECOND2が奇数なら点灯する

    return

;-------------------------
TOTEM_POLE                  ; トーテムポールモード
    call    CLEAR_LED_DATA
    movlw   LED_DATA1       ; １列目のデフォルト位置
    movwf   FSR
    btfss   MINUTE, 6       ; 64分オーバーの部分がある？
    goto    TPOLE2          ; なければ先へ

    movlw   LED_DATA0       ; ある場合
    movwf   FSR             ; ポール変動位置を変える
    movlw   b'11111111'
    movwf   LED_DATA1       ; その隣は全点灯

TPOLE2
    movf    MINUTE, W
    andlw   b'00111111'     ; MINUTEの下６ビットだけにする
    call    CALC_DIV8       ; ８で割る
    call    TPOLE_SUB       ; まず分の上位ポールを計算
    movlw   LED_DATA3       ; 次に分の下位ポールへ
    movwf   FSR
    movf    DIV_MOD, W      ; 使うのは割り算の余り
    call    TPOLE_SUB       ; これで分の下位ポールも終了

    movlw   LED_DATA5       ; 今度は秒のポールを設定
    movwf   FSR             ; 秒の上位ポール
    movf    SECOND2, W
    movwf   WK1             ; SECOND2をWK1にコピー
    bcf     STATUS, C
    rrf     WK1, f          ; 右シフト１回で秒未満を削除
    movf    WK1, W          ; Wレジスタに入れなおして
    call    CALC_DIV8       ; ８で割る
    call    TPOLE_SUB

    btfsc   OPTION_MODE, 6  ; LED点灯モードをチェック
    return                  ; 点灯モードなら下位ポール非表示

    movlw   LED_DATA7       ; 次に下位ポールへ
    movwf   FSR
    movf    DIV_MOD, W      ; 使うのは割り算の余り
    call    TPOLE_SUB       ; これで分の下位ポールも終了
    return

;-------------------------
TPOLE_SUB                   ; Wに入った値でポール表示
    movwf   WK1             ; WをWK1に退避
    movf    WK1, f          ; ゼロかどうかチェック
    btfsc   STATUS, Z
    return                  ; ゼロなら何もせず戻る

TPOLE_SUB_LOOP
    bsf     STATUS, C       ; Cフラグをオンにして
    rrf     INDF, f         ; 右シフトでドットをせり上げ
    decfsz  WK1, f
    goto    TPOLE_SUB_LOOP  ; WK1がゼロになるまでくり返す

    return

;-----------------------------------------------------------------
;   タクトスイッチが押された際のアラーム時間設定ルーチン
;-----------------------------------------------------------------
SET_TIME
    movf    MINUTE, W
    movwf   WK1             ; WK1にMINUTEをコピー
    movlw   d'114'          ; SECOND2が114～119（最初の３秒）の間は
    subwf   SECOND2, W      ; 1分多い分数からの計算扱いにする
    btfsc   STATUS, C       ; C=0、つまりボローが発生したらスキップ
    incf    WK1, f          ; 計算上のMINUTEを１つ増やす

    movlw   d'5'
    subwf   WK1, W          ; WK1 - 5を計算してみる
    btfsc   STATUS, C       ; C=0、つまりボローが発生したらスキップ
    goto    SET_TIME_OVER4  ; 4分超のルーチンへ

    incf    WK1, f          ; WK1＝現在分数＋１分
    goto    UPDATE_MINUTE

SET_TIME_OVER4
    movlw   d'5'            ; 4分超の場合の時間設定
    movwf   DIV_OPERAND     ; 割る数は５
    movf    WK1, W          ; 割られる数はWK1
    call    CALC_DIV        ; 割り算する
    movwf   WK1             ; 結果をWK1に入れる
    bcf     STATUS, C       ; Cフラグクリア
    rlf     WK1, f
    rlf     WK1, f          ; WK1を４倍する
    addwf   WK1, f          ; さらに元のWを足して５倍
    movlw   d'5'
    addwf   WK1, f          ; さらに５を足して５倍＋５
    movlw   d'121'
    subwf   WK1, W          ; WK1 - 121を計算してみる
    btfss   STATUS, C       ; C=1、つまりボローがなければスキップ
    goto    UPDATE_MINUTE   ; ボローあり＝WK1が120以下なら先に飛ぶ

    movlw   d'1'            ; 120分超になった場合、
    movwf   WK1             ; WK2に1をセットする

UPDATE_MINUTE
    movf    WK1, W
    movwf   MINUTE          ; WK1をMINUTEに代入する
    movwf   MIN_INI         ; MIN_INIにも代入する
    call    CHECK_TIMEZONE  ; 設定時間のレンジをフラグセットする
    clrf    SECOND2         ; SECOND2はゼロにする
    call    RESET_TIMER     ; タイマーカウンタもリセット
    return

;-------------------------
CHECK_TIMEZONE              ; 設定時間がどのレンジに入っているか
    clrf    MIN_ZONE
    movlw   d'4';
    subwf   MINUTE, W       ; MINUTE - 4を計算
    btfsc   STATUS, C       ; C=0、つまりボローが発生したらスキップ
    goto    CHECK_TZ_OVER3  ; ボローなし、つまり3分超（4分以上）なら先に飛ぶ

    bsf     MIN_ZONE, 1     ; ボローあり、つまり3分以下ならフラグセットして
    goto    CHECK_TZ_SP     ; 特殊表示モードのチェックに飛ぶ

CHECK_TZ_OVER3              ; 3分超の場合の処理
    movlw   d'41';
    subwf   MINUTE, W       ; MINUTE - 41を計算
    btfss   STATUS, C       ; C=1、つまりボローがなければスキップ
    goto    CHECK_TZ_SP     ; ボローあり、つまり40分以下なら先に飛ぶ

    bsf     MIN_ZONE, 0     ; ボローなし、つまり41分以上ならフラグセットして
    movlw   d'81';          ; さらに80分超かどうかをチェック
    subwf   MINUTE, W       ; MINUTE - 81を計算
    btfsc   STATUS, C       ; C=0、つまりボローが発生したらスキップ
    bsf     MIN_ZONE, 2     ; ボローなし、つまり81分以上ならフラグセット

CHECK_TZ_SP
    btfss   OPTION_MODE, 3  ; 特殊表示モードをチェック
    return                  ; 通常表示モードならすぐ戻る

    btfsc   OPTION_MODE, 2  ; タイマー系モードをチェック
    return                  ; タイマー系モードならすぐ戻る

    btfsc   OPTION_MODE, 1  ; 砂時計モードをチェック
    goto    CHECK_TZ_HG     ; 砂時計モードなら必要な処理ルーチンへ

    movf    MINUTE, W       ; フルLEDモード時の処理
    movwf   MIN_COUNT_D     ; MINUTEをMIN_COUNT_Dにコピー
    bcf     STATUS, C
    rlf     MIN_COUNT_D, f  ; MIN_COUNT_Dを２倍する
    movf    MIN_COUNT_D, W
    movwf   MIN_COUNT       ; MIN_COUNT_Dの値をMIN_COUNTに代入
    movlw   d'63';
    movwf   MIN_LED         ; LED数は63個からスタート
    return

CHECK_TZ_HG                 ; 砂時計モード時の処理
    movlw   d'3'
    movwf   HG_DIVOPE       ; 分数を割る数を計算；スタートは３
    btfsc   OPTION_MODE, 0  ; 固定モードかどうかをチェック
    goto    CHECK_TZ_HG2    ; 固定モードなら割る数を３固定

    btfss   MIN_ZONE, 2     ; 80分超フラグをチェック
    decf    HG_DIVOPE, f    ; 80分以下ならー１（２になる）

    btfss   MIN_ZONE, 0     ; 40分超フラグをチェック
    decf    HG_DIVOPE, f    ; 40分以下ならー１（１になる）

CHECK_TZ_HG2
    movf    HG_DIVOPE, W
    movwf   DIV_OPERAND     ; 上で計算したHG_DIVOPEを割る数にセット
    movf    MINUTE, W       ; 割られる数はMINUTE
    call    CALC_DIV        ; 割り算実行
    movf    DIV_MOD, f      ; 余りがゼロかどうかチェック
    btfss   STATUS, Z
    addlw   d'1'            ; 余りがあればWに１を足しておく

    call    CALC_DIV8       ; さらに8で割る
    movwf   WK1             ; 商をWK1にコピーする
    sublw   d'8'            ; さらに再利用；「8 - 商」を計算して、
    movwf   HG_GAP          ; それをHG_GAPにコピーする（秒の落下間隔になる）
    movf    DIV_MOD, f      ; DIV_MODがゼロかどうかチェック
    btfsc   STATUS, Z
    incf    HG_GAP, f       ; DIV_MODがゼロならHG_GAPに１を足す

    movf    DIV_MOD, W
    movwf   HG_MOD          ; 余りをHG_MODにコピーする
    addlw   d'7'            ; さらに再利用；7を足して
    andlw   b'00000111'     ; 8で割った余りにして(基本的に-1をしている)
    addlw   LED_BUF0        ; LED_BUF0のアドレスを足して
    movwf   HG_MOD_ADDR     ; それをHG_MOD_ADDRにセットする

    clrf    HG_LEDBASE      ; HG_LEDBASEを00000000にする
    movf    WK1, f          ; WK1がゼロかどうかチェック
    btfsc   STATUS, Z
    return                  ; ゼロなら終了

CHECK_TZ_HG_LOOP
    bsf     STATUS, C       ; キャリーフラグをオンにして
    rlf     HG_LEDBASE, f   ; 左シフト（右からONのビットが入ってくる）
    decfsz  WK1, f
    goto    CHECK_TZ_HG_LOOP    ; WK1の値だけ繰り返す

    return

;-------------------------
RESET_TIMER
    movlw   SECTIMER_CONST  ; タイマー関連レジスタのリセット
    movwf   SECTIMER
    movlw   d'8'
    movwf   SECTIMER_LEAP
    clrf    SEC2_CHANGE
    return

;-----------------------------------------------------------------
;  アラームを鳴らす
;-----------------------------------------------------------------
ALARM
    movlw   d'32'           ; アラーム音を最大32回（約８秒）鳴らす
    movwf   WK1
    clrf    GRA_PTR         ; GRA_PTRをゼロにセット（これが標準）
    btfss   OPTION_MODE, 3  ; 特殊表示モードかをチェック
    goto    ALARM_SHOWGRA   ; 通常表示モードなら「０」表示

ALARM_SP                    ; 特殊表示モード時の処理
    btfss   OPTION_MODE, 2  ; タイマーモードをチェック
    goto    ALARM_NONTIMER  ; タイマーモードでない場合

    btfsc   OPTION_MODE, 1  ; アナログ／トーテムポールモードをチェック
    goto    ALARM_SHOWGRA   ; アナログ／トーテムポールモードなら「０」表示

    btfsc   OPTION_MODE, 0  ; デジタルリバースモードをチェック
    goto    ALARM_DIG_R     ; デジタルリバースモードへ

    clrw                    ; デジタルモードの処理
    call    SHOW_NUMBER     ; 「００」を表示する
    goto    ALARM_LOOP

ALARM_DIG_R                 ; デジタルリバースモードの処理
    movf    MIN_INI, W      ; MIN_INIの時間をWにセットして
    call    SHOW_NUMBER     ; それを表示する
    goto    ALARM_LOOP

ALARM_NONTIMER              ; タイマーモードでない場合
    btfsc   OPTION_MODE, 1  ; 砂時計モードをチェック
    goto    ALARM_SHOWGRA   ; 砂時計モードなら「０」表示

    btfss   OPTION_MODE, 0  ; フルLED・リバースモードをチェック
    goto    ALARM_SHOWGRA   ; フルLEDモードなら「０」表示

    movlw   h'018'          ; フルLED・リバースモードの場合、
    movwf   GRA_PTR         ; GRA_PTRに018h（■）をセット

ALARM_SHOWGRA
    call    SHOW_GRAPHIC    ; GRA_PTRで設定されたグラフィックを表示

ALARM_LOOP
    bcf     INTCON, TOIE    ; Timer0の割り込みを禁止する
```

```
    clrf    PORTC           ; LED表示を消す
    call    BEEP            ; ピッと音を出す（約0.1秒）
    btfss   OPTION_MODE, 5  ; アラームの設定をチェック
    goto    ALARM_WAIT      ; アラームありモードなら先へ飛ぶ

    movlw   d'100'          ; アラームなしモードなら0.1秒時間稼ぎ
    call    WAIT_NMS

ALARM_WAIT
    bsf     INTCON, TOIE    ; Timer0の割り込みを許可する
    movlw   d'150'
    call    WAIT_NMS        ; 時間かせぎ（約150ms）
    btfsc   BTN_STAT, 1     ; キーが押されなければそのまま
    return                  ; 押されたらすぐリターン

    decfsz  WK1, f          ; WK1がゼロになるまでくり返す
    goto    ALARM_LOOP

    return

;-----------------------------------------------------------------
;  ここから下は汎用ルーチンなのでWK1～WK4の使用は厳禁
;-----------------------------------------------------------------
;  BEEP音（約2650Hz、0.1秒）を出す
;-----------------------------------------------------------------
BEEP
    btfsc   OPTION_MODE, 5  ; アラームなしフラグをチェック
    return                  ; フラグがセットされていたら音は出さない

    bcf     INTCON, TOIE    ; Timer0の割り込みを禁止する
    clrf    PORTC           ; LEDにゴミが残るのを防止
    movlw   d'255'          ; 2650Hzの矩形波を255周期分発生させるので
    movwf   BEEP_CNT        ; 所要時間は255÷2650≒0.1秒

BEEP_LOOP
    movf    PORTA, W        ; 現在のポート状態を取り込む
    iorlw   b'00100000'
    movwf   PORTA           ; ブザーON
    call    WAIT_BEEP       ; call先では180*2サイクル、本体で8*2サイクル
    nop                     ; かかっているので計376サイクル≒2650Hz
    nop
    nop
    movf    PORTA, W        ; 現在のポート状態を取り込む
    andlw   b'11011111'
    movwf   PORTA           ; ブザーOFF
    call    WAIT_BEEP
    decfsz  BEEP_CNT, f
    goto    BEEP_LOOP       ; 255回くり返す
    bsf     INTCON, TOIE    ; Timer0の割り込みを許可する
    return

;-----------------------------------------------------------------
;  TICK音（カチッという音）を出す
;-----------------------------------------------------------------
TICK_SOUND                  ; カチ音
    btfsc   OPTION_MODE, 4  ; クリック音なしフラグをチェック
    return                  ; フラグがセットされていたら音は出さない

    movf    PORTA, W
    iorlw   b'00100000'
    movwf   PORTA           ; ブザーON
    call    WAIT_BEEP       ; 180サイクル時間待ち
    movf    PORTA, W
    andlw   b'11011111'
    movwf   PORTA           ; ブザーOFF
    return

;-----------------------------------------------------------------
;  時間待ち系
;-----------------------------------------------------------------
WAIT_BEEP                   ; BEEP音用180サイクルの時間待ち
    movlw   d'59'           ; 1
    movwf   WAIT_CNT1       ; 1
WAIT_BEEP_LOOP
    decfsz  WAIT_CNT1, f    ; 1 (2)
    goto    WAIT_BEEP_LOOP  ; 2
    return                  ; 2 → 1+1+(1+2)*58+2+2 = 180cycles

;-------------------------
WAIT_1MS                    ; 1msecの時間待ち
    movlw   d'249'          ; 1
    movwf   WAIT_CNT1       ; 1
WAIT_1MS_LOOP
    nop                     ; 1
    decfsz  WAIT_CNT1, f    ; 1 (2)
    goto    WAIT_1MS_LOOP   ; 2
    return                  ; 2 → 1+1+(1+1+2)*248+2+2=998(+ callで2)

;-------------------------
WAIT_NMS                    ; 約 n msec(n=255まで、Wにセットする）の時間待ち
    movwf   WAIT_CNT2       ; かなりいい加減な時間計算

WAIT_NMS_LOOP
    call    WAIT_1MS
    decfsz  WAIT_CNT2, f
    goto    WAIT_NMS_LOOP
    return

;-----------------------------------------------------------------
;  割り算をする
;  Wに割られる数、DIV_OPERANDに割る数をセットして呼ぶと
;  Wに商、DIV_MODに余りがセットされて戻る
;  CALC_DIV8は８で割るとき専用の高速計算ルーチン
;-----------------------------------------------------------------
CALC_DIV
    movwf   WK_CALCDIV1     ; WK_CALCDIV1に割られる数をコピー
    clrf    WK_CALCDIV2     ; WK_CALCDIV2はゼロ
    movf    DIV_OPERAND, W  ; Wレジスタに割る数をセット

CALC_DIV_LOOP
    subwf   WK_CALCDIV1, f
    btfss   STATUS, C       ; C=1、つまりボローがなければスキップ
    goto    CALC_DIV_FINISH ; ボローがあればループから抜ける

    incf    WK_CALCDIV2, f  ; WK_CALCDIV2には商が入る
    goto    CALC_DIV_LOOP

CALC_DIV_FINISH
    addwf   WK_CALCDIV1, W  ; WK_CALCDIV1は引きすぎなので戻す（Wレジで）
    movwf   DIV_MOD         ; Wレジの値をDIV_MOD（余り）にセット
    movf    WK_CALCDIV2, W  ; WK_CALCDIV2の値をWにセット
    return

;-------------------------
CALC_DIV8                   ; 8で割る場合のみの専用計算モード
    movwf   WK_CALCDIV2     ; WK_CALCDIV1に割られる数をコピー
    movwf   DIV_MOD         ; DIV_MODにも割られる数をコピー
    movlw   b'11111000'     ; まず商を計算
    andwf   WK_CALCDIV2, f  ; 上位５ビットのみを残して、
    bcf     STATUS, C       ; シフトのゴミになるCフラグをクリア
    rrf     WK_CALCDIV2, f  ; ３回右にシフト
    rrf     WK_CALCDIV2, f
    rrf     WK_CALCDIV2, f  ; これで商が求まる
    movlw   b'00000111'     ; 下位３ビットのみを残せば
    andwf   DIV_MOD, f      ; これで余りの計算はOK
    movf    WK_CALCDIV2, W  ; WK_CALCDIV2の値をWにセット
    return

;-----------------------------------------------------------------
;  LEDデータを空にする
;-----------------------------------------------------------------
CLEAR_LED_DATA
    clrf    LED_DATA0
    clrf    LED_DATA1
    clrf    LED_DATA2
    clrf    LED_DATA3
    clrf    LED_DATA4
    clrf    LED_DATA5
    clrf    LED_DATA6
    clrf    LED_DATA7
    return

CLEAR_LED_BUF
    clrf    LED_BUF0
    clrf    LED_BUF1
    clrf    LED_BUF2
    clrf    LED_BUF3
    clrf    LED_BUF4
    clrf    LED_BUF5
    clrf    LED_BUF6
    clrf    LED_BUF7
    return

;-----------------------------------------------------------------
;  LED_BUF から LED_DATAへデータ転送
;-----------------------------------------------------------------
COPY_BUF2DATA               ; BUFのパターンを表示エリアにコピー
    movf    LED_BUF0, W
    movwf   LED_DATA0
    movf    LED_BUF1, W
    movwf   LED_DATA1
    movf    LED_BUF2, W
    movwf   LED_DATA2
    movf    LED_BUF3, W
    movwf   LED_DATA3
    movf    LED_BUF4, W
    movwf   LED_DATA4
    movf    LED_BUF5, W
    movwf   LED_DATA5
    movf    LED_BUF6, W
    movwf   LED_DATA6
    movf    LED_BUF7, W
    movwf   LED_DATA7
    return

;-----------------------------------------------------------------
;  EEPROMにOPTION_MODE, MIN_DEFAULT, OSC_ADJUSTの値を書き込む
;       参考：各レジスタのBANK一覧（PIC16F689の場合）
;               BANK0:  PIR2
;               BANK2:  EEDAT, EEADR, INTCON
;               BANK3:  EECON1, EECON2
;-----------------------------------------------------------------
WRITE_EEPROM
    bcf     INTCON, GIE     ; 割込み禁止
    bsf     STATUS, RP0     ; BANK3に切り替え
    bsf     STATUS, RP1
    bcf     EECON1, EEPGD   ; データメモリーにアクセスする
    bsf     EECON1, WREN    ; EEPROMへの書き込み許可
    bcf     STATUS, RP0     ; BANK2に切り替え
    clrw                    ; EEPROMアドレスは0hを使う
    movwf   EEADR           ; EEPROMアドレス指定

    movf    OPTION_MODE, W  ; WレジスタにOPTION_MODEの値をコピー
    movwf   EEDAT           ; EEPROMに書き込むデータ指定
    call    WRITE_INITIATION    ; 書き込み
    movf    MIN_DEFAULT, W  ; WレジスタにMIN_DEFAULTの値をコピー
    movwf   EEDAT           ; EEPROMに書き込むデータ指定
    call    WRITE_INITIATION    ; 書き込み
    movf    OSC_ADJUST, W   ; WレジスタにOSC_ADJUSTの値をコピー
    movwf   EEDAT           ; EEPROMに書き込むデータ指定
    call    WRITE_INITIATION    ; 書き込み
    movf    HIST_CNT1, W    ; WレジスタにHIST_CNT1の値をコピー
    movwf   EEDAT           ; EEPROMに書き込むデータ指定
    call    WRITE_INITIATION    ; 書き込み
    movf    HIST_CNT2, W    ; WレジスタにHIST_CNT2の値をコピー
    movwf   EEDAT           ; EEPROMに書き込むデータ指定
    call    WRITE_INITIATION    ; 書き込み
    movf    HIST_CNT3, W    ; WレジスタにHIST_CNT3の値をコピー
    movwf   EEDAT           ; EEPROMに書き込むデータ指定
    call    WRITE_INITIATION    ; 書き込み

    bsf     STATUS, RP0     ; BANK3に切り替え
    bcf     EECON1, WREN    ; EEPROMへの書き込み禁止
    bcf     STATUS, RP0     ; BANK2に切り替え
    bsf     INTCON, GIE     ; 割り込み許可
    bcf     STATUS, RP1     ; BANK0に切り替え
    bcf     PIR2, EEIF      ; EEPROM書き込み終了フラグをクリア
    return

;-------------------------
WRITE_INITIATION
    bsf     STATUS, RP0     ; BANK3に切り替え
    movlw   55h             ; ここから書き込み儀式
    movwf   EECON2
    movlw   0aah
    movwf   EECON2          ; 儀式終了
    bsf     EECON1, WR      ; EECON1のWRビットをセット（書き込み開始）

WRITE_EEPROM_LOOP
    btfsc   EECON1, WR      ; WRビットが0になる（書き込み終了）のを待つ
    goto    WRITE_EEPROM_LOOP

    bcf     STATUS, RP0     ; BANK2に切り替え
    incf    EEADR, f        ; EEPROMアドレスを１つインクリメント
    return

;-----------------------------------------------------------------
;  EEPROMから設定値を読み出す
;-----------------------------------------------------------------
READ_EEPROM
    bcf     INTCON, GIE     ; 割込み禁止（BANK切り替え中の割り込みを防止）
    bsf     STATUS, RP0     ; BANK3に切り替え
    bsf     STATUS, RP1
    bcf     EECON1, EEPGD   ; データメモリーにアクセスする
    bcf     STATUS, RP0     ; BANK2に切り替え
    clrw                    ; EEPROMアドレスは0hを使う
    movwf   EEADR           ; EEPROMアドレス指定

    call    READ_INITIATION ; 読み出し
    movwf   OPTION_MODE     ; OPTION_MODEに書き込み
    call    READ_INITIATION ; 読み出し
    movwf   MIN_DEFAULT     ; MIN_DEFAULTに書き込み
    call    READ_INITIATION ; 読み出し
    andlw   b'00011111'     ; 念のため、余計なビットはクリアする
    movwf   OSC_ADJUST      ; OSC_ADJUSTに書き込み
    call    READ_INITIATION ; 読み出し
    movwf   HIST_CNT1       ; HIST_CNT1に書き込み
    call    READ_INITIATION ; 読み出し
    movwf   HIST_CNT2       ; HIST_CNT2に書き込み
    call    READ_INITIATION ; 読み出し
    movwf   HIST_CNT3       ; HIST_CNT3に書き込み

    bcf     STATUS, RP1     ; BANK0に切り替え
    bsf     INTCON, GIE     ; 割り込み許可

    movf    MIN_DEFAULT, f  ; MIN_DEFAULTが0でないかチェック
    btfsc   STATUS, Z
    goto    READ_EEPROM_ILLEGAL ; 0なら異常値処理

    movlw   d'121'
    subwf   MIN_DEFAULT, W  ; MIN_DEFAULT - 121
    btfss   STATUS, C       ; C=1、つまりボローがなければスキップ
```

```
    return                  ; ボローあり＝MIN_DEFAULTが120以下なら戻る

READ_EEPROM_ILLEGAL
    movlw   d'1'            ; MIN_DEFAULTに異常値が入った場合
    movwf   MIN_DEFAULT     ; MIN_DEFAULTは1（1分）にする
    return

;-------------------------
READ_INITIATION
    bsf     STATUS, RP0     ; BANK3に切り替え
    bsf     EECON1, RD      ; EEPROM読み出しモード
    bcf     STATUS, RP0     ; BANK2に切り替え
    movf    EEDAT, W        ; WレジスタにEEPROMから読み出した値が入る
    incf    EEADR, f        ; EEPROMアドレスを１つインクリメント
    return

;-----------------------------------------------------------------
;  Wにセットされた0～129の数字を表示する
;-----------------------------------------------------------------
    org     h'0540'
SHOW_NUMBER
    call    CLEAR_LED_DATA
    movwf   WK_SHOWNUM      ; WをWK_SHOWNUMに退避
    movlw   d'100'
    subwf   WK_SHOWNUM, W   ; WK_SHOWNUM - 100を計算してみる
    btfss   STATUS, C       ; C=1, つまりボローなしの場合スキップ
    goto    SHOWNUM2        ; ボローあり（100未満）の場合は先へ

    movlw   b'00011111'     ; 100以上の場合は
    movwf   LED_DATA0       ; 100の位に１を表示

SHOWNUM2
    movlw   d'10'
    movwf   DIV_OPERAND     ; 割る数は10
    movf    WK_SHOWNUM, W   ; 割られる数はWK_SHOWMIN
    call    CALC_DIV        ; 割り算を実行
    movwf   WK_SHOWNUM      ; 商をWK_SHOWNUMに入れる
    movlw   LED_DATA1
    movwf   FSR             ; 間接アクセスのポインタをLED_DATA1に
    movf    WK_SHOWNUM, W   ; 表示対象はWK_SHOWMIN＝10の位
    btfss   STATUS, Z       ; 10の位がゼロなら表示しない
    call    SHOWNUM_SUB

    movlw   LED_DATA5
    movwf   FSR             ; 間接アクセスのポインタをLED_DATA5に
    movf    DIV_MOD, W      ; 表示対象はDIV_MOD＝1の位
    call    SHOWNUM_SUB
    return

;-------------------------
SHOWNUM_SUB
    movwf   SHOWNUM_PTR     ; WをSHOWNUM_PTRに代入して
    bcf     STATUS, C
    rlf     SHOWNUM_PTR, f  ; 2倍して
    addwf   SHOWNUM_PTR, f  ; もう一度Wを足してWの3倍がSHOWNUM_PTRに入る
    movlw   d'3'            ; 3列分処理する
    movwf   SHOWNUM_CNT

SHOWNUM_SUB_LOOP
    call    GET_NUM_ROW     ; 1行取り出す
    movwf   INDF            ; LED_DATAに間接アクセスで書き込み
    incf    FSR, f          ; 次の行へ
    decfsz  SHOWNUM_CNT, f
    goto    SHOWNUM_SUB_LOOP    ; 3列分終わるまでループ

    return

;-------------------------
GET_NUM_ROW                 ; テーブルから必要な1行分取り出し
    movlw   h'05'
    movwf   PCLATH          ; PCLの上位アドレス

    movf    SHOWNUM_PTR, W  ; W ← SHOWNUM_PTR
    incf    SHOWNUM_PTR, f  ; 次に備えてSHOWNUM_PTRをインクリメント
    addwf   PCL, f          ; PCL ← PCL + W でテーブルアクセス

    retlw   b'00011111'     ; 0
    retlw   b'00010001'
    retlw   b'00011111'

    retlw   b'00010010'     ; 1
    retlw   b'00011111'
    retlw   b'00010000'

    retlw   b'00011101'     ; 2
    retlw   b'00010101'
    retlw   b'00010111'

    retlw   b'00010101'     ; 3
    retlw   b'00010101'
    retlw   b'00011111'

    retlw   b'00000111'     ; 4
    retlw   b'00000100'
    retlw   b'00011111'

    retlw   b'00010111'     ; 5
    retlw   b'00010101'
    retlw   b'00011101'

    retlw   b'00011111'     ; 6
    retlw   b'00010101'
    retlw   b'00011101'

    retlw   b'00000001'     ; 7
    retlw   b'00011001'
    retlw   b'00000111'

    retlw   b'00011111'     ; 8
    retlw   b'00010101'
    retlw   b'00011111'

    retlw   b'00010111'     ; 9
    retlw   b'00010101'
    retlw   b'00011111'

    retlw   b'00001110'     ; 10用の0
    retlw   b'00010001'
    retlw   b'00001110'

    retlw   b'00000000'     ; 11用の1
    retlw   b'00011111'
    retlw   b'00000000'

    retlw   b'00000000'     ; 12用の2
    retlw   b'00011101'
    retlw   b'00010111'

;-----------------------------------------------------------------
;  分表示LED点灯パターンへのテーブルアクセス
;-----------------------------------------------------------------
SHOW_GRAPHIC                ; retlwが始まるまでのこのプログラムは15Words
    movlw   d'8'            ; 8列分処理する
    movwf   SHOWGRA_CNT
    movlw   LED_DATA0
    movwf   FSR             ; 間接アクセスのポインタをLED_DATA0に

SHOW_GRA_LOOP
    call    GET_GRA_ROW     ; 1行取り出す
    movwf   INDF            ; LED_DATAに間接アクセスで書き込み
    incf    FSR, f          ; 次の行へ
    decfsz  SHOWGRA_CNT, f
    goto    SHOW_GRA_LOOP   ; 8列分終わるまでループ
    return

;-------------------------
GET_GRA_ROW                 ; テーブルから必要な1行分取り出し
    movlw   h'05'
    movwf   PCLATH          ; PCLの上位アドレス

    movf    GRA_PTR, W      ; W ← GRA_PTR
    incf    GRA_PTR, f      ; 次に備えてGRA_PTRをインクリメント
    addwf   PCL, f          ; PCL ← PCL + W でテーブルアクセス

    retlw   b'00000000'     ; 0 (00h)
    retlw   b'00111100'
    retlw   b'01111110'
    retlw   b'10000001'
    retlw   b'10000001'
    retlw   b'01111110'
    retlw   b'00111100'
    retlw   b'00000000'

    retlw   b'11000000'     ; バージョン (08h)
    retlw   b'00000000'     ; 左の４列がバージョンを表現。
    retlw   b'11111000'     ; 現在は「２．５４」
    retlw   b'11110000'
    retlw   b'00000000'
    retlw   b'00000000'
    retlw   b'00000000'

    retlw   b'00001110'     ; OK (010h)
    retlw   b'00010001'
    retlw   b'00001110'
    retlw   b'00000000'
    retlw   b'11111000'
    retlw   b'00100000'
    retlw   b'01010000'
    retlw   b'10001000'

    retlw   b'11111111'     ; ■ (018h)
    retlw   b'11111111'
    retlw   b'11111111'
    retlw   b'11111111'
    retlw   b'11111111'
    retlw   b'11111111'
    retlw   b'11111111'

    retlw   b'11111111'

    retlw   b'11111111'     ; 反転R (020h)
    retlw   b'10000001'
    retlw   b'10000001'
    retlw   b'11101101'
    retlw   b'11001101'
    retlw   b'10100001'
    retlw   b'10110011'
    retlw   b'11111111'

    retlw   b'10000001'     ; 砂時計 (028h)
    retlw   b'11000011'
    retlw   b'11100101'
    retlw   b'11011001'
    retlw   b'11011001'
    retlw   b'11100101'
    retlw   b'11000011'
    retlw   b'10000001'

    retlw   b'10011111'     ; 砂時計Fixed (030h)
    retlw   b'11000101'
    retlw   b'11100001'
    retlw   b'11011000'
    retlw   b'11011001'
    retlw   b'11100101'
    retlw   b'11000011'
    retlw   b'10000001'

    retlw   b'00000000'     ; デジタル (038h)
    retlw   b'00000000'
    retlw   b'01111100'
    retlw   b'00000000'
    retlw   b'01011100'
    retlw   b'01010100'
    retlw   b'01110100'
    retlw   b'00000000'

    retlw   b'11111111'     ; デジタル・リバース (040h)
    retlw   b'11111111'
    retlw   b'10000011'
    retlw   b'11111111'
    retlw   b'10100011'
    retlw   b'10101011'
    retlw   b'10001011'
    retlw   b'11111111'

    retlw   b'01111110'     ; アナログ (048h)
    retlw   b'10000001'
    retlw   b'10000001'
    retlw   b'10000001'
    retlw   b'10000001'
    retlw   b'10000001'
    retlw   b'10000001'
    retlw   b'01111110'

    retlw   b'00000000'     ; トーテムポール (050h)
    retlw   b'11111100'
    retlw   b'00000000'
    retlw   b'11110000'
    retlw   b'00000000'
    retlw   b'11111110'
    retlw   b'00000000'
    retlw   b'11000000'

    retlw   b'11100111'     ; 設定画面２用 (058h)
    retlw   b'01010101'
    retlw   b'11100101'
    retlw   b'00000000'
    retlw   b'11110111'
    retlw   b'10100101'
    retlw   b'01000010'
    retlw   b'00000000'

;-----------------------------------------------------------------
    end                     ; ここで終わり
;-----------------------------------------------------------------
```