Hardver alapok 4. laborgyakorlat

A laborgyakorlat célja:
Egy olyan mikrokontrolleres fejlesztői környezet megismerése, amely tartalmazza a szövegszerkesztőt, a fordítót, a szimulátort és valamilyen letöltő/hardver debug egységeket.

A szimulátor használatának elsajátítása a következő tevékenységek során:

- Szoftveres késleltetés készítése, használata
- Időzítés pontos meghatározása (számítással, majd ellenőrzése szimulációval)
- Nyomógomb megnyomásának szimulációja adott portbiteken
- Kimeneti portbitek működésének grafikus megjelenítése („logikai analizátor”)
- Egyszerű program lépésenkénti végrehajtása, flag-ek és regiszterek (watch) megfigyelése
- Töréspont elhelyezése
- Hiba keresése/javítása adott rutinban

Bevezetés

A laborfoglalkozások során rövid programokat fogunk készíteni, melyek egyetlen .asm fájlból állnak. Ennek a fájlnak a programunkon kívül bizonyos kiegészítő részeket is tartalmaznia kell a helyes működéshez.
A kód legelején egy külső definíciós fájlt kell meghivatkozni a `#include` direktívával a programozni kívánt mikrokontroller típusának megfelelően. Esetünkben a mikrokontroller típusa PIC16F18875, ezért a következőt írjuk a forráskódba:

```
    #include "p16f18875.inc"
```

A következő blokk a konfigurációs bitek megadását tartalmazza. Ezek automatikusan generálhatóak („Window”→”PIC Memory Views”→”Configuration Bits” menü), részletes beállításukkal nem foglalkozunk, csak megadunk egy a laborfoglalkozásokhoz használandó konfigurációt. Ez a konfiguráció 32MHz belső oszcillátort állít be, kikapcsolja a kódvédelmet, kikapcsolja a watchdog áramkört és a brown-out reset áramkört.

```
    __CONFIG _CONFIG1, 0x3F8C
    __CONFIG _CONFIG2, 0x3F3F
    __CONFIG _CONFIG3, 0x3F9F
    __CONFIG _CONFIG4, 0x3FFF
    __CONFIG _CONFIG5, 0x3FFF
```

Ezt követően definiálhatjuk a konstansokat, változókat. A BANK0 memóriaterületre az `UDATA` direktíva segítségével, a kezdőcím megadásával helyezhetünk változókat. A változók helyfoglalására használjuk a `RES` direktívát.

```
    #define GOMB_MASK B'00010000'   ; RB4 maszk

    Bank0vars   UDATA     0x020 ;A következő változók elhelyezése H’020’ címtől
    GOMB_TEMP   RES 1;
    GOMB_MOST   RES 1;
    FALL_CNT    RES 1;
```

A programunk a külső reset hatására a 0000h címről indul. Ahhoz, hogy a fordító ide helyezze az általunk írt kódot, a következő direktívák használata javasolt:

```
RES_VECT  CODE    0x0000            ; processor reset vector
    GOTO    START                   ; go to beginning of program

MAIN_PROG CODE                      ; Kódszegmens eleje
START:
   ; ide jöhet az inicializálás
   CALL  INIT_BOARD  ; perifériák alaphelyzetbe állítása
   ; változók kezdő érték adása…

MAIN_LOOP: ; főciklus
   ; ide jön a feladat
   GOTO  MAIN_LOOP

END; jelzés a fordítónak a forráskód végéről
```

A bevezető végén még pár gondolat a perifériák inicializálásáról. A mikrokontroller a reset hatására az összes portbitjét analóg üzemmódra, bemenetre programozza. A RAM terület értéke véletlenszerű, a perifériák kikapcsolt állapotban, a megszakítások letiltott állapotban vannak. A mi feladatunk, hogy a programunk legelején minden használni kívánt egységet megfelelő üzemmódra állítsunk. Javasoljuk, hogy ezt az inicializáló részt tegyük egy külön szubrutinba (**INIT_BOARD**), és a reset után nem sokkal hívjuk meg a főciklusba lépés előtt. Ezt a kódrészletet minden feladatnál át kell írni a felhasználni kívánt perifériák szerint.

```
INIT_BOARD:
    BANKSEL ANSELA
    CLRF    ANSELA    ; PORTA digitális
    BSF     ANSELA,0  ; A0 analóg (potenciométer)
    CLRF    ANSELB    ; PORTB digitális
    CLRF    ANSELC    ; PORTC digitális
    CLRF    ANSELD    ; PORTD digitális
    BANKSEL TRISA     ; innentől minden 0-s bankban van
    MOVLW   0x0F      ; A4-5-6-7 ledek kimenetere
    MOVWF   TRISA
    RETURN
```

Ezt követően nem szabad még megfeledkeznünk a programunk által használt változók megfelelő kezdőértékének beállításáról sem.

Mivel a programot igazából nem lehet leállítani, mert a mikrokontroller mindig szeretne utasításokat végrehajtani (még ha sleep üzemmódban néha le is áll egy kicsit), ezért a programban célszerű egy nagy végtelen főciklust létrehozni (**MAIN_LOOP**). Ebben a főciklusban várakozhatunk pl.: egy nyomógomb lenyomására, melynek hatására elvégzünk bizonyos teendőket, majd ismét visszatérünk.

Számábrázolás, konstansok megadása

Az MPASM fordító támogatja a bináris, decimális és hexadecimális számábrázolást egyaránt. A fordítónak külön paraméterként megadható az alapértelmezett számrendszer, de ha biztosra szeretnénk menni, akkor javasolt a pontos formátum megadás. Vegyük például a decimális 25-öt. Ezt a különböző számrendszerekben a következőféleképpen adhatjuk meg:

Számrendszer	**Lehetséges megadások**		
Bináris	B'00011001'	b'00011001'	0b11001
Decimális	D'25'	d'25'	.25
Hexadecimális	H'19'	h'19'	0x19

Ha egyszerűen csak 25-öt írunk, akkor az alapértelmezett számábrázolástól függően kerül értelmezésre. A frissen telepített fejlesztő rendszer esetében hexadecimális az alapértelmezett, ezért a leírt 25 számot decimális 37-nek fogja értelmezni.

0. Szimulátor kipróbálása

Egy hallgatónak a következő specifikáció szerinti házi feladatot kellett megvalósítania ASM nyelven.

***„Írjon programot**, amely a PORTB 4. bitjére kapcsolt alacsony aktív nyomógomb lenyomásának hatására (lefutó él) két darab 10us hosszúságú impulzust állít elő 20us távolsággal a mikrokontroller RA7 kimenetén.”*

A feladat megoldására a következő program készült el:

```
;*******************************************************************************
; Demo feladat szimulátor kipróbálásához
;*******************************************************************************
;    Revision History:       V1.0
;*******************************************************************************
#include "p16f18875.inc" ; mikrokontroller specifikus definíciók
   __CONFIG _CONFIG1, 0x3F8C
   __CONFIG _CONFIG2, 0x3F3F
   __CONFIG _CONFIG3, 0x3F9F
   __CONFIG _CONFIG4, 0x3FFF
   __CONFIG _CONFIG5, 0x3FFF
;*******************************************************************************
; Reset Vector
;*******************************************************************************
RES_VECT  CODE    0x0000            ; processor reset vector
    GOTO    START                   ; go to beginning of program
;*******************************************************************************
; MAIN PROGRAM
;*******************************************************************************
#define     Gomb_S1    PORTB,4
#define     Gomb_S1_IRANY     TRISB,4
#define     Gomb_S2    PORTC,5
#define     Gomb_S2_IRANY     TRISC,5
#define     LED_D2     LATA,4
#define     LED_D2_IRANY      TRISA,4
#define     LED_D3     LATA,5
#define     LED_D3_IRANY      TRISA,5
#define     LED_D4     LATA,6
#define     LED_D4_IRANY      TRISA,6
#define     LED_D5     LATA,7
#define     LED_D5_IRANY      TRISA,7

GPR_VAR     UDATA 0x20
BITCNT      RES        1; ciklusváltozó

MAIN_PROG CODE          ; Kódszegmens eleje

START:
     BANKSEL ANSELA
     CLRF    ANSELA     ; port digitális
     CLRF    ANSELB     ; port digitális
     CLRF    ANSELC     ; port digitális
     BANKSEL LATA
     NOV     ; ez egy szintaktikai hiba
     BCF     LED_D2      ; kezdoérték
     BCF     LED_D2_IRANY      ; kimenet
     BCF     LED_D3      ; kezdoérték
     BCF     LED_D3_IRANY      ; kimenet
     BCF     LED_D4      ; kezdoérték
     BCF     LED_D4_IRANY      ; kimenet
     BCF     LED_D5      ; kezdoérték
     BCF     LED_D5_IRANY      ; kimenet

     BSF     Gomb_S1_IRANY     ; bemenet
     CLRF    STATUS
```

```
MAIN_LOOP:
     BTFSS   Gomb_S1    ; alaphelyzetre vár
     GOTO    MAIN_LOOP

Var0:
     BTFSC   Gomb_S1    ; lefutó él
     GOTO    Var0
     MOVLW   D'02'
     MOVWF   BITCNT

PULS_LOOP:
     DECF    BITCNT,F   ; ciklus változó csökkentése
     BTFSC   STATUS,C
     GOTO    MAIN_LOOP
     BSF LED_D5         ; LED bekapcsolása
     CALL    Delay10us
     BCF LED_D5         ; LED kikapcsolása
     CALL    Delay10us
     GOTO    PULS_LOOP
     GOTO MAIN_LOOP

Delay10us: ; 125ns/ciklus@Fosc=32MHz-->8MHZ instructionclock
     MOVLW   D'10'      ; 3
D10CK:
     DECFSZ  WREG,W     ; 1 /2
     GOTO    D10CK      ; 2
     RETURN             ; 2

    END
```

Feladatunk értékelni a hallgató munkáját, a specifikációtól való eltéréseket megkeresni és kijavítani az esetleges programhibákat.

Az elkészített projektet a tárgy honlapjáról tudjuk letölteni (HFCHECK).

Feladatok:

1. Indítsuk el az MPLAB-X fejlesztői környezetet, nyissuk meg a HFCHECK projektet.
2. Szimuláljuk a gombnyomást, használjuk az „Asynchron stimulus” funkciót. ellenőrizzük a működést (melyik élre indul, tényleg két pulzus jön-e?)
3. Jelenítsük meg a program változóit (Watch). Lépésenkénti végrehajtással figyeljük meg a program működését. A szubrutin hívó utasításon érdemes kipróbálni a „Step into”, „Step out”, illetve a „Run to cursor” funkciókat.
4. Próbáljunk meg töréspontot elhelyezni („Toggle Line Breakpoint”) a második impulzus végére.
5. Jelenítsük meg a kimenetet (Logic Analyzer).
6. Ellenőrizzük az időzítések hosszát, használjuk a „Stopwatch” funkciót.
7. Javítsuk meg a talált eltéréseket, dokumentáljuk az elvégzett munkát.

1. Szoftveres késleltetés készítése

Feladat:
Felhasználva az előző példaprogramot, írjunk három assembly szubrutint (**`Delay10us`**, **`Delay1ms`** és **`Delay250ms`**) amely a meghívásához szükséges CALL utasítás végrehajtási idejét is figyelembe véve 10µs, 1ms és 250ms késleltetést okoz.

1. A mikrokontroller órajelének ismeretében határozzuk meg a szükséges utasítás ciklusszámot.
2. Készítsünk olyan ciklust, amely egy változó értékét csökkenti annak nullázódásáig.
3. Határozzuk meg a ciklusváltozó megfelelő kezdőértékét 10us késleltetéshez.
4. Készítsük el a szubrutint.
5. A szimulátor stopwatch funkciója segítségével ellenőrizzük az időzítés alakulását, végezzük el a szükséges „finom hangolást”.
6. A **`Delay10us`** rutint felhasználva készítsünk 1ms késleltető szubrutint (**`Delay1ms`**).
7. A szimulátor stopwatch funkciója segítségével ellenőrizzük az időzítés alakulását, végezzük el a szükséges „finom hangolást”.
8. A **`Delay1ms`** rutint felhasználva készítsünk 250ms késleltető szubrutint (**`Delay250ms`**).
9. A szimulátor stopwatch funkciója segítségével ellenőrizzük az időzítés alakulását, végezzük el a szükséges „finom hangolást”.

2. Adott idejű Impulzus előállítása

Feladat:
Írjunk assembly programot, amely az előző feladatban elkészített **`Delay1ms`** szubrutint felhasználva egy 1ms hosszú impulzust állít elő az RA4 kimeneten. Az impulzust az RB4 bemeneten található nyomógomb lefutó élének észlelésekor állítsa elő. A lefutó él érzékeléséhez használjuk az első feladatban látott megoldást. A perifériák inicializálását helyezzük egy külön szubrutinba (**`BOARDINIT`**)

1. Állítsuk elő a pulzust lefutó él érzékelésének hatására
2. Szimuláljuk a működést, használjuk az „Asynchron stimulus” funkciót a nyomógomb modellezésére.
3. Jelenítsük meg a gomb állapotát és a kimenetet a logikai analizátor ablakban.

3. Reklám világítás vezérlése

Feladat:
A mérőpanelen található D2, D3, D4 és D5 világító diódák (LED) felhasználásával írjunk olyan programot, amely egy futófényt valósít meg. Egyszerre mindig pontosan egy LED világítson. (Hasonlóan a 3. laborfoglalkozás feladatához, de most szoftveres megvalósítással)
A programot először szimulátorban teszteljük, az időzítéshez használjuk a korábban elkészített **`Delay10us`** szubrutint.
A panelen történő futtatáskor az időzítéshez a **`Delay250ms`** szubrutint használjuk.

Portbit	LED
RA7	D5
RA6	D4
RA5	D3
RA4	D2
RB4	S1 nyomógomb
RC5	S2 nyomógomb

Módosítsuk a programot úgy, hogy az S1 nyomógomb lenyomásával lehessen a futás irányát változtatni.

Segítség a megoldáshoz:
Először állítsuk kimenetre a LED-eket vezérlő portbiteket. Használjuk ki, hogy az RA4..7 bitek sorban vannak. Definiáljunk egy DATAREG változót, ebben tároljuk a kiviendő mintát. A regiszter értékét bit léptető utasítással módosítsuk, ha elértük a szélét, újra betöltjük a kezdőértéket a DATAREG változóba.
Az irányváltáshoz elkészítjük a balra futó változatot is, majd a főciklusban a nyomógomb értékétől függően ugrunk a megfelelő léptető rutinba.

Szorgalmi feladatok:

Memória tartalmának másolása

Feladat:
Készítsünk szubrutint (**MEMCOPY**), amely az **FSR0** kezdőcímtől kezdődő memória területet átmásolja az **FSR1** regiszterben kijelölt kezdőcímtől. Az átmásolandó bájtok számát a **W** regiszter tartalmazza.

Ellenőrző összeg számolása

Feladat:
Készítsünk szubrutint (**CALCSUM**), amely az **FSR0** kezdőcímtől kezdődő memória területen található **W** darabszámú bájt mod256 ellenőrző összegét számolja ki és helyezi el a terület végén. A mod256 ellenőrző összeg a bájtok összege 256-tal való egész osztásának maradéka.

4. Ellenőrző kérdések

Az alábbi ellenőrző kérdések megválaszolásához nézze át az előadáson elhangzott anyagokat és tanulmányozza a mérési útmutatót.

- Mire használható a szimulátor "stopwatch" funkciója?
- Mire használható a szimulátor "watch" ablaka?
- Mire használható a szimulátor "Asynchron stimulus" funkciója?
- Program nyomkövetése esetén mikor használjuk a "Step into", "Step over", és a "Run to cursor" funkciót?
- Írja fel assembly szintaxissal a decimális 20 értéket binárisan.
- Írja fel assembly szintaxissal a decimális 20 értéket decimálisan.
- Írja fel assembly szintaxissal a decimális 20 értéket hexadecimálisan.
- Adott az alábbi programrészlet. Rajzolja fel a program lefutása során keletkező kimenet jelalakját, ha a portláb kimenetre van konfigurálva és kezdeti értéke 0.

```
BSF  LATA,5
NOP
BCF  LATA,5
```

- Adott az alábbi szubrutin. Rajzolja fel a szubrutin lefutása során keletkező kimenet jelalakját, ha a portláb kimenetre van konfigurálva.

```
TP:
   MOVLW  H'03'
TPCK:
   BSF    LATA,5
   NOP
   BCF    LATA,5
   DECFSZ WREG,F
   BRA    TPCK
   RETURN
```

- Hányszor hajtódik végre a **NOP** utasítás a fenti szubrutin lefutása során?.