Указания за задачи 1,2,3 и 4

Преди всичко трябва да се усвоят основните похвати при писане, асемблиране, програмиране, тестване с помощта на **MPLAB** и **PIC16F716**. Средата за програмиране **MPLAB** е инсталирана на компютрите в лабораторията. Може да се ползва и от www.Microchip.com.

В началото трябва да се създаде проект. Най-лесно е като се използва **`Project Wizart`**. Спазват се стандартните процедури при Windows програми. **Броят на знаците с които се описва достъпът до проекта трябва да е не по-голям от 62 знака.** Препоръчва се проектът да се разполага в поддиректория на директория **c:\uprpic**. Разполагането в Desktop не е много удачно защото вече са "изхабени" 35-40 знака - C:\Documents and Settings\user\Desktop\..., а са допустими 62!

След написване на програмата се проверява за грешки с командите в меню **`Projekt`** -> **`Make`** или **`Build All`**. Тези команди, като често ползвани, имат и съответни икони.

Като се отстранят грешките контролерът се програмира с програматор **PICKIT-2(3).**

Разяснения по програма kit_tmp.asm

Всяка програма започва с няколко задължителни директиви за асемблера:

```
list  p=16f716    ; указания за типа микроконтролер
__config h'3ffb   ;  задаване на конфигурационната дума (две долни тиретаconfig)
```

Конфигурационната дума задава типа на осцилатора, работата на **`WDT`** (кучето), някои параметри на Reset схемата и защитата на програмата от прочитане. При по-сложните контролери се задават и други параметри. Важно е, че тази дума се записва с програматора и **не може** да се променя при изпълнение на програмата. Ако не бъде дефинирана, тя остава в състояние **`h'3fff`**, контролерът ще работи според значението на битовете и най-вероятно програмата няма да работи. Значението на всеки бит е указано в документацията на контролера.

В началото на програмата трябва да се дефинират променливите и константите. Асемблерът предварително "не знае" кой е контролерът. Всички променливи трябва да бъдат указани. Тези които са свързани с контролера (таймер, програмен брояч, портовете, регистрите и отделните битове в тях и …) винаги се използват. Имената им са дадени в описанието на контролера (datasheet) и за да не се описват всеки път може да се ползва файл който да се включи в основния:

```
 ...
      include p16f716.inc    ; в този файл е описан контъролерът
 ...
```

Ако този файл не се използва, всички променливи (адреси) трябва да се опишат (дефинират):

```
...
INDF        equ   00    ; Във всяка програма тези
TIMER       equ   01    ; регистри са едни и същи за
PCL         equ   02    ; конкретния контролер
STATUS      equ   03
FSR         equ   04
PORTA       equ   05
PORTB       equ   06

INTCON      equ   0b
...
```

Някои програмисти предпочитат да не работят със стандартния файл **`*.inc`** защото ползват имена с които са свикнали. **Не се препоръчва** модифициране на **`*.inc`** файловете тъй като се губи унификацията. Може да се ползват собствени файлове, но с друго име.

Имената на основните регистри в **`*.inc`** файла **обикновено** са както в документацията (datasheet) на Microchip, но се случва да има несъответствия.

Другите променливи които се ползват също трябва да се дефинират. Обикновено това става в процеса на писане на програмата – когато потрябва променлива тя се дефинира. Дефинират се и константите. В случая това са константите за изключени индикатори и сегменти. **Дефинирането на константите не е задължително**. Ако вместо името на

константата се зададе нейната стойност, генерираният код ще е същия. С предварителното дефиниране, програмата по-лесно се чете и коригира. Когато се налага промяна, това се прави само при дефинирането, а не навсякъде където се ползва константата.

```
Numb        equ   20
xxx         equ   21
yyy         equ   22
zzz         equ   23
...         equ   24

OffInd      equ   b'1111'
OffSeg      equ   h'ff'
;****************
 ...
```

Регистрите с общо приложение (RAM), за PIC16F716, започват от **h'20'**. За други контролери началният адрес може да е друг – за PIC16x84 е **h'1c'**.

Директивата **ORG h'xx'** указва на асемблера от кой адрес да започне или (ако **ORG** не е в началото) продължи асемблирането. Ако липсва директива **ORG** асемблерът започва от **h'00'** по подразбиране. Микроконтролерът, обаче, **винаги започва изпълнението на програмата от адрес h'00'** затова на този адрес трябва да има смислена инструкция.

От листинга на програмата се вижда, че веднага след Reset (**h'00'**) се изпълняват две подпрограми. В първата се задават началните стойности на регистрите, а в другата се нулира RAM. Повечето от регистрите имат начални стойности (едни и същи след Reset) и ако те отговарят на необходимите може да се спести предефинирането. Тази "спестовност", може да създаде проблеми когато се премине към друг контролер с други Reset стойности.

В подпрограма **IniPic** се определя посоката на данните – кои изводи да са входове и кои изходи, задава се работа на таймера и т.н.

В случая **PORTB** се използва за управление на сегментите и всички изводи (**RB0-RB7**) са конфигурирани като изходи. Четири от изводите на **PORTA** (**RA0-RA3**) управляват транзисторите за индикаторите и са изходи, а **RA4** е за четене на бутоните и е вход. За да се укаже, че даден извод ще бъде изход, съответният бит в **TRISx** регистъра се установява в 0, а ако е вход – в 1.

Битовете в **option** регистъра задават режимите на таймера, коефициента на делене на предварителния делител (прескалер) и др. За да се зададат се ползва документацията на контролера PIC16F716.

Втората подпрограма **ClrRAM** е за нулиране на регистрите (паметта) които се използват от програмата. Нулират се само тези с общо приложение. Тези от **0** до **h'1f'** определят режимите на работата на контролера . При нулирането се използва индексна (с **FSR**) адресация. Операциите които се извършват с адрес **00h** (**INDF**), всъщност се изпълняват с адрес записан във **FSR**. Така, като се започне от клетка **Numb**, последователно се нулират 20 клетки. Тази подпрограма е дадена като пример за индексна адресация но и като препоръка RAM винаги да се нулира. Ако програмата е написана без грешки тя ще работи и без началното нулиране. Когато има грешки, за да се открият, е добре изпълнението да започва при еднакви условия – нулирана памет. В противен случай (при грешка в кода) ходът на изпълнение на програмата може да се влияе от стойността в някоя клетка, а тя да се променя след изключване на захранването.

Главната програма **Main** започва със задаване на състоянието на сегментите на първия индикатор, включване на индикатора и изчакване **T_us**. Следва вторият индикатор, третият и т.н. След това управлението се връща отново към първия – така работи динамичната индикация.

Изчакването (закъснение) **T_us** се реализира с цикли на контролера. Неговата тактова честота е ¼ от осцилаторната. За **KIT16F716** RC осцилаторът работи на около 2MHz.

Описаната програма е прекалено проста и няма реално приложение – на индикаторите се показва едно и също, а процесорът не може да прави нищо друго - само отброява цикли (време).

След като се напише, програмата се асемблира с командата **Build All** в **MPLAB**. Грешките се отстраняват последователно от начало, защото често една грешка води до съобщения за други и когато тя се отстрани се премахват и останалите. В полето **Output** на прозореца MPLAB се изписват грешките и техният тип. Бързо двойно щракване с мишката

върху някоя грешка показва мястото и във файла. Файл с който може да се програмира контролера (***.hex**) се генерира САМО след отстраняване на всички грешки.

Програмирането става с програматор PICKIT-2 или PICKIT-3. От него се получава и захранването за **KIT16F716**. Това става като се зададе Vdd=5V и Vdd=On. При успешно програмиране индикацията "светва". На всеки индикатор свети по един сегмент.

Прави впечатление, че освен зададените сегменти слабо "просветват" и сегменти които светят на съседните индикатори. Причина за това е, че при така написаната програма, по време на смяна на разряда, сегментите остават включени. За да се избегне това програмата се променя – преди смяна на сегментите се изключват всички разряди. Това става като се добавят следните два реда преди смяната на сегментите за **всеки** индикатор - на четири места.

```
movlw OffInd          ; за всички разряди състояние "изключено"
movwf PORTA           ;
movlw b'11110111'
movwf PORTB
movlw b'0111'
movwf PORTA
call  T_us
```

По-долу е дадена програмата ***KIT_tmp.asm*** с коментар. Такъв коментар **никога** не се прави от опитни програмисти, защото те знаят езика. Коментарът трябва да припомня алгоритъма на програмата, както и особености при реализацията.

В случая коментарът помага за научаване на инструкциите **(INSTRUCTION SET)**.

Добре е при писане да се ползва табулация, да се отделят етикетите (имената на под-програмите), командите и операндите. С точка-запетая се отделя коментарът.

```
; Ver. 10.2015
;
      list p=16f716         ; програмата е за контролер PIC16F716
      __config h'fffb'      ; config определя типа генератор и др.
      include p16f716.inc   ; в този файл са имената и адресите на
                            ; регистрите, флагове и др. на ...716
;
Numb        equ   20        ; дефинират се регистрите (клетки от RAM)

OffInd      equ   b'1111'   ; всички индикатори не светят (off)
OffSeg      equ   h'ff'     ; всички сегменти не свтят


;****************
      org    h'00'        ; указва на асемблера от къде в паметта
                          ; да е започне (продължи) програмата
Reset
      call   IniPic       ; подпрограма за инициализация на ...716
      call  ClrRAM        ; нулиране на RAM

Main
      movlw  b'11111110'  ; в W се зарежда константа
      movwf  PORTB        ; това което е в W отива в PORTB
      movlw  b'1110'      ; в W се зарежда константа
      movwf  PORTA        ; това което е в W отива в PORTA
      call   T_us         ; изпълнява се програма за закъснение

      movlw  b'11111101'  ; същото както по-горе с други
      movwf  PORTB        ; сегменти и друг индикатор
      movlw  b'1101'
      movwf  PORTA
      call   T_us

      movlw  b'11111011'  ; следващ индикатор
```

```
      movwf  PORTB
      movlw  b'1011'
      movwf  PORTA
      call   T_us

      movlw  b'11110111' ; четвърти индикатор
      movwf  PORTB
      movlw  b'0111'
      movwf  PORTA
      call   T_us

      goto   Main        ; връщане към първия индикатор – цикълът
                         ; се затваря
;**********************
;*       SubRout
;**********************
IniPic
      movlw  b'10000'    ; в W се зарежда константа
      tris   PORTA       ; W - в регистъра за посока на данните в PORTA
      movlw  OffInd      ; в W се зарежда константа -> индик.=Off
      movwf  PORTA       ; W в PORTA -> индикаторите се изключват
      movlw  b'00000000' ; задават се всички изводи на
      tris   PORTB       ; PORTB като изходи
      movlw  OffSeg      ; както по-горе но се изключват
      movwf  PORTB       ; всички сегменти
      movlw  b'11010010' ; в W се зарежда константа
      option             ; W в регистъра OPTION
      return             ; излизане от подпрограма

ClrRAM                   ; подпрограма за нулиране на RAM
      movlw  Numb        ; в W се зарежда константа = адреса на Numb
      movwf  FSR         ; FSR=W,така в индексния рег. е адр.на Numb
      movlw  d'20'       ; в W конст. = брой на клетки за нулиране
      movwf  Numb        ; в брояч Numb – броя на клетките
Clr1
      incf   FSR         ; увеличаване с 1 на регистър, така FSR вече
                         ; указва следващия адрес
      clrf   INDF        ; инструкция за нулиране. В случая INDF е
                         ; регистър за индексна адресация, т.е се
                         ; нулира регистър с адрес записан в FSR
      decfsz Numb        ; намаляване с единица на регистър, проверка
                         ; дали не е станал 0
      goto   Clr1        ; изпълнява се докато Numb не е 0
      return             ; излизане от подпрограма (Numb=0)

T_us                     ; подпрограма за програмно закъснение
      movlw  6           ;
      movwf  Numb        ; в Numb се зарежда числото 6
T_01
      decfsz Numb        ; Numb се намалява докато стане 0
      goto   T_01        ; decfsz и goto заемат 3 такта
      return             ; общо закъснението е 24 такта:
                         ; 2     – извикване на подпрограмата T_us
                         ; 1+1   – movlw и movwf
                         ; 18    – 6x3, 6 пъти се минава през цикъла
                         ; 2     – излизане от подпрограма
                         ; 24    – общо 24

      End                ; директива за край на асемблирането
;**********************
```

Времето през което всеки индикатор свети се задава от подпрограмата за закъснение **T_us**. Това закъснение се определя от стойността записана в **Numb**. В случая максималната стойност е **256x3x2us –> 1,5ms,** където **Numb** е един байт (256), всяко намаляване е три такта, а в случая периодът на тактовата честота на контролера е 2us (Tosc/4 като Tosc е около 2 MHz).

За да видим как работи динамичната индикация, т.е. как се сменят разрядите, превключването трябва да става по-бавно – закъснението да е над 20ms. За да направим толкова голямо закъснение програмно, трябва да използваме вложени цикли като подпрограмата **T_us** се замени с:

```
T_us  clrf      Numb1
      movlw 0
      movwf     Numb
T_01  call      T_02
      decfsz    Numb
      goto      T_01
      return

T_02  decfsz    Numb1
      goto  T_02
      return
```

Използва се втора променлива **Numb1** с която се реализира подпрограма за закъснение **T_02**. Така максималното закъснение се получава 256x256x3x2 -> около 400ms.

Друг начин за реализиране на закъснение е с използването на таймера **TMR0**. Този таймер има флаг който се установява в 1 когато таймерът се препълни, т.е когато премине от състояние **h'ff'** в **h'00'**. Таймерът е еднобайтов, т.е брои от 0 до 255. Закъснението се реализира като се чака таймерът да се препълни, като се следи флага **TMR0IF**. Този флаг е втори бит в регистъра **INTCON**, т.е **INTCON,2 = TMR0IF.**

```
T_us  movlw     0           ; 0 = 256
      movwf     Numb

T_01  btfss     INTCON,2
      goto      T_01
      bcf       INTCON,2
      decfsz    Numb
      goto      T_01
      return
```

В тази подпрограма в **Numb** се задава колко препълвания на таймера да е закъснението. Тук закъснението (в примера е максималното) се определя като (Numb) x256x2xN, където N е стойността на прескалера – предварителния делител към входа на таймера. Този делител се задава в **Option** регистъра и може да е 1,2,4,8 до 256 по степените на 2.

Желателно е таймерът да не ползва прескалера т.е делителят да е 1, а закъснението да се променя с числото записано в **Numb**.

Прескалерът обикновено се използва от таймера на "кучето". Така ще може да се задават по-големи интервали на задействане на **WDT** (таймер куче). Без прескалер времето в което трябва да се нулира **WDT** е само около 18ms – малко за нормално изпълнение на много програми, например за делене, умножение и т.н. Пренасочването на таймера към кучето става като в програмата **IniPic** се промени съдържанието на **option**:

```
IniPic
      ...
      movlw   b'00000000'   ; задават се всички изводи на
      tris    PORTB         ; PORTB като изходи
      movlw   OffSeg        ; както по-горе но се изключват
      movwf   PORTB         ; всички сегменти
      movlw  b'11011111'    ; в W се зарежда константа
      option                ; W в регистъра OPTION
      return                ; излизане от подпрограма
```

При този вариант тактовата честота на таймера ще е винаги основната (0,5MHz), а прескалерът ще се ползва от таймера на кучето.

Като правило програмите трябва да работят с включен **WDT** (куче). Това става като в конфигурационната дума се зададе съответният бит (2). Освен това се включва и още един таймер **PWRT** за закъснение след включване на захранването. Той се използва за осцилатори които "тръгват" бавно - като правило такива са кварцовите. Така директивата за конфигурационната дума става:

```
      ...
      list  p=16f716
      __config   h'fff7'     ; WDT и PWRTE са включени
;     __config   h'fffb'     ; WDT е изключен
      include    p16f716.inc
      ...
```

При така зададената конфигурационна дума **WDT** работи и контролерът ще се нулира (reset) периодично. Ако **OPTION** регистъра е **b'11011111'** това ще става през 2-3 секунди – може да се види като контролерът се програмира с дадените по-горе стойности на **OPTION** регистъра и на конфигурационната дума (с включен **WDT**). Вижда се, че периодично индикацията примигва – минава се през Reset. За да не става това в основната програма се включва командата за нулиране на **WDT clrwdt** . След като тази команда се включи в главната програма индикацията работи нормално.

```
      ...
      movwf      PORTA
      call       T_us

      clrwdt

      goto       Main
;**********************
;*        SubRout
      ...
```

Смисълът на използване на **WDT** е да се предизвика начално установяване при объркване и ненормална работа на програмата. Мястото в програмата където трябва да се нулира таймера (**WDT**) е много важно. От една страна трябва да е сигурно, че през този клон се преминава на интервали по-малки от времето за препълване на **WDT**. От друга страна дрябва да е сигурно, че ако програмата преминава през този участък тя не се е объркала. Затова не се препочъчва нулирането на кучето да се намира в подпрограми, особено в такива за прекъсване – програмата може да е "зациклила" на едно място, но да изпълнява прекъсванията и съответно да нулира **WDT**.

Следващата стъпка е използване на прекъсване за управление на индикацията. Така за редуване на разрядите ще се ползва само необходимото време, а интервалите ще се задават с таймера, който ще предизвиква прекъсване.

Прекъсванията в контролерите от тази серия може да се предизвикат по различен начин – от външни сигнали, от таймера и др. Винаги при прекъсване се запомня адресът след последната изпълнена инструкция (там където ще се върне изпълнението след края на прекъсването) и програмният брояч се зарежда с адрес **h'04'**. При тези контролери винаги, независимо кой е предизвикал прекъсването, то започва да се изпълнява от адрес 4. Затова при писане на програми за контролерите PIC16… в които има прекъсване се спазва следното:

1. В началото на програмата за прекъсване, ако има няколко източника на прекъсване се проверява кой го е предизвикал;

2. Тъй като изпълнението след Reset винаги започва от **h'00'**, адрес **h'04'** се прескача, т.е адресите 0, 1 и 2 може да се ползват, но на адрес 3 трябва да има **goto xxx**, като се отива към друга част на основната програма. В примера ***kit_int.asm*** това е показано.

Разяснения по програмата kit_int.asm

Програмата е подобна на ***kit_tmp.asm***, но индикацията се управлява с прекъсване и така времето извън прекъсването, в което се сменят индикаторите, може да се използва за друго.

В сравнение с ***kit_tmp.asm***, са добавени четири байта **Dig0-Dig4** в които се помни комбинацията от сегменти за всеки индикатор. Тези клетки се "зареждат" в основната програма, а се ползват в програмата която управлява динамичната индикация.

В **IniPic** са добавени два реда за определяне на източника на прекъсване:

```
...
option
movlw b'00100000'      ; задава се прекъсване от таймера
movwf INTCON           ;
return
...
```

В началото след **call IniPic** и **call ClrRAM** се разрешават прекъсванията – вдига се флаг **GIE -> bsf INTCON,GIE**.

Прескача се запазеното място за прекъсване **h'04' – goto Main**.

Програмата за прекъсване започва и завършва със задължителни операции за запомняне на състоянието на регистрите в момента в който то е настъпило. Не се запомнят всички регистри, а само тези които се променят при изпълнението на прекъсването. Почти винаги това са работният регистър **W** и **Status** регистъра. Много малко са операциите които не използват или не променят тези два регистъра. Ако някои от регистрите не се използват от други програми, а само в прекъсването, не е нужно те да се запомнят.

При възстановяване съдържанието на **W** регистъра се използва "странна" процедура:

```
...
movf       S_Stack,w
movwf      STATUS
swapf      W_Stack
swapf      W_Stack,w
retfie
...
```

При операция **swapf** не се променя **STATUS** регистъра, а той вече е възстановен след прекъсването и би се "развалил", ако например се ползва **movfw W_Stack,w** .

Същинската програма за обработка на прекъсването започва с нулиране на флага **TMR0IF** който го е предизвикал **-> bcf INTCON,2**. Ако това не се направи (допускана грешка) веднага след излизане от прекъсването то ще се изпълни отново и така безкрай.

Задачата на прекъсването е да управлява динамичната индикация - при всяко влизане изключва индикаторите, променя състоянието на буфера за сегментите (**PORTB**) и включва съответния индикатор. Когато управлението става последователно без прекъсване, както в ***kit_tmp.asm***, е ясно кои сегменти на кой индикатор съответстват. Сега обаче, една и съща програма трябва да редува индикаторите. Използва се брояч с четири различни състояния 0-3. Този брояч е **Inds** и числото в него показва кой индикатор трябва да се покаже. При всяко влизане в прекъсването броячът се увеличава с 1:

```
...
incf       Inds,w      ; с тези три реда Inds
andlw      h'03'       ; брои от 0 до 3
movwf      Inds        ;
...
```

За да се определи кой индикатор трябва да се включи се проверява състоянието на брояча, Ако е 3 – се зарежда трети (**Ld_Ind3**), ако не е – проверява се за 2 и т.н. Проверка за последния не се прави защото, ако не е бил 3,2,1 е 0.

```
...
movlw 3            ; проверка дали е 3
subwf Inds,w       ; ако е 3 резултатът е 0
btfsc STATUS,Z     ; Z флагът е вдигнат и се
goto  Ld_Ind3      ; изпълнява съответната
                   ; подпрограма, ако не е, се
movlw 2            ; минава към проверка за 2
```

```
            subwf Inds,w
            btfsc STATUS,Z
            goto  Ld_Ind2

            movlw 1
            subwf Inds,w
            btfsc STATUS,Z
            goto  Ld_Ind1     ; не се проверява дали е 0
                              ; защото, ако не е 3,2,1 е 0

Ld_Ind0     movlw b'1110'     ; нулата задава се кой индикатор да свети
            movwf PortA       ; включва се, но още не свети защото
            movfw Dig0        ; сегментите са изключени
            goto  Ld_Sgm

Ld_Ind3     movlw b'0111'
            movwf PortA
            movfw Dig3
            goto  Ld_Sgm
...         ...
...         ...
Ld_Sgm      movwf PORTB
...         ...
```

Проверката за съответствие става с изваждане - дали резултатът е нула. Ако не е, се отива към следващата проверка, а ако съвпада се зарежда съответната комбинация от сегменти.

Тази програма е лесна за разбиране но има някои недостатъци. Времето за смяна на индикаторите зависи от техния номер – нулевият индикатор свети най-кратко защото е угасен докато се проверява дали **Inds** не е 3,2,1…

Това може и да не се забележи при големи интервали, но въпросът е принципен. При някои програми времето за изпълнение на прекъсването е много важно и от него може да зависи точността на измерване.

Освен това програмите в прекъсването трябва да са възможно най-кратки.

Вместо този подход може да се ползва една възможност на контролера PIC16… Ако към програмния брояч **PCL** се добави число програмата ще “скочи” на ново място. Ако прибавим 0 ще изпълни следващия ред, ако прибавим 1 – през един, при 2 – през два и т.н. Пример за това е даден по-долу:

```
        ...
        andlw   h'03'           ; брои от 0 до 3
        movwf   Inds            ;

        addwf   PCL             ; към програмният брояч се добавя W=Inds
        goto    Ld_Ind0         ; от номера на индикатора
        goto    Ld_Ind1         ; се определя къде
        goto    Ld_Ind2         ; да "скочи" програмата
      ; goto    Ld_Ind3         ; този ред е излишен защото скокът
                                ; трябва да е към следващия
Ld_Ind3
        movlw   b'0111'         ; нулата задава се кой
        ...
```

Вместо многото проверки и 12 реда код са използвани 5. Разликата не е само в това. Сега времето за изпълнение е едно и също – скокът е винаги 2 такта независимо дали се скача не следващия ред или на последния.

Всяка програма се оценява по няколко критерия:

1. Дали работи правилно (винаги има неоткрити грешки…);
2. Дали работи бързо, т.е колко време отнема на процесора;
3. Колко програмна памет заема.

Последното е важно за програмиране на вградени системи, на евтини устройства с малък ресурс. Обикновено критерии 2 и 3 трудно се изпълняват едновременно – търси се компромис.

Разбира се има и други показатели за оценка на програмите – лесно ли се настройват и променят, дали са преносими, използват ли стандартни (и вече проверени) модули.

Горната програма не отговаря на изискването за икономия на памет. Вижда се, че модулите за зареждане на сегментите се повтарят за всеки индикатор. Ако се увеличи броят на индикаторите програмата "ще набъбне".

Вариантите за написване на програмата са много и всеки би написал собствена версия. Ето една която до голяма степен изпълнява дадените по-горе критерии:

```
      ...
      bcf   INTCON,2    ; задължително се нулира флагът който е предизвикал
                        ; прекъсването - иначе то ще се изпълнява безкрай!

      movlw OffSeg      ; изключване на индикаторите
      movwf PORTB       ; за да не се вижда смяната

      incf  Inds,w      ; с тези три реда Inds
      andlw h'03'       ; брои от 0 до 3
      movwf Inds        ;
                        ; чрез номера на индикатора, от таблицата се
      call  TblInds     ; прочита състоянието на изходите. Може да стане и
      movwf PORTA       ; програмно но така програмата е по-кратка.

      movlw Dig0        ; W се зарежда с константа = адреса на Dig0
      addwf Inds,w
      movwf FSR         ; сега FSR "сочи" сегментите на индикатора който
      movf  INDF,w      ; ще се индицира при следващото прекъсване.
      movwf PORTB       ; Dig[0-3] се прехвърля в регистъра за сегменти.

EndInt
      movf  FSR_Stack,w      ; Възстановяват се регистрите в състоянието в
      ...
```

В сравнение с досегашната програма определянето на индикатора който ще се индицира става като се използва таблица за включване на анодите и индексна адресация за.

`call TblInds` предава управлението на следната таблица която е част от програмата:

```
      ...
TblInds addwf   PCL
        retlw   b'11111110'      ; индикатор 0
        retlw   b'11111101'      ; индикатор 1
        retlw   b'11111011'      ; индикатор 2
        retlw   b'11110111'      ; индикатор 3
        ...
```

На първия ред от **`TblInds`** към програмния брояч **`PCL`** се добавя **`W`**, а в **`W`** е номерът на индикатора. Така при 0 се изпълнява следващият ред, при 1 вторият, при 2 третият и т.н. С **`retlw`** изпълнението се връща там откъдето е извикана подпрограмата, като във **`W`** е заредено съответното число (**`b'11111110'`**). В конкретния случай с 0 се задава кой индикатор да е включен (свети).

В програмният модел на PIC16ьххх не е предвидено директно четене на програмната памет. Това става по посочения по-горе начин – с извикване на подпрограма и връщане от нея със зареден регистър **`W`**.

Когато се пише подпрограмата за индикация с прекъсване трябва да се има предвид, че едновременно с това може да се проверява дали има натиснат бутон.

Сорсът на програмата е даден по-долу:

```
;   Interrupt
;   Filename:   kit-int.asm
;   Date:       10.2015
;   File Ver.:  1.00
;
    list   p=16f716
    __config    h'fff7'    ; WDT и PWRTE са включени
;   __config    h'fffb'    ; WDT е изключен, редът е "коментиран"
;   CP=Off, PWRTE=On, WDT=On, OSC=RC
    include     p16f716.inc
;
;
Numb  equ   20

Dig0  equ   25
Dig1  equ   26
Dig2  equ   27
Dig3  equ   28
Inds  equ   29

W_Stack    equ   2d
S_Stack    equ   2e
FSR_Stack  equ   2f

OffInd     equ   b'1111'
OffSeg     equ   h'ff'

;****************
    org    h'00'      ; Начало
;****************
Reset
    call   IniPic
    call   ClrRAM
    bsf    INTCON,GIE ; след подготвителните операции се разрешават
                      ; прекъсванията зададени в INTCON.
    goto   Main       ; прескача се областта заделена за прекъсвания!

;***************
;   Interrupt ;
    org    h'04'      ; ! Всички прекъсвания на тази серия контролери се
; насочват към този адрес. Тук се разпознава от къде е прекъсването
; и пренасочва управлението.
; Тук са разрешени прекъсвания само от таймера, няма какво да се разпознава

    movwf  W_Stack    ; запомнят се всички регистри които се ползват в
    movf   STATUS,w   ; подпрограмата за прекъсване
    movwf  S_Stack    ; W и Status регистрите се използват почти винаги
    movf   FSR,w
    movwf  FSR_Stack

    bcf    INTCON,2   ; задължително се нулира флагът който е предизвикал
                      ; прекъсването - иначе то ще се изпълнява безкрай!

    movlw  OffSeg     ; изключване на индикаторите
    movwf  PORTB      ; за да не се вижда смяната

    incf   Inds,w     ; с тези три реда Inds
    andlw  h'03'      ; брои от 0 до 3
    movwf  Inds       ;
                      ; чрез номера на индикатора, от таблицата се
```

```
    call   TblInds     ; прочита състоянието на изходите. Може да стане и
    movwf  PORTA       ; програмно но така програмата е по-кратка.

    movlw  Dig0        ; W се зарежда с константа = адреса на Dig0
    addwf  Inds,w
    movwf  FSR         ; сега FSR "сочи" сегментите на индикатора който
    movf   INDF,w      ; ще се индицира при следващото прекъсване.
    movwf  PORTB       ; Dig[0-3] се прехвърля в регистъра за сегменти.

EndInt
    movf   FSR_Stack,w      ; Възстановяват се регистрите в състоянието в
    movwf  FSR         ; което са били при възникване на прекъсването.
    movf   S_Stack,w
    movwf  STATUS
    swapf  W_Stack
    swapf  W_Stack,w
    retfie

;***************
;  Основна програма
;
Main
    movlw  b'11111110'      ; Тази програма зарежда в сегментите за
    movwf  Dig0             ; съответния индикатор някаква комбинация

    movlw  b'11111101'
    movwf  Dig1

    movlw  b'11111011'
    movwf  Dig2

    movlw  b'11110111'
    movwf  Dig3

    clrwdt                  ; WDT е включен и трябва да се нулира

    goto   Main
;**********************
;*       SubRout
;**********************
IniPic
    movlw  b'10000'
    tris   PORTA
    movlw  OffInd
    movwf  PORTA
    movlw  b'00000000'
    tris   PORTB
    movlw  OffSeg
    movwf  PORTB
    movlw  b'11011111'      ; WDT е включен и трябва да се нулира
    option
    movlw  b'00100000'      ; задава се прекъсване по таймера, но GIE ще
    movwf  INTCON           ; разреши всички прекъсванията по-нататък
    return

ClrRAM
    movlw  Numb
    movwf  FSR
    movlw  d'20'
    movwf  Numb
Clr1
```

```
    incf   FSR
    clrf   INDF
    decfsz Numb
    goto   Clr1
    return

;***
TblInds
    addwf  PCL
    retlw  b'11111110'      ; индикатор 0
    retlw  b'11111101'      ; индикатор 1
    retlw  b'11111011'      ; индикатор 2
    retlw  b'11110111'      ; индикатор 3
;***
TblSegm                     ; таблица в която всяка цифра е кодирана с
    addwf  PCL              ; комбинация от сегменти
    retlw  b'11000000'      ; 0
    retlw  b'11111001'      ; 1
    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 – не се индицира F, а се използва за гасене
                            ; на индикацията
    End
;**********************
```

Подходът за управление на индикацията с прекъсване е по-удобен в сравнение с този в ***kit_tmp.asm*** защото равномерността на превключване се гарантира и не е необходимо временните интервали да се задават програмно – така не се губи време.

Основната програма се "грижи" да зададе сегментите в **Dig0-Dig3** когато е необходимо, а прекъсването изпълнява алгоритъма за динамична индикация.

Външно (като се наблюдава работата им) програмите ***kit_tmp.asm*** и ***kit_int.asm*** се държат еднакво, но тази с прекъсване може да се доразвива без да е необходимо да се променя съществуващата част.

Задачите по KIT_int.asm са:

1. Да се проследи цялата програма, ред по ред. На всяка стъпка да е ясно какво прави процесорът.
2. Да се въведе и провери работата на тази програма.
3. Да се промени времето през което се извиква прекъсването чрез презареждане на таймера. Това означава в началото на прекъсването да се зададе начална стойност. Таймерът без задаване на начална стойност таймерът брои до 256 и през толкова такта настъпва прекъсване. Чрез презареждане на таймера предизвикването на прекъсване може да се ускори, но не може да се забави (таймерът е осем разряден и 255 е максималната стойност).
4. Да се реализира динамична индикация с по-ниска честота. Доколкото максималният коефициент на делене на таймера е 256, това може да стане като подпрограмата за управление на индикацията се изпълнява веднъж на няколко прекъсвания. За целта програмно се прави брояч.

Разяснения по програмата KIT_but.asm

От схемата на свързване на бутоните и индикаторите се вижда, че и бутоните трябва да се четат "динамично' – т.е синхронно с индикаторите. Дали един бутон е включен може да се провери само, ако съответният индикатор е включен. Проверката става в подпрограмата за прекъсване, непосредствено след като са заредени анодите:

```
    . . .
    . . .
    call    TblInds     ; прочита състоянието на изходите.Може да стане
    movwf   PORTA       ; и програмно но така програмата е по-кратка.
                        ; бутоните се проверяват заедно с индикаторите
    btfsc   But         ; дали е натиснат бутона (RA4=1?)
    goto    CheckBt     ; натиснат е -> към CheckBt
    andwf   Buttons     ; не е натиснат -> нулира флага на бутона
    goto    Int1

CheckBt
    xorlw   h'ff'       ; когато бутонът е натиснат съответният
    iorwf   Buttons     ; флаг се установява в 1

Int1
    movlw   Dig0        ; W се зарежда с константа = адреса на Dig0
    addwf   Inds,w
    . . .
    . . .
```

Използва се последният вариант на програмата за управление на индикацията с прекъсване. След като се зададат анодите с **movwf PORTA**, в **W** има една нула (светещият индикатор), останалите битове са във високо ниво 1. С операции AND и OR може да зададем бита на съответния бутон. Ако няма натиснат бутон с **andwf Buttons** нулираме бита. Ако бутонът е натиснат, с **iorwf** установяваме съответният бит в 1. Преди това инвертираме **W** (нулите единици, единиците нули).

За да помним състоянието на бутоните дефинираме клетка **Buttons** в която първите четири бита показват състоянието на съответните бутони. За да разберем дали всичко е наред, в основната програма, там където задаваме състоянието на сегментите добавяме следното:

```
    . . .
    . . .
    Main
       movlw b'11111110'  ; Тази програма зарежда в сегментите за
       btfsc Buttons,0    ; проверяваме дали бутонът е натиснат
       andlw b'01111111'  ; ако е натиснат - вкл. сегмент точка
       movwf Dig0         ; съответния индикатор някаква комбинация
    . . .
    . . .
```

С тези два реда, при натиснат бутон на първия индикатор, съответната десетична точка трябва да свети. За другите индикатори може да се направи същото.

При проверка на работата, при някои модули, се вижда, че светят по две точки. Това не е грешка в програмата, а особеност на изключването на транзисторите в анодите – те изключват бавно. За да може да изключи трябва време – в програмата преди проверката на бутоните преместваме зареждането на сегментите – показано е в следващия сорс. Използваме помощна клетка **Temp** от паметта в която да запомним състоянието на **PORTA**. След тези корекции програмата работи нормално.

Друга промяна би следвало да направим в основната програма **Main**. Вместо да зареждаме сегментите може да ползваме комбинациите дадени в **TblSegm**. По-долу е даден сорс на описаните промени:

```
...
 andlw h'03'      ; брои от 0 до 3
 movwf Inds       ;
                  ; чрез номера на индикатора, от таблицата се
 call  TblInds    ; прочита състоянието на изходите. Може да стане и
 movwf PORTA      ; програмно но така програмата е по-кратка.
 movwf Temp       ; ще се използва по-късно

 movlw Dig0       ; W се зарежда с константа = адреса на Dig0
 addwf Inds,w
 movwf FSR        ; сега FSR "сочи" сегментите на индикатора който
 movf  INDF,w          ; ще се индицира при следващото прекъсване.
 movwf PORTB      ; Dig[0-3] се прехвърля в регистъра за сегменти.

 movf  Temp,w     ; в Temp е състоянието на PORTA
 btfsc But        ; дали е натиснат бутона (RA4=1?)
 goto  CheckBt    ; натиснат е -> към CheckBt
 andwf Buttons    ; не е натиснат -> нулира флага на бутона
 goto    Int1
CheckBt
 xorlw   h'ff'    ; когато бутонът е натиснат съответният
 iorwf   Buttons  ; флаг се установява в 1
Int1
...
...
Main
 movlw 5
 call  TblSegm    ; от TblSegm прочитаме комбинацията за цифра 5
 btfsc Buttons,0
 andlw b'01111111'
 movwf Dig0       ; съответния индикатор някаква комбинация
...
```

Друг вариант за бързо изключване на индикатора, за да се избегне ефекта с двете точки, е като се промени малко програмата в прекъсването. В сегашния вариант за да няма просветване от съседния индикатор се изключват сегментите, след това се задава новият индикатор и след това се включват неговите сегменти. Може да се направи, със същият ефект като първо се изключат индикаторите, след това се сменят сегментите и накрая се включи съответния индикатор. Така се ускорява изключването на предишния индикатор.

```
...
...
 bcf   INTCON,2   ; задължително се нулира флагът който е предизвикал
                  ; прекъсването - иначе то ще се изпълнява безкрай!

 incf  Inds,w     ; с тези три реда Inds
 andlw h'03'      ; брои от 0 до 3
 movwf Inds       ;

 movlw OffInd     ; изключване на индикаторите
 movwf PORTA      ; за да не се вижда смяната

 movlw Dig0       ; W се зарежда с константа = адреса на Dig0
 addwf Inds,w
 movwf FSR        ; сега FSR "сочи" сегментите на индикатора който
 movf  INDF,w     ; ще се индицира при следващото прекъсване.
 movwf PORTB      ; Dig[0-3] се прехвърля в регистъра за сегменти.

 movf  Inds,w     ; чрез номера на индикатора, от таблицата се
 call  TblInds    ; прочита състоянието на изходите. Може да стане и
```

```
 movwf PORTA      ; програмно но така програмата е по-кратка.

 btfsc But        ; дали е натиснат бутона (RA4=1?)
 goto  CheckBt    ; натиснат е -> към CheckBt
 andwf Buttons    ; не е натиснат -> нулира флага на бутона
 goto    Int1
CheckBt
 xorlw   h'ff'    ; когато бутонът е натиснат съответният
 iorwf   Buttons  ; флаг се установява в 1
Int1

EndInt
...
...
```

При всички досега разгледани варианти времето за смяна на индикаторите, през което те не светят, не е малко. В последната версия е около 12 такта. Това време, особено когато се работи с ниска честота на осцилатора трябва да е минимално.

Портовете **PORTA** и **PORTB** може да се сменят бързо с предварително подготвени данни които са запомнени в две помощни клетки **Segm** и **PortAm**:

```
...
 movlw OffInd
 movwf PortA      ; изключват се всички индикатори
 movf  Segm,w
 movwf PortB      ; зареждат се новите сегменти
 movf  PortAm,w
 movwf PortA      ; включва се новият индикатор
...
```

Така времето за превключване винаги е 4 такта, като няма просветване между съседните индикатори. А двете помощни клетки се зареждат след това:

```
 ...
 andlw h'03'      ; брои от 0 до 3
 movwf Inds       ;

 movlw OffInd     ; изключване на индикаторите
 movwf PORTA      ; за да не се вижда смяната
 movf  Segm,w     ; с предварително подготвени
 movwf PORTB      ; данни в Segm и PortAm се
 movf  PortAm,w   ; зареждат новите сегменти и
 movwf PORTA      ; се включва новият индикатор

 btfsc But        ; дали е натиснат бутона (RA4=1?)
 goto  CheckBt    ; натиснат е -> към CheckBt
 andwf Buttons    ; не е натиснат -> нулира флага на бутона
 goto  Int1
CheckBt
 xorlw   h'ff'    ; когато бутонът е натиснат съответният
 iorwf   Buttons  ; флаг се установява в 1
Int1
 movlw Dig0       ; W се зарежда с константа = адреса на Dig0
 addwf Inds,w
 movwf FSR        ; сега FSR "сочи" сегментите на индикатора който
 movf  INDF,w     ; ще се индицира при следващото прекъсване.
 movwf Segm       ; Dig[0-3] се прехвърля в регистъра за сегменти.

 movf  Inds,w     ; чрез номера на индикатора, от таблицата се
 call  TblInds    ; прочита състоянието на изходите. Може да стане и
 movwf PortAm     ; програмно но така програмата е по-кратка.

EndInt
 ...
```

Препоръки за реализация на програмата KIT_nxt.asm

В програмите **KIT_tmp, KIT_int и KIT_but** времето за смяна на индикацията не може да се задава плавно. По-надолу ще се разгледат възможностите как това да се направи.

1. Как да се променя периодът на обновяване на индикацията?

Може да стане като се променя честотата която се подава от прескалера (предварителен делител) към таймера. В реални устройства това обикновено **не се прави**, а прескалерът се ползва от **WDT** (куче пазач). Микроконтролерите **PIC16F716** имат 8-битов таймер. Има и прескалер който може да дели на 2, 4, 8… и т.н по степените на 2. Коефициентът на делене се задава **програмно** с трите най-младши бита на **OPTION** регистъра. Прескалерът се пренасочва към таймера или към **WDT** с бит **PSA** на **OPTION** регистъра. Препоръчва се програмите, особено за промишлеността, да ползват **WDT**. Активирането му става с бит в конфигурационната дума, която се записва при програмирането на микроконтролера и не може да се променя при изпълнение на програмата. При използване на **WDT** прескалерът се насочва към него, а таймерът работи с основната тактова честота която, за **KIT16F716,** е около 500kHz (тактовата честота е ¼ от осцилатора – 2MHz). Така прекъсването (препълване на таймера) ще става на около 0,5ms (256x1/500).

Най-лесно промяна на периода на обновяване на индикацията може да стане като обновяването се прави не при всяко прекъсване, а през няколко. Периодът се задава като се броят прекъсванията:

```
...
CountInt      equ   2d    ; дефинира се брояч на прекъсванията
...
```

В програмата за прекъсване **CountInt** се намалява с 1 и проверява дали е 0. Ако не е - се излиза от прекъсването без да се прави нищо.

```
...
bcf     IntCon,2
decfsz  CountInt    ; добавят се тези редове
goto    EndInt      ; броячът не е нула – излиза от прекъсване
movlw   5           ; на 5 прекъсвания ще се завърта индикацията
movwf   CountInt    ;
movlw   OffInd
movwf   PortA
movwf   Dig1
...
```

Като се променя числото което се зарежда в **CountInt** ще се променя и честотата на смяна на индикацията. В примера промяната може да се прави при писане на програмата – да се провери какъв е ефектът.

За да се провери при каква честота започва да се забелязва трепкането на индикацията програмно може да се задава на колко прекъсвания да се сменя разрядът. Така в регистър **SpeedInd** ще се задава честотата на смяна - вместо константа 5 се ползва **SpeedInd**:

```
...
movlw   5, -> movf SpeedInd,w ; така съдържанието на SpeedInd
              movwf CountInt   ; отива в CountInt
...
```

Стойността на **SpeedInd** задаваме в главната програма. Този регистър може да се променя при натискане на бутон, на няколко секунди или по друг начин. Целта е да се види работата на динамичната индикация.

2. Как да се отброява времето?

Една от задачите на този практикум е да се направи програма за часовник, а това значи, че трябва да се отчита време. Във всички случаи, при това схемно решение, в основата е тактовата честота на контролера.

В **KIT_tmp.asm** времето се отчита с "празни" цикли на контролера. В практиката това се ползва само за много малки временни интервали, напр. когато трябва да се осигури закъснение между такт и данни при работа с външни устройства по I2C. За задаване на основни интервали този подход е непригоден най-вече поради това, че вмъкването на всяка нова инструкция води до промяна на времето.

Използването на прекъсване е единствено решение, особено при контролери само с един таймер. Всъщност има и решение без прекъсване, но то се прилага само когато не е предвиден механизъм за прекъсване – най-простите контролери **PIC16C5x**. Тогава в програмата се следи флагът за препълване на таймера **TOIF** и когато той стане 1 се изпълнява програмата за индикация. При този вариант интервалите не са много точни.

В **KIT_int.asm** таймерът дели на 256 и генерира прекъсване при препълване. Когато не се ползва прескалера, това е прекалено често за управление на индикацията и "хаби" от програмното време. Затова, както бе показано по-горе, въртенето на индикацията става на няколко прекъсвания чрез **CountInt**.

Броенето на прекъсванията може да се използва и за отброяване на времето. Ако не се променя честотата за въртене на индикацията може да се ползва един и същ брояч, но за по-голяма свобода при програмиране се предпочита отделен брояч. Коефициентът на делене трябва да се избере така, че да се получи 1s – задачата е часовник. Освен това трябва да се заделят регистри за секунди, минути и часове.

За по-плавно задаване на времето, прекъсванията може да не са на 256 такта, а на други, по-малки стойности. Това става, като се зададе предварителна стойност на таймера и той да не брои от 0 до 255, а от xx до 255. За целта в програмата за прекъсване се добавят два реда:

```
...
bcf     IntCon,2    ; дотук е от досегашната програма
movlw   -TimOvl+3   ; интервалът на прекъсване TimOvl
addwf   Timer       ; се добавя към съдържанието на таймера
...                 ; Timer = TMR0
```

Броят тактове на които става прекъсването се задава в **TimOvl**. Разрешените стойности са от 0 до 255. Трябва да се внимава с малките стойности. **Най-малката стойност трябва да е по-голяма от времето (тактове) необходимо за изпълнение на прекъсването!**

На пръв поглед би трябвало **-TimOvl** да се зареди в **Timer**. Прекъсването, обаче, започва да се изпълнява не веднага когато се вдигне флаг **TOIF**, а няколко такта по-късно. Това зависи от инструкцията която се изпълнява в момента на прекъсване. Освен това, при някои процедури, за няколко такта се забраняват всички прекъсвания (с флаг GIE). Това би довело до неточности при отчитане на времето. Затова стойността на презареждане се добавя към текущата стойност на таймера и така автоматично се коригира закъснението при влизане в прекъсване. Знакът (-) е защото таймерът брои "нагоре". Добавката **+3** е за компенсиране на времето за изпълнение на **addwf Timer.**

3. Как да се ползват бутоните?

При работата с бутони има няколко основни правила.

- При всички механични бутони се наблюдава "разтрептяване" на контактите. То се изразява в неколкократна смяна на изходното състояние при преминаване от включено в изключено и обратно – обикновено преходният процес е 1-10ms.

- Времето за включване/изключване зависи от бутона но обикновено е 10-20ms. Ако състоянието на бутона се смени за по-кратко време то не бива да се отчита и се приема за грешка.

- За да се използват по-малко изводи на контролера, особено при по-голям брой бутони, се правят схеми за динамично четене.

- При натискане на бутон се изпълнява някакво действие – нулиране, старт/стоп, увеличаване/намаляване и т.н. Действие може да се изпълнява и при отпускане на бутона. Ако бутонът остане натиснат по-дълго от времето за изпълнение, това може да се възприеме като второ натискане. За да не стане това, ново натискане на бутона се изпълнява едва след като той е бил отпуснат. За да се реши това, програмно има различни начини – един е показан по-долу:

За всеки бутон са заделени два флага, единият показва текущото състояние на бутона, а другият предишното. Първият се променя синхронно със състоянието на бутона. Вторият също, но в 1 се установява при изпълнение на действието предвидено за този бутон. Това става само веднъж при едно натискане на бутона. Регистрите заделени за бутоните са **Buttons** (показва текущото състояние на бутоните) и **ButtonF** (при последната проверка). Използват се първите четири бита (**0-3**) на всеки регистър. За удобство - битовете са дефинирани **But0-But3** и **But0f-But3f**. В ***kit_int.asm*** се ползва само клетка **Buttons**, а съответните бутони се указват в клетката като номер на бит **Buttons,x** (**x=0-3**). Дефинирането на бутоните като **But0, But2...** създава удобство при писане.

В прекъсването, в регистър **Buttons**, се установява само текущото състояние на бутоните. Проверката, дали има промяна на бутоните и какво да се прави, се намира или в главната програма, или в специална подпрограма. Анализът включва:

```
...
movf    ButtonF,w     ; с xorwf се проверява дали в Buttons има
xorwf   Buttons,w     ; промяна (ако няма - Z е 0)
btfss   STATUS,Z
call    ButTest       ; изпълнява се когато има промяна
...
```

Ако текущото състояние е равно на предишното – няма промяна и нищо не се прави. В подпрограмата **ButTest** се проверява дали промяната е от натискане или отпускане на бутон. Проверките се правят побитово. Изпълняват се и действията според предназначението на бутоните. При излизане от подпрограмата регистър **ButtonF=Buttons**. Ето част от програмата:

```
...
ButTest                     ; ползват се само два бутона
   movf    Buttons,w
   movwf   ButtonF          ; запомня се състоянието на бутоните
   btfss   But0             ; ако But0 е натиснат
   goto    ClrBut0          ; се увеличава периодът
   incf    SpeedInd         ; на опресняване на индикацията
   return
ClrBut0
   btfss   But1             ; с But1 се
   return
   decf    SpeedInd         ; намалява периода
   return
...
```

Ако физически имаше само два бутона втората проверка за **But1** би била излишна – щом програмата е влязла в **ButTest** значи има натиснат бутон, а щом той не е **But0** значи е **But1**. В тази програма се ползват само двата бутона но може да се натиснат четири и затова трябва да се проверяват и двата. Ако се натиснат **But2** или **But3** просто нищо не се прави.

Тази програма не отчита натискането на два бутона едновременно, не избягва "разтрептяването" на контактите **но работи**. Натискането на два бутона едновременно не е допустимо при тази схема на динамична индикация – анодите на индикаторите към които са свързани двата бутона се свързват накъсо.

Разтрептяването на контактите също не е сериозен проблем – то продължава няколко ms и когато бутоните се четат по-рядко, няма грешни отчитания. Достатъчно е индикацията да се обновява с честота не по-висока от 100-200Hz. Ако честотата е по-висока за по-голяма сигурност се вземат програмни мерки - когато се отчете промяна на състоянието на бутоните, не се предприемат действия веднага, а се изчаква потвърждение на промяната. Ако още един-два пъти се прочете същото състояние на бутоните, то се приема за истинско и се изпълняват предвидените действия.

Коментари по листинга на програмата KIT_nxt.asm

Програмата доразвива **KIT_int.asm** с добавяне на решения на част от коментираните по-горе въпроси. Индицира се **SpeedInd** – на колко прекъсвания се обновява индикацията. С единия бутон стойността на **SpeedInd** се увеличава с другия – намалява. Подпрограмата за обработване на бутоните е:

```
ButTest                 ; работят само два бутона
   movf   Buttons,w
   movwf  ButtonF       ; запомня се състоянието на бутоните
   btfss  But0
   goto   ClrBut0
   incf   SpeedInd      ; увеличава периода на опресняване на индикацията
   return
ClrBut0
   btfss  But1
   return
   decf   SpeedInd      ; намалява периода
   return
```

Частта от главната програма за индициране на периода на опресняване:

```
   ...
   movf   SpeedInd,w    ; индицира се SpeedInd
   andlw  h'0f'         ; младши полубайт
   call   TblSegm
   movwf  Dig0

   swapf  SpeedInd,w    ; инструкцията разменя старши и младши полубайт
   andlw  h'0f'         ; старши полубайт
   call   TblSegm
   movwf  Dig1
   ...
```

В прекъсването времето се отброява с клетка **Clock**. В главната програма **Clock** се използва като най-младши брояч на часовник. Мястото в прекъсването където **Clock** се увеличава е важно – ако е в самото начало ще отброява всички прекъсвания, ако е при зареждане на индикацията ще работи с по-ниска честота и няма да се налага допълнителен делител. Това означава, че за да не се объркава времето, **SpeedInd** трябва да е константа.

По-долу е даден сорсът на програмата.

Задачите са подобни на предишните – да се експериментират описаните по-горе варианти.

```
;  Interrupt
;  Filename:      kit-nxt.asm
;  Date:          10.2015
;  File Ver.:     1.00
;
   list     p=16f716
   __config       h'fff7'     ; WDT и PWRTE са включени
;  CP=Off, PWRTE=On, WDT=On, OSC=RC
   include  p16f716.inc
;
;
Numb               equ   20

Dig0               equ   23
Dig1               equ   24
Dig2               equ   25
Dig3               equ   26
Inds               equ   27
Buttons            equ   28
ButtonF            equ   29
Segm               equ   2a
PortAm             equ   2b
CountInt           equ   2c
SpeedInd           equ   2d
Clock              equ   2e

Temp               equ   3c
W_Stack            equ   3d
S_Stack            equ   3e
FSR_Stack          equ   3f

#define     But   PORTA,4     ; вход за бутони
#define     But0  Buttons,0
#define     But1  Buttons,1
#define     But2  Buttons,2
#define     But3  Buttons,3

OffInd      equ   b'1111'
OffSeg      equ   h'ff'
TimOvl      equ   d'100'      ; време за препълване на таймера
;*****************
   org      h'00'             ; Начало
;*****************
Reset
   call     IniPic
   call     ClrRAM
   bsf      INTCON,GIE  ; след подготвителните операции се разрешават
                        ; прекъсванията зададени в INTCON.
   goto     Main        ; прескача се областта заделена за прекъсвания!

;***************
;   Interrupt ;
   org      h'04'       ; ! Всички прекъсвания на тази серия контролери се
; насочват към този адрес. Тук съответната програма разпознава от къде е
```

```
; прекъсването и пренасочва управлението. В случая са разрешени
; прекъсвания само от таймера и няма какво да се разпознава.

   movwf    W_Stack      ; запомнят се всички регистри които се ползват в
   movf     STATUS,w     ; подпрограмата за прекъсване
   movwf    S_Stack      ; W и Status регистрите се използват почти винаги
   movf     FSR,w
   movwf    FSR_Stack

   bcf      INTCON,2     ; задължително се нулира флагът който е предизвикал
                         ; прекъсването - иначе то ще се изпълнява безкрай!

   movlw    -TimOvl+3    ; интервалът на прекъсване се дава от TimOvl
   addwf    TMR0         ; и се добавя към съдържанието на таймера
   incf     Clock        ; увеличава се основният брояч

   decfsz   CountInt     ; добавят се тези редове
   goto     EndInt       ; броячът не е нула - излиза от прекъсването
;  movlw    5            ; на 5 прекъсвания ще се завърта индикацията
   movf     SpeedInd,w
   movwf    CountInt     ;

   incf     Inds,w       ; с тези три реда Inds
   andlw    h'03'        ; брои от 0 до 3
   movwf    Inds         ;

   movlw    OffInd       ; изключване на индикаторите
   movwf    PORTA        ; за да не се вижда смяната
   movf     Segm,w       ; с предварително подготвени
   movwf    PORTB        ; данни в Segm и PortAm се
   movf     PortAm,w     ; зареждат новите сегменти и
   movwf    PORTA        ; се включва новият индикатор

   btfsc    But          ; дали е натиснат бутона (RA4=1?)
   goto     CheckBt      ; натиснат е -> към CheckBt
   andwf    Buttons      ; не е натиснат -> нулира флага на бутона
   goto    Int1
CheckBt
   xorlw   h'ff'         ; когато бутонът е натиснат съответният
   iorwf   Buttons       ; флаг се установява в 1
Int1
   movlw    Dig0         ; W се зарежда с константа = адреса на Dig0
   addwf    Inds,w
   movwf    FSR          ; сега FSR "сочи" сегментите на индикатора който
   movf     INDF,w       ; ще се индицира при следващото прекъсване.
   movwf    Segm         ; Dig[0-3] се прехвърля в регистъра за сегменти.

   movf     Inds,w       ; чрез номера на индикатора, от таблицата се
   call     TblInds      ; прочита състоянието на изходите. Може да стане и
   movwf    PortAm       ; програмно но така програмата е по-кратка.

EndInt
   movf     FSR_Stack,w ; Възстановяват се регистрите в състоянието в
   movwf    FSR          ; което са били при възникване на прекъсването.
```

```
   movf     S_Stack,w
   movwf    STATUS
   swapf    W_Stack
   swapf    W_Stack,w
   retfie

;***************
;  Основна програма
;
Main  movlw d'10'       ; задават се начални стойности
   movwf    SpeedInd    ; на брояча на прекъсванията
   movwf    CountInt    ;
Main_
   movf     SpeedInd,w  ; индицира се SpeedInd
   andlw    h'0f'       ; младши полубайт
   call     TblSegm
   movwf    Dig0

   swapf    SpeedInd,w
   andlw    h'0f'       ; старши полубайт
   call     TblSegm
   movwf    Dig1

   movlw    b'00101111' ; r.
   movwf    Dig2

   movlw    b'10000111' ; t
   movwf    Dig3

   movf    ButtonF,w    ; с xorwf се проверява дали в Buttons има
   xorwf   Buttons,w    ; промяна (ако няма - Z е 0)
   btfss   STATUS,Z     ; проверка за нула -> Z=1
   call    ButTest      ; изпълнява се когато има промяна

   clrwdt               ; WDT е включен и трябва да се нулира

   goto     Main_
;**********************
;*       SubRout
;**********************
IniPic
   movlw    b'10000'
   tris     PORTA
   movlw    OffInd
   movwf    PORTA
   movlw    b'00000000'
   tris     PORTB
   movlw    OffSeg
   movwf    PORTB
   movlw    b'11011111' ; WDT е включен и трябва да се нулира
   option
   movlw    b'00100000' ; задава се прекъсване по таймера, но GIE ще
   movwf    INTCON      ; разреши всички прекъсванията по-нататък
   return
```

```
ClrRAM
   movlw    Numb
   movwf    FSR
   movlw    d'20'
   movwf    Numb
Clr1
   incf     FSR
   clrf     INDF
   decfsz   Numb
   goto     Clr1
   return

ButTest                 ; работят само два бутона
   movf     Buttons,w
   movwf    ButtonF     ; запомня се състоянието на бутоните
   btfss    But0
   goto     ClrBut0
   incf     SpeedInd    ; увеличава периода на опресняване
   return               ; на индикацията
ClrBut0     btfss But1
   return
   decf     SpeedInd    ; намалява периода
   return
;***
TblInds
   addwf    PCL
   retlw    b'11111110' ; индикатор 0
   retlw    b'11111101' ; индикатор 1
   retlw    b'11111011' ; индикатор 2
   retlw    b'11110111' ; индикатор 3
;***
TblSegm                 ; таблица в която всяка цифра е кодирана с
   addwf    PCL         ; комбинация от сегменти
   retlw    b'11000000' ; 0
   retlw    b'11111001' ; 1
   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'10001110' ; F

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

За да се продължи с написване на самостоятелни програми горната програма трябва да се разучи внимателно, да няма неясни участъци.

Програмата за прекъсване, в този си вид, може да се ползва от всяка следваща програма. В нея се управлява динамична индикация, като на индикаторите се изписват сегментите дадени в клетки **Dig0–Dig3**. В тази подпрограма се променя и клетка **Clock** която се увеличава с единица на всеки **SpeedInd** прекъсвания. Ако **Clock** ще се използва за отчитане на времето прекъсванията трябва да са на равни интервали от време. Затова **TimOvl** и **SpeedInd** трябва да са константи (в предишен вариант **SpeedInd** беше 5). Всъщност идеята **SpeedInd** да се променя е единствено с цел да се покаже как работи динамичната индикация.

В програмата **Main** задължително трябва да се остави инструкцията за нулиране на **WDT-clrwdt**. Може да се ползва и частта за проверка на бутоните, като се остави извикването на подпрограма **ButTest** но самата програма ще бъде с друго съдържание.

Може да се пробват различни програми в които се отчита време – таймер, хронометър, измервател на рефлекси. Във всички тях, за да се отчита правилно времето, **TimOvl** и **SpeedInd** трябва да се подберат така, че да се увеличава на интервали свързани със секундата – ms, μs и т.н. Ако това не се прави, в главната програма трябва да се правят сложни изчисления. За да не се получават чести прекъсвания **TimOvl** трябва да е между 200 и 255, а **SpeedInd** се определя така, че да няма трепкане на индикацията.

Започнете с програма хронометър. При натискане на бутон да започва броене, при повторно натискане – броенето спира. С друг бутон броячът се нулира.

Преди това може да се направи програма за броене до определено число – от 0 до 9, от 0 до 5 и др. За целта се работи подобно на програмата за опресняване на индикацията където при натискане на единия бутон показанието се увеличава, а на другия – намалява. След всяко натискане трябва да се проверява дали не е достигната максималната стойност (или минималната). В общия случай числото трябва да се започва от някаква минимална стойност и когато се достигне максималната със скок да се отиде на минималната. Например, както е при класическото броене – 0, 1, 2, … 8, 9, 0, 1, … В общия случай се започва от произволно число, напр. 3, 4, 5, 6, 7, 3, 4, … Същото да се направи и в обратна посока – числата да намаляват.

Във всички случаи след всяко натискане на бутон трябва да се проверява дали не е достигната максималната (или минималната стойност) и, ако е достигната, вместо да се увеличи числото да се запише минималната стойност.

Проверката може да се направи по няколко начина – с **xorlw, xorwf** са функции “изключващо ИЛИ” и ако резултатът е нула значи двете величини участвали в операцията са еднакви. По подобен начин се ползва и функцията за изваждане – ако резултатът е нула имаме равенство.

При изваждането имаме предимството, че може да проверим за по-голямо или равно. Когато ползваме **xor...** и числото в брояча което увеличаваме е по-голямо от максималното, то равенство няма да се получи докато не се препълни броячът – при еднобайтов брояч трябва да се достигне до 255, да се мине през 0 и тогава ще започне нормална работа. При проверка с изваждане (равно или по-голямо) веднага ще влезем в нормален режим.

По подобен начин се проверява със събиране – **addlw, addwf**. Събира се с отрицателно число. За да проверим дали броячът е достигнал 10 към него добавяме -10. Ако се получи нула значи е станал 10 и трябва да го нулираме. Проверките може да бъдат за флаг Z (нула) или за флаг C (пренос).

В следващата програма е реализиран таймер (брояч) с време на нарастване около 1s. Периодичността на прекъсването е зададено от константата **TimOvl**, а опресняването на индикацията става на всеки **SpeedInd** прекъсвания. Конкретните стойности са 200 и 10. Така на 2000 такта се завърта индикацията и клетката **Clock** се увеличава с 1. Това, при генератор на 4 MHz, означава, че увеличаването на **Clock** става на 2ms. Ако честотата е друга, грубата настройка става със **SpeedInd**, а фината - с **TimOvl**.

Добавени са две клетки за отчитане на времето – за 100ms **Timer_100**, а за 1 секунда - **Timer_1s**.

```
;  Interrupt
;  Filename: kit-nxt.asm
;  Date:           10.2015
;  File Ver.:      1.00
;
   list      p=16f716
   __config  h'fff7'       ; WDT и PWRTE са включени
;  CP=Off, PWRTE=On, WDT=On, OSC=RC
   include   p16f716.inc
;
;
Numb          equ    20

Dig0          equ    22
Dig1          equ    23
Dig2          equ    24
Dig3          equ    25
Inds          equ    26
Buttons       equ    27
ButtonF       equ    28
Segm          equ    29
PortAm        equ    2a
CountInt      equ    2b
Clock         equ    2c
Timer_100     equ    2d
Timer_1s      equ    2e

Temp          equ    3c
W_Stack       equ    3d
S_Stack       equ    3e
FSR_Stack     equ    3f

#define       But    PORTA,4       ; вход за бутони
#define       But0   Buttons,0
#define       But1   Buttons,1
#define       But2   Buttons,2
#define       But3   Buttons,3

OffInd        equ    b'1111'
OffSeg        equ    h'ff'
TimOvl        equ    d'200' ; време за препълване на таймера
SpeedInd      equ    d'10'  ; индикацията се "завърта" SpeedInd x TimOvl = 2000 такта
;*****************
   orgh'00'               ; Начало
;*****************
Reset
   call      IniPic
   call      ClrRAM
   bsf       INTCON,GIE    ; след подготвителните операции се разрешават
                           ; прекъсванията зададени в INTCON.
   goto      Main          ; прескача се областта заделена за прекъсвания!

;***************
;  Interrupt ;
   orgh'04'         ; ! Всички прекъсвания на тази серия контролери се
; насочват към този адрес. Тук съответната програма разпознава от къде е прекъсването
; и пренасочва управлението.
; В случая са разрешени прекъсвания само от таймера и няма какво да се разпознава.
```

```
   movwf    W_Stack       ; запомнят се всички регистри които се ползват в
   movf     STATUS,w      ; подпрограмата за прекъсване
   movwf    S_Stack       ; W и Status регистрите се използват почти винаги
   movf     FSR,w
   movwf    FSR_Stack

   bcf      INTCON,2      ; задължително се нулира флагът който е предизвикал
                          ; прекъсването - иначе то ще се изпълнява безкрай!

   movlw    -TimOvl+3     ; интервалът на прекъсване се дава от TimOvl
   addwf    TMR0          ; и се добавя към съдържанието на таймера

   decfsz   CountInt      ; добавят се тези редове
   goto     EndInt        ; броячът не е нула – излиза от прекъсване
   movlw    SpeedInd      ; на TimOvl x SpeedInd ще се
   movwf    CountInt      ; завърта индикацията и се
   incf     Clock         ; увеличава основният брояч Clock

   incf     Inds,w        ; с тези три реда Inds
   andlw    h'03'         ; брои от 0 до 3
   movwf    Inds          ;

   movlw    OffInd        ; изключване на индикаторите
   movwf    PORTA         ; за да не се вижда смяната
   movf     Segm,w        ; с предварително подготвени
   movwf    PORTB         ; данни в Segm и PortAm се
   movf     PortAm,w      ; зареждат новите сегменти и
   movwf    PORTA         ; се включва новият индикатор

   btfsc    But           ; дали е натиснат бутона (RA4=1?)
   goto     CheckBt       ; натиснат е -> към CheckBt
   andwf    Buttons       ; не е натиснат -> нулира флага на бутона
   goto     Int1
CheckBt
   Xorlw    h'ff'         ; когато бутонът е натиснат съответният
   Iorwf    Buttons       ; флаг се установява в 1
Int1
   movlw    Dig0          ; W се зарежда с константа = адреса на Dig0
   addwf    Inds,w
   movwf    FSR           ; сега FSR "сочи" сегментите на индикатора който
   movf     INDF,w        ; ще се индицира при следващото прекъсване.
   movwf    Segm          ; Dig[0-3] се прехвърля в регистъра за сегменти.

   movf     Inds,w        ; чрез номера на индикатора, от таблицата се
   call     TblInds             ; прочита състоянието на изходите. Може да стане и
   movwf    PortAm        ; програмно но така програмата е по-кратка.

EndInt
   movf     FSR_Stack,w   ; Възстановяват се регистрите в състоянието в
   movwf    FSR           ; което са били при възникване на прекъсването.
   movf     S_Stack,w
   movwf    STATUS
   swapf    W_Stack
   swapf    W_Stack,w
   retfie
```

```
;***************
;  Основна програма
;
Main
    movlw     SpeedInd      ;
    movwf     CountInt      ;
Main_
    movf      Timer_1s,w    ; индицира се Timer_1s
    andlw     h'0f'         ; младши полубайт
    call      TblSegm
    movwf     Dig0

    swapf     Timer_1s,w
    andlw     h'0f'         ; старши полубайт
    call      TblSegm
    movwf     Dig1

    movlw     b'10011100'   ; o
    movwf     Dig2

    movlw     b'10011100'   ; o
    movwf     Dig3

    call      Make_Time     ; подпрограма за обработка на времето

    movf      ButtonF,w     ; с xorwf се проверява дали в Buttons има
    xorwf     Buttons,w     ; промяна (ако няма - Z е 0)
    btfss     STATUS,Z      ; проверка за нула -> Z=1
    call      ButTest       ; изпълнява се когато има промяна



    clrwdt                  ; WDT е включен и трябва да се нулира

    goto      Main_
;***********************
;*       SubRout
;***********************
IniPic
    movlw     b'10000'
    tris      PORTA
    movlw     OffInd
    movwf     PORTA
    movlw     b'00000000'
    tris      PORTB
    movlw     OffSeg
    movwf     PORTB
    movlw     b'11011111'   ; WDT е включен и трябва да се нулира
    option
    movlw     b'00100000'   ; задава се прекъсване по таймера, но GIE ще
    movwf     INTCON        ; разреши всички прекъсванията по-нататък
    return

ClrRAM
    movlw     Numb
    movwf     FSR
    movlw     d'20'
    movwf     Numb
Clr1
```

```
    incf      FSR
    clrf      INDF
    decfsz    Numb
    goto      Clr1
    return

ButTest                         ; работят само два бутона
    movf      Buttons,w
    movwf     ButtonF           ; запомня се състоянието на бутоните
    btfss     But0
    goto      ClrBut0
; подпрограма за изпълнение на бутон 0
    return
ClrBut0       btfss  But1
    return
; подпрограма за изпълнение на бутон 1
    return

Make_Time
    movlw     -d'50'        ; за да се увеличи с единица Timer_100
    addwf     Clock,w       ; трябва да са минали 100ms. Clock се увеличава
    btfss     STATUS,C      ; на всеки 2ms (200x10)
    return
    movlw     -d'50'
    addwf     Clock
    decfsz    Timer_100
    return
    movlw     d'10'         ; на всеки 10 единици в Timer_100
    movwf     Timer_100     ; минава една секунда -> увеличаваме
    incf      Timer_1s      ; Timer_1s
    return
;***
TblInds
    addwf     PCL
    retlw     b'11111110'   ; индикатор 0
    retlw     b'11111101'   ; индикатор 1
    retlw     b'11111011'   ; индикатор 2
    retlw     b'11110111'   ; индикатор 3
;***
TblSegm                           ; таблица в която всяка цифра е кодирана с
    addwf     PCL           ; комбинация от сегменти
    retlw     b'11000000'   ; 0
    retlw     b'11111001'   ; 1
    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'10001110'   ; F

    End
```