Коментар по програмата – часовник

Коментарът е по-подробен от нормалното, в него има и пояснения за самия контролер PIC16F716

```
; Ver. 10.2013
;
      list  p=16F716
      __config h'fffb  ; __config h'fffb' е по-специална директива с която се задава стойността на конфигурационната дума, т.е типа на генератора,
                       ; дали ще е включено кучето (Watchdog) и други параметри задаващи режимите на работа на процесора. Тази дума се записва
                       ; апаратно – с програматор и не може да се променя в хода на изпълнението на програмата.
                       ;
Indf        equ   00   ; От адрес 0 до 0b са регистри заети от процесора – входно-изходните портове, указател за косвена адресация, програмен брояч и т.н
Timer       equ   01   ; С тях операциите се извършват както с останалите клетки от паметта които са с общо предназначение.
PCL         equ   02   ;
Status      equ   03   ;
Fsr         equ   04   ;
PortA       equ   05   ;
PortB       equ   06   ;
IntCon      equ   0b   ;

Numb        equ   20   ; Тук са дефинирани клетките които се използват в програмата. Имената им са зададени с оглед на предназначението им. Това е
Clock       equ   21   ; направено за по-добра читаемост. Може да се използват и директно адресите им - вместо Clock  да се използва 0x21. Това
Sec01       equ   22   ; обикновено се практикува в стандартни универсални програми, напр. за аритметични функции.
Sec1        equ   23
Sec10       equ   24
Min1        equ   25
Min10       equ   26
Hours1      equ   27
Hours10     equ   28
Flags       equ   29
Inds        equ   2a
PortAm      equ   2b
Segm        equ   2c
Dig0        equ   2d
Dig1        equ   2e
Dig2        equ   2f
Dig3        equ   30
Buttons     equ   31
ButtonF     equ   32
```

```
S_stack    equ   33     ; Тези клетки са заделени за "стек" защото в процесора няма специално адресно пространство за тази цел. Истинският стек не е
W_stack    equ   34     ; достъпен за програмиста и в него се помни само адресът на връщане от подпрограма или прекъсване. Стекът има осем нива, т.е
F_stack    equ   35     ; осем подпрограми могат да са една в друга. Едно ниво винаги трябва да се пази ,ако има разрешено прекъсване!

#define    C     Status,0   ; Тук се задават битовите променливи - флаговете за пренос, за нулев резултат или за управление на прекъсванията. Това са
#define    DC    Status,1   ; флагове на процесора и са в резултат от работата му или задават режима (напр. вкл./изкл. на прекъсване).
#define    Z     Status,2   ;
#define    GIE   IntCon,7   ;

#define    m_s   Flags,0    ; Това са флагове зададени от програмиста необходими за работата на програмата. Тези флагове не са
#define    AdjT  Flags,1    ; известни предварително, а се добавят в процеса на писане на програмата.
#defin     AddCl Flags,2    ;
#define    But   PortA,4    ;

#define    But0  Buttons ,0 ; Флаговете може да са битове от портовете или от регистри
#define    But1  Buttons ,1
#define    But2  Buttons ,2
#define    But3  Buttons ,3
#define    But0f ButtonF ,0
#define    But1f ButtonF ,1
#define    But2f ButtonF ,2
#define    But3f ButtonF ,3

OffInd     equ   b'1111'    ; Константи. - дефинират се за да може програмата по-лесно да се чете. Аналогично би могло на съответното място вместо
OffSeg     equ   h'00'      ; OffInd да се използва b'1111'. Всъщност асемблерът прави точно това – замества имената със стойности. Но ако
;                           ; константа се ползва на няколко места, чрез предварителното дефиниране промяната става само на едно място.
;***************
Reset
     call  IniPic      ; От това място започва същинската програма. При този процесор след Reset програмният брояч се установява в нула и от нулев
     call  ClrRAM      ; адрес се взема  първата команда. След това стъпка след стъпка се изпълнява програмата. Обикновено първите команди или
     bsf   GIE         ; подпрограми са за начално установяване. В случая това са програми за инициализация и установяване (нулиране) на RAM паметта.
     goto  Rst         ; Следва инструкция за разрешаване на прекъсването (GIE) което не е трябвало да се изпълнява докато не са зададени началните
                       ; условия. Тъй като подпрограмата за прекъсване задължително е на адрес 004 изпълнението прескача това място - goto ...
;
Intrpt
     Org   0x4         ;
     movwf W_stack     ; С тези три последователни инструкции се запазват работния регистър W и статус регистъра S.Индексният регистър Fsr се използва
```

```
        swapf Status,w   ; в програмата за прекъсване, но не се използва в основната програма. Това е изключение, а като правило всички регистри (клетки от
        movwf S_stack    ; RAM) използвани в прекъсването трябва да се запазят при влизане в прекъсването и после да се възстановят (при излизане).

        bcf   IntCon,2   ; Тук се нулира флагът за прекъсване. Указва се директно като втори бит от IntCon защото не е дефиниран в началото. Ако флагът
                         ; не се нулира, след края на програмата за прекъсване отново ще се генерира такова. Прекъсване се изпълнява когато и докато
                         ; този флаг е в единица, независимо как е станало това.
        movlw OffInd     ; Тази част от прекъсването управлява индикацията: преди всичко се гасят всички индикатори (OffInd -> PortA), след това се задава
        movwf PortA      ; нова стойност на сегментите и  веднага се включва поредният индикатор. За да не се губи от яркостта на индикацията стойностите
        movf  Segm,w     ; на сегментите и  активния индикатор са подготвени предварително в края на предишното прекъсване. Така във всяко прекъсване се
        movwf PortB      ; индицира индикаторът подготвен в предното.
        movf  PortAm,w   ;
        movwf PortA      ;
        incf  Clock      ; Това е основният брояч на прекъсванията. С него се отчита и времето – т.е една секунда е равна на определен брой прекъсвания.
        btfsc But        ; Проверява се дали няма натиснат бутон.
        goto  CheckBt    ;
        andwf Buttons    ;
Int1    incf  Inds       ; Подготвя се следващият индикатор.
        movlw h'03'      ; Индикаторите са 4. Затова регистърът с номера на поредния индикатор Inds се "маскира" с 03 (0b11).
        andwf Inds       ; Така валидни стойности са само 0, 1, 2 и 3 (двоично 00, 01, 10 и 11).
        movf  Inds,w     ;
        call  TblInds    ;
        movwf PortAm     ; Номерът на индикатора се запомня в регистъра PortAm и ще се използва при следващото прекъсване.
        movlw Dig0       ; В клетки Dig0-Dig3 се намира комбинацията от сегменти за всеки от индикаторите. За зареждане на поредния индикатор се
        addwf Inds,w     ; използва индексна адресация - с помощта на индексния регистър Fsr и регистъра с номера на поредния индикатор Inds.
        movwf Fsr        ;
        movf  Indf,w     ;
        movwf Segm       ;
;
        swapf S_stack,w  ; Излизане от прекъсването. Следващите  последователни инструкции възстановят състоянието  на работния регистър w
        movwf Status     ; и на статус регистъра s. Използва се инструкция swapf защото при изпълнението на тази инструкция не се променят
        swapf W_stack    ; флаговете в статус регистъра!
        swapf W_stack,w  ;
        retfie           ; С тази команда завършва изпълнението на програмата за прекъсване и работата на процесора продължава от мястото където е
                         ; постъпило прекъсването.

CheckBt                  ; Подпрограмата  е част от прекъсването и се изпълнява, ако входът за бутоните е във високо ниво, т.е има натиснат бутон. В този
        xorlw h'ff'      ; момент във w регистъра се показва кой индикатор е включен в момента - съответният бит е 0. С командата xorlw h'ff' регистърът
```

```
      iorwf Buttons     ; се инвертира и при изпълнение на командата iorwf в регистъра за бутоните флагът на съответния бутон се установява в 1.
      goto  Int1        ; Работата се връща към програмата за прекъсване.
;
;                       ; Това е продължението на основната програма. За да бъде началното състояние след Reset дефинирано, се нулира флагът за
Rst   bcf   m_s         ; индикация на минути/секунди. Когато флагът е нула се индицират часове и минути, а когато е 1 - минути и секунди. Така винаги
                        ; след включване на захранването на индикацията ще се показват часове и минути.

Rst1  movf  ButtonF,w   ; Това е началото на основния цикъл на програмата. В нея се проверява състоянието на бутоните и се отчита времето. С първите
      xorwf Buttons,w   ; четири инструкции се проверява дали има промяна в състоянието на бутоните, т.е дали някой от бутоните е натиснат или отпуснат.
      btfss Z           ; Ако има промяна се изпълнява подпрограма за анализ.
      call  ButTest     ; Функцията на бутона се изпълнява само при натискане.
      movlw -d'128'     ; Проверява се дали е дошло време за добавяне на една секунда към времето. Една секунда се получава на всеки 256x8x128x2 такта -
      addwf Clock,w     ; така е настроено прекъсването от таймера- осемразряден брояч (256) по прескалер с коефициент на делене 8, т.е общо 524288. Това
      btfss C           ; е при осцилатор около 2,1 MHz. Ако се работи с кварцов резонатор на честота 4194304 (специални кварцове за часовници)
                        ; прескалерът се настройва да дели на 16. Така общо коефициентът става 1048576 (системният такт е fосц./4)
      goto  Point       ; Ако не са минали 128 прекъсвания се отива към подпрограма за управление на мигането на  точката между часове и минути.
      movlw -d'128'     ;
      bcf   GIE         ; Когато се извършват операции с регистри  които се използват и в прекъсването, е по-добре то да се забрани докато се изпълни
      addwf Clock       ; операцията. В този случай от Clock се вади 128 (т.е добавя се -128) и с команди bcf GIE и bsf GIE се забранява и разрешава
      bsf   GIE         ; прекъсването.
      movlw b'000100'   ; С помощта на флаг (AddCl = Flags,2) подпрограмата за добавяне на секунда Add1s се изпълнява веднъж на две преминавания
      xorwf Flags       ; през този клон. Така се реализира горната формула 128x2 (ако се работи с 256 вместо 2x128 програмата се усложнява – 256 е число
      btfsc AddCl       ; което се описва с два разряда).
      call  Add1s       ;
      goto  Rst1        ;
Point movlw -d'128'/3   ; Подпрограма  за изключване на точката. Точката се включва в началото на всяка секунда и се изключва  след около 170 ms.
      addwf Clock,w     ; И  тук, както и по-горе когато се добавя секундата, се проверява дали клетката Clock е по-голяма от  зададената стойност, а не
      btfss C           ; дали е равна.  Този подход дори е усложнил проверката за цяла секунда = 128x2 вместо 256, т.е би могло да се проверява дали
      goto  Rst1        ; броячът е станал 0.Това е направено защото прекъсването има приоритет  и ако се проверява за  точно определена стойност на
                        ; Clock има опасност да се пропусне моментът - при изпълнение на по дълга подпрограма може да настъпят две и повече
                        ; прекъсвания преди да се премине през клона на програмата където се проверява състоянието на брояча. Така ще се пропусне
                        ; секунда. Ако външните програми се изпълняват изцяло за времето между две прекъсвания може да се работи и по-опростено.
      btfsc m_s         ; Точката се изключва само в режим на индициране на часове-минути, и свети непрекъснато при минути-секунди.
      goto  Rst1        ;
      movlw H'80'       ; Гасенето се извършва като най-старшият бит на съответния разряд се установи в 1 (сегментите светят когато съответният бит е 0)
      iorwf Dig2        ;
      goto  Rst1        ; Управлението се предава в началото на основния цикъл.
```

```
; **********************
; *       SubRout
; **********************
;
IniPic                  ; Програма за инициализация на контролера. Задава  кои изводи са входове и кои изходи, режимите на таймера, прекъсването и т..н
      movlw h'10'       ; Pa0-3 =>Out,
      tris  PortA       ; Pa4 =>In
      movlw OffInd      ; Изключват се индикаторите за да не светят докато започне нормално изпълнение. В конкретния случай това не е много важно но
      movwf PortA       ; при някои устройства началната инициализация отнема няколко секунди и в това време е желателно индикацията да не се вижда.
      movlw 0           ; Pb0-7 =>Out
      tris  PortB
      movlw OffSeg
      movwf PortB       ; Изключват се сегментите
      movlw b'11010010' ; В Option регистъра се задава насочване на прескалера към таймера и коефициент на делене 8.
      option            ;
      movlw b'00100000' ; Прекъсване се разрешава само от  таймера.  Общото прекъсване се управлява с флаг  GIE в главната програма.
      movwf IntCon      ;
      return

ClrRAM                  ; Програма за нулиране на RAM-а. Препоръчва се за да може контролерът да работи при еднакви начални условия. Това е важно
      movlw #Clock      ; най-вече при настройката на програмите, за да се повторят “ефектите”.
      movwf Fsr         ; Нулирането става с индексна адресация с помощта на индексния регистър Fsr.
      movlw #d'20'      ; Двадесет клетки след Clock ще се нулират.
      movwf Numb        ;.
      movlw 0           ;
Clr1  movwf Indf        ; Записът в адрес 0 (с командата movwf  Indf), всъщност записва на адреса указан в Fsr. Тъй като този регистър се променя при
      decfsz Numb       ; изпълнението на цикъла Clr1 клетките се нулират последователно една след друга докато се нулира броячът Numb
      incf  Fsr         ;
      goto  Clr1        ;
      return            ;

ButTest                 ; Тази програма се изпълнява, ако в основната програма е установено, че има промяна на състоянието на бутоните. Последователно
      btfss But0        ; се проверяват и четирите бутона.  Логиката и на четирите е подобна - ако е натиснат бутон се изпълнява неговата функция, ако не е
      goto  ClrBut0     ; се нулира флагът на бутона и се преминава към следващия. В тази програма флаговете на бутоните Butxf не се използват, но те са
      bsf   But0f       ; предвидени, ако е необходимо да се изпълняват функции с два бутона, т.е при задържан натиснат  бутон да се изпълни друга
      movlw b'00000001' ; функция при натискане на някой от другите бутони. Друга задача на тези флагове е когато бутоните са по-бавни, т.е има
      xorwf Flags       ; разтрептяване на контактите. Това се случва на практика при всички бутони и обикновено се приема, че времето за установяване е
      return            ; под 10ms. Ако периодът на сканиране на бутоните ( в случая това е периодът на динамичната индикация) е малък при първото
```

```
ClrBut0                 ; задействане на бутона  се установява съответния флаг. Ако и следващият път пак е задействан чак тогава се изпълнява функцията.
      bcf   But0f       ; Бутон 0 сменя показанието от часове-минути на минути-секунди и обратно. Става с xorwf Flags т.е при всяко натискане на
      btfss But1        ; бутона се инвертира най-младшият бит на Flags, а това е флагът m_s.
      goto  ClrB1
      bsf   But1f
      bcf   m_s         ; При сверяване на часове винаги се минава в режим на изобразяване на часове и минути.
      goto  AdHours     ; Бутонът е за добавяне на часове.
ClrB1 bcf   But1f       ;
      btfss But2        ;
      goto  ClrB2       ;
      bsf   But2f       ;
      bcf   m_s         ; При сверяване на минути се минава в режим на изобразяване на часове и минути.
      goto  AdMin       ; Бутонът е за добавяне на минути.
ClrB2 bcf   But2f       ;
      btfss But3        ; Бутон 3 е за нулиране на секундите.
      goto  ClrB3       ;
      bsf   But3f       ;
      bsf   m_s         ; Когато се нулират секундите индикацията минава в режим минути-секунди.
      clrf  Sec1        ; Бутонът е за нулиране на секундите и на всички предварителни делители. В случая това са Sec1, Sec10 и Clock.
      clrf  Sec10       ;
      clrf  Clock       ; За да се усети ефектът от сверяването веднага, индикацията се презарежда веднага след натискане на бутона (LoInd). В противен
      goto  LoInd       ; случай ще се индицира 01, защото индикацията се презарежда на всяка нова секунда т.е едва след като изтече първата секунда.
ClrB3 bcf   But3f       ;
      Return            ;

AdjTime                 ; Това е част от програмата (Add1s) която се изпълнява всяка секунда. В тази част (AdjTime)се изважда (би могло и да се добавя) по
      btfss AdjT        ; една секунда на кръгли часове. Това става когато е установен флагът AdjT. Такава корекция е необходима когато от тактовата
      goto  LoInd       ; честота на процесора не може с целочислено делене да се получи една секунда. Например при кварц 32 768 Hz тактовата честота е
;     bcf   AdjT        ; 8192 Hz (осцилаторната честота се дели на 4). За да не трепка индикацията обаче честотата на опресняване трябва да е поне 45-50
;     decf  Sec1        ; Hz за всеки индикатор, т.е за четирите индикатора прекъсването трябва да е с честота 180-200 Hz. Точно 8192 се получава от
      goto  LoInd       ; 64x128, но в този случай честотата е ниска и индикацията ‘трепка’. Затова се избира друго произведение - 42x195=8190 при
                        ; което обаче се налага корекция. В този вариант на програмата, тъй като се работи с друга тактова честота (2MHz), не се налага
                        ; тази корекция - изваждането на секунда decf Sec1 е “коментирано” и не се асемблира.
Add1s incf  Sec1        ; Програмата за добавяне на секунда работи като десетичен брояч - секундите се увеличават с единица, а когато се получи 10 –
      movf  Sec1,w      ;  те се нулират, а следващият разряд се увеличава с единица. Проверката за това дали секундите са станали 10 става като към тях се
      addlw -d'10'      ; добави -10, ако резултатът е 0 ( флаг Z е вдигнат) секундите се нулират и се увеличава следващият разряд.
      btfss Z           ;
```

```
      goto  AdjTime     ; Когато резултатът не е 0 се проверява дали не предвидена корекцията описана по-горе - AdjTime.
      clrf  Sec1        ;
      incf  Sec10       ;
      movf  Sec10,w     ; Проверката за десетки секунди е подобна, но тук се проверява дали е достигната стойност 6 (60 s) и ако е достигната се увеличава
      addlw -d'6'       ; разрядът за минутите, а десетките секунди се нулират.
      btfss Z           ;
      goto  LoInd       ;
      clrf  Sec10       ;
AdMin incf  Min1        ; При минутите се работи както при секундите - младшите минути броят до 10, а старшите до 6.
      movf  Min1,w      ;
      addlw -d'10'      ;
      btfss Z           ;
      goto  LoInd       ;
      clrf  Min1        ;
      incf  Min10       ;
      movf  Min10,w     ;
      addlw -d'6'       ;
      btfss Z           ;
      goto  LoInd       ;
      clrf  Min10       ;
AdHours                 ; При часовете броенето е малко по-сложно защото трябва да се работи до 24, т.е веднъж младшите часове броят до 10, веднъж до 4.
      incf  Hours1      ; Старшите часове имат три разрешени стойности - 0, 1 и 2.
      movf  Hours1,w    ;
      addlw -d'10'      ;
      btfss Z           ; Ако е достигната стойност 10 е сигурно, че младшите часове трябва да се нулират, а старшите да се увеличат с 1. Ако обаче тази
      goto  Addh1       ; стойност не е достигната в Addh1 се проверява дали не е достигната стойност 4.
      clrf  Hours1      ;
      incf  Hours10     ;
      goto  SetAjT      ;
Addh1 addlw d'6'        ; Проверката за достигане на 4 става като се добави 6 (преди това от младшите часове е извадено 10). Ако резултатът е 0 (т.е
      btfss Z           ; младшите часове са 4) се проверява дали старшите часове са 2, т.е дали е достигната стойността 24 ч.
      goto  SetAjT      ;
      movf  Hours10,w   ;
      addlw -d'2'       ;
      btfss Z           ;
      goto  SetAjT      ;
      clrf  Hours1      ; Ако е достигната стойността 24 броячите за часовете се нулират. Така на индикацията никога не се показват стойности 60 за
```

```
      clrf  Hours10     ; минути и секунди и 24 за часовете защото индикацията се зарежда след изпълнение на операциите.
SetAjT
      movf  Hours10     ; Корекцията в броенето става когато старшите часове не са 0, т.е на всеки час от 10 до 23 се отнема по една секунда. Това става с
      btfss Z           ; вдигане на флага AdjT, който се анализира всяка секунда различна от 0. Ефектът е, че на всеки кръгъл час (10-23) "нулевата"
      bsf   AdjT        ; секунда е с двойна дължина. Както бе обяснено по-горе в този вариант на програмата корекция не е необходима и не се прави.

LoInd
      btfss m_s         ; Зареждането на индикацията става веднага след изпълнението на програмата за добавяне на секунда. Така всички промени веднага
      goto  Lo_h_m      ; се виждат. По тази причина при сверяването LoInd се извиква самостоятелно за да се усети ефектът от работата с бутоните. В
      movf  Sec1,w      ; зависимост от състоянието на флаг m_s в Dig0-Dig3 се зареждат секунди и минути или се отива към Lo_h_m където се зареждат
      call  TblSegm     ; минути и часове. Самото зареждане става с извикване на подпрограма TblSegm. В нея се реализира типичния подход при този тип
      movwf Dig0        ; контролери за четене на константи от таблици – при извикване на подпрограмата във w регистъра е отместването на константата от
      movf  Sec10,w     ; началото на таблицата, а при връщане в него е константата. В случая константите показват състоянието на сегментите за всяка
      call  TblSegm     ; цифра.
      movwf Dig1
      movf  Min1,w
      call  TblSegm
;     iorlw h'80'       ; С тази команда се "запалва" запетайката. При общ катод с iorlw h'80', а при общ анод с andlw h'7f'. В случая е за общ анод.
      andlw h'7f'
      movwf Dig2
      movf  Min10,w
      call  TblSegm
      movwf Dig3
      return

Lo_h_m                  ; Тази подпрограма е аналогична на по-горната, но се индицират минути и часове вместо секунди и минути.
      movf  Min1,w
      call  TblSegm
      movwf Dig0
      movf  Min10,w
      call  TblSegm
      movwf Dig1
      movf  Hours1,w
      call  TblSegm
;     iorlw h'80'
      andlw h'7f'
      movwf Dig2
      movf  Hours10,w
      btfsc Z
      movlw h'f'
```

```
      call  TblSegm
      movwf Dig3
      return

;********************
TblInds                 ; При този тип процесори четенето на константи става като се извиква подпрограма която завършва с retlw xx – при тази команда
      addwf PCL         ; в регистъра w се записва xx. При извикването на подпрограмата, във w е отместването спрямо началото. Тъй като първата
                        ; инструкция в таблицата добавя w към програмния брояч, се “прочита” w-ти ред от таблицата..
      retlw b'11111110' ; Ниско ниво включва съответния индикатор
      retlw b'11111101'
      retlw b'11111011'
      retlw b'11110111'

TblSegm                 ; при извикването на тази подпрограма от таблицата се прочита комбинацията от сегменти съответстващи на съдържанието на w.
      addwf PCL
      retlw b'11000000' ; 0     Сегментите изписват съответните цифри. Нулите отговарят на светещите сегменти. Най-старшият бит е запетайката и
      retlw b'11111001' ; 1     е 1 – т.е не свети
      retlw b'10100100' ; 2
      retlw b'10110000' ; 3
      retlw b'10011001' ; 4
      retlw b'10010010' ; 5
      retlw b'10000010' ; 6
      retlw b'11111000' ; 7
      retlw b'10000000' ; 8
      retlw b'10010000' ; 9
      retlw b'10001000' ; A
      retlw b'10000011' ; B
      retlw b'11000110' ; C
      retlw b'10100001' ; D
      retlw b'10000110' ; E
      retlw b'11111111' ; F – всъщност не се изписва F, а се затъмнява индикацията

; **********************
      End
; **********************
```