TD2_bis Microcontrôleur

Exercice 1

1. Donner la signification de chaque instruction et son mode d'adressage dans le deux cas a) et b).
   a) Movlw 0x55 ; mettre la valeur 0x55 dans WREG « (WREG)←0x55 ». Adressage immédiat
      Movlb 0x02 ; mettre la valeur 0x02 dans BSR « (BSR)←0x55 ». Adressage immédiat
      Movwf 0x40,1 ; transférer le contenu de W (0x55) dans la case mémoire d'adresse 0x240 « (0x240)← (WREG) ». Adressage direct
      Movwf 0x41,1 ; (0x241) ← (WREG). Adressage direct
      Movwf 0x42,1 ; (0x242) ← (WREG). Adressage direct
      Movwf 0x43,1 ; (0x243) ← (WREG). Adressage direct

   b) Movlw 0x55 ; mettre la valeur 0x55 dans WREG « (WREG)←0x55 ». Adressage immédiat
      Lfsr FSR0,0x240 ; FSR0← 0x240 : charge dans FSR l'adresse 0x240. Adressage immédiat
      Movwf INDF0 ; charger dans la case mémoire pointer par FSR0 le contenu de W. Adressage indirect
      Incf FSR0L,f,1 ; incrémenter le contenu de FSR0, FSR0 contient l'adresse suivante : 0x241. Adressage direct
      Movwf INDF0 ; (FSR0)=(0x241) ← 0x55 . Adressage indirect
      Incf FSR0L,f,1 ; incrémenter FSR0 : FSR0←0x42. Adressage direct
      Movwf INDF0 ; (FSR0)=(0x242) ← 0x55 . Adressage indirect
      Incf FSR0L,f,1 ; incrémenter FSR0 : FSR0←0x43. Adressage direct
      Movwf INDF0 ; (FSR0)=(0x243) ← 0x55 . Adressage indirect
2. Que réalisent les séquences a) et b) ?
   Les séquences a) et b) font la même chose. Charger les cases mémoires d'adresses 0x240, 0x241, 0x243 et 0x243 par la valeur 0x55.

Exercice 2

1. Quel est le rôle du registre BSR ? permet de sélectionner l'un des banques de 0 à 15
2. Ecrire le fragment de code assembleur permettant de charger la valeur 0x99 dans la case mémoire d'adresse 0x202 ;
   L'adresse de la mémoire est codée sur 12 bits. C'est la concaténation de BSR et les 8 bits qui proviennent de l'instruction.
   Adresse de la mémoire = 0x202 = 0b001000000010
   Movlb 0x02 ; sélectionner la banque 2
   Movlw 0x99 ; charger dans W la valeur 0x99
   Movwf 0x02,1 ; transférer le contenu de W à l'@ 0x02 située dans la banque 2.
3. Ecrire le fragment de code assembleur permettant de copier la contenu de la case mémoire d'adresse 0x202 dans la case mémoire d'adresse 0x203 ;
   Les deux cases mémoire sont situées dans la banque 2
   Movlb 0x02 ; sélectionner la banque 2

   Movf 0x02, w,1 ; charger la contenu d'adresse 0x202 dans W
   Movwf 0x03,1 ; charger la contenu de WREG dans la case mémoire d'adresse 0x203
   On peut utiliser l'instruction moff : movff 0x203, 0x202
4. Ecrire la séquence assembleur qui permet de transférer le contenu du PORTB dans WREG. Movf PORTB,W
5. Quel sont les registres utilisés comme pointeurs de données en adressage indirect ?
   FSR0, FSR1 et FSR2
6. Donner les capacités des mémoires RAM et Flash du microcontrôleur PIC18F4520.
   RAM = 1536 octets ; Flash = 32ko
7. Quel est l'avantage de mettre des données dans la mémoire Flash ?
   Décharger la RAM et optimiser le code
8. Quels sont les registres à utiliser pour accéder à une donnée chargée dans la mémoire Flash ? TBLPTR comme pointeur et TABLAT comme registre de donnée
9. Donner les étapes nécessaires pour lire une information située dans la mémoire Flash à l'adresse 0x2000.
   - Charger dans TBLPTRL la partie basse de l'adresse = 0x00
   - Charger dans TBLPTRH la valeur 0x20
   - Charger dans TBLPTRU la valaur 0
   - Invoquer l'instruction TBLRD, le contenu de la case mémoire d'adresse 0x2000 sera chargé immédiatement dans le registre TABLAT
10. Dans quel mémoire (RAM ou Flash) doit-on sauvegarder les données suivantes :
    a) Température ; RAM
    b) Date et heure ; RAM
    c) Version du programme ; Flash
11. Ecrire le code assembleur qui calcule l'expression : k = i –j . les variables i, j et k ont pour adresses 0x400, 0x401 et 0x402.
    Les variables i, j , k sont situées dans la banque 4
    Movlb 0x04 ; sélectionner la banque 2
    Movf 0x01,w,1 ; charger la contenu d'adresse 0x401 dans W
    Subwf 0x00,W,1 ; soustraire W de la case mémoire d'adresse 0x400. Le résultat dans W
    Movwf 0x02,1 ; charger le contenu de W (résultat) à l'adresse 0x402
12. Donner la valeur de la case mémoire affectée après exécution de chaque instruction. En prend pour états initiales :
    WREG = 0x3D ; BSR = 0x00 ;

Valeurs initiales	
Adresse	**Contenu**
0x5A	0x3B
0x5B	0xA2
0x5C	0xF4

Instructions

subwf 0x5C,f
movlw 0x5C
addwf 0x5C,f
addwf 0x5A,f
incf 0x5B,w

Subwf 0x5C,f ; (0x5C)← (0x5C) –W = 0xF4 – 0x3D = 0xB7

```
Movlw 0x5C          ; W ← 0x5C
Addwf 0x5C,f        ; (0x5C) ← (0x5C) +W = 0xB7 + 0x5C = 0x13
Addwf 0x5A,f        ; (0x5A) ← (0x5A) +W = 0x3B + 0x5C = 0x97
Incf 0x5B, w  ; w ← (0x5B) +1 = 0xA2 + 1 = 0xA3
```

Après exécution

Adresse	Contenu	
0x5A	0x97	WREG = 0xA3
0x5B	0xA2	
0x5C	0x13	

13. Traduire en assembleur le fragment du code suivant :

```
Char i,j ;
If(i>j)
   i = j ;
        movf i,w      ; charger dans W le contenu de i
        subwf j,w     ; soustraire w de j, résultat dans W
        btfsc STATUS, C ; sauter si C = 0 (si  j – i <0 c-à-d i > j )
        goto suite
        movf j,w      ; charger dans W le contenu de j
        movwf  i      ; charger le contanu de W dans i
 suite :
```

Autre méthode

```
        movf j,w      ; charger dans W le contenu de j
        cpfsgt  i     ; skip si i >j
        goto suite
        movf j,w      ; charger dans W le contenu de j
        movwf  i      ; charger le contanu de W dans i
 suite :
```

Exercice 3

On considère des nombres signés codés sur 8 bits, le 8ème bit est un bit de signe.

Donner le résultat ainsi que l'état des indicateurs N, Z, et C dans les cas suivants :

0x01 + 0xFF ; 0x80 + 0x7F ; 0xF2 – 0x20

0x01 + 0xFF = 0x100 C = 1 et Z = 1

0x80 + 0x7F = 0xFF C = 0 et Z = 0

0xF2 – 0x20 = 0xD2 C = 1 et Z = 0

Exercice 4

Soit le programme suivant :

```
signed char MyNum[]={+1, -1, +2, -2, +3, -3, +4, -4} ;
char z ;
void main()
{       TRISD =0 ;
        for(z =0 ; z < 8 ; z++)
                PORTD = MyNum[z] ;
        while(1) ;
}
```

Donner dans un tableau les valeurs (en binaire) à la sortie du PORTD en fonction de z.

Z = 0 PORTD = 0b00000001

Z = 1 PORTD = 0b11111111

Z = 2 PORTD = 0b00000010

Z = 3 PORTD = 0b11111110

Z = 4 PORTD = 0b00000011

Z = 5 PORTD = 0b11111101

Z = 6 PORTD = 0b00000100

Z = 7 PORTD = 0b11111100

Exercice 5

Donner les valeurs des ports B, C et D après exécution.

```
void main()
{       TRISB =0 ; TRISC = 0 ; TRISD = 0 ;
        PORTB = 0x35 & 0x0F ;   PORTC = 0x04 | 0x68 ;   PORTD = 0x54 ^0x78 ;
        PORTB = ~0x55 ;         PORTC |= (0x9A>> 3) ;   PORTD &= 0x06 ;
        While(1) ;
}
```

PORTB = 0x05 ; PORTC = 0x6C ; PORTD = 0x2C

PORTB = 0xAA ; PORTC = 0x7F ; PORTD = 0x04

Exercice 6

```
char MyDATA[] = {0x25, 0x62, 0x3F, 0x52} ;
char sum =0 ;
char z, Checksum ;
void main()
{       TRISC = 0 ;
        For(z = 0 ; z < 4 ; z++)
        {       PORTC = MyDATA[z] ;
                Sum = sum +MyDATA[z] ;
```

```
    }
    Checksum = ~sum+1 ; PORTC = Checksum ;
    While(1) ;
}
```

Donner les valeurs des variables « sum » et « Checksum » après exécution du programme.

A quoi sert la variable Checksum ?

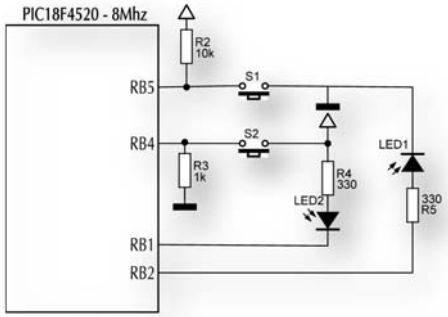
Sum = 0X25 + 0X62 + 0X3F + 0X52 = 0X18 (C= 1)

Chechsum = 0XE8

Cette variable est utilisée pour la vérification d'erreur.

Exercice 7

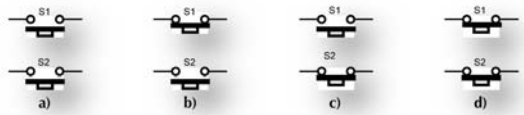
On donne le schéma du montage suivant :



1. Comment configurer le PORTB, sachant les bits RB7, RB6, RB3 et RB0 seront considérés comme des entrées. TRISB = 0b11111001 ;
2. Dans le programme C, on trouve l'instruction suivante : **BPs = PORTB & 0x30**.

Quel doit être le type de la variable **BPs** ? BPs est codée sur 8 bits, c'est le type « char »

3. Donner pour les cas suivants les valeurs possibles de **BPs** en Hexadécimal.



a) BPS = 0X20 b) BPS = 0X00 c) BPs = 0X30 d) BPs = 0X10

4. En utilisant l'instruction à choix multiples « switch … case », écrire le programme C permettant d'allumer une LED lorsque le bouton correspondant est enfoncé.

```
char BPs ;
void main()
{   TRISB = 0XF9 ;
    while(1)
    {   BPs = PORTB & 0X30 ;
        switch(BPs)
        {   case  0X20 : {RB1 = 1 ; RB2 = 0 ;}break ;
            case  0X00 : {RB1 = 1 ; RB2 = 1 ;}break ;
            case  0X30 : {RB1 = 0 ; RB2 = 0 ;}break ;
            case  0X01 : {RB1 = 0 ; RB2 = 1 ;}break ;
        }
    }
}
```


Exercice 8

1. En utilisant les masques, écrire l'instruction qui permet :
   - De mettre à 1 les bits RB3 et RB5 du PORTB ;PORTB =PORTB |0X28 ;
   - De mettre à 0 les bits RB1 et TB7 du PORTB ; PORTB =PORTB |0X7D ;
   - D'inverser le bit RB6 du PORTB; PORTB =PORTB ^0X40 ;
   - De mettre le bit RB2 à 1 et RB6 à 0. ; PORTB =(PORTB &0XBF) |0X40 ;
2. Ecrire les expressions booléennes sur les tests suivants : Expression vraie si,
   - RB2 à 1 et RB0 à 0 ; if((RB2 == 1)&&(RB0 == 0))
   - RB3 à 1 ou RB1 à 0 ; if((RB3 == 1)||(RB1 == 0))
3. Donner les valeurs possibles de l'expression suivante :

```
((p1 & 0x80) >>3)^(p1 & 0x10)
```

Deux possibilités : p1 = 0 et p1 = 0X10

Exercice 9

1. Traduire en langage C l'algorithme suivant :

```
Début
    PORTC ← 0x81 ;
    Répéter toujours
        PORTC ← (PORTC << 1)|(PORTC >> 7));
        Attente_1000ms;
    Fin répéter
Fin
```

```
void main()
{   PORTC = 0X01 ;
    while(1)
    {   PORTC = (PORTC << 1)|(PORTC >> 7) ;
        delay_ms(1000) ;
    }
}
```

2. Donner dans un tableau, les états possibles du PORTC

RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0
1	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1
0	0	0	0	0	1	1	0
0	0	0	0	1	1	0	0
0	0	0	1	1	0	0	0
0	0	1	1	0	0	0	0
0	1	1	0	0	0	0	0
1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	1

3. Quelle est la fonction réalisée par l'instruction :

```
PORTC ← (PORTC << 1)|(PORTC >> 7));
```

Rotation du PORTC

Exercice 10

1. Ecrire un programme permettant de faire clignoter la LED connectée à la broche RC0 au rythme de 500ms allumée; 500ms éteinte.

On utilisera la fonction prédéfinie delay_ms().

```
void main()
{       TRISC = 0xFE ;
        while(1)
        {       RC2 = !RC2 ;
                delay_ms(500) ;
        }
}
```

2. Modifier le programme pour faire clignoter toutes les LEDs du PORTC en même temps au rythme de 500ms allumée; 500ms éteinte.

```
void main()
{   TRISC = 0x00 ; PORTC = 0 ;
    while(1)
    {       PORTC = ~PORTC ;
            delay_ms(500) ;
    }
}
```

3. Modifier le programme pour faire clignoter les 8 leds par paquet de 4 au rythme de 500ms allumée ; 500ms éteinte.

```
void main()
{       TRISC = 0x00 ; PORTC = 0x0F ;
        while(1)
        {       PORTC = ~PORTC ;
                delay_ms(500) ;
        }
}
```


4. Modifier le programme pour faire le décalage à gauche puis à droite. Vous pouvez utiliser l'algorithme suivant :

```
Debut
PORTC ← 0x01
  Tant que (1) Faire
        Pour i allant de 0 à 6 Faire
            PORTC ← PORTC << 1
            Attente 500ms
        FinFaire
        Pour i allant de 0 à 6 Faire
            PORTC ← PORTC >> 1
            Attente 500ms
        FinFaire
  Fin Tant que
Fin
```

```
void main()
{       char i ;
        TRISC = 0x00 ; PORTC = 0x01 ;
        while(1)
        {       for(i = 0 ; i<7 ; i++){
                    PORTC = PORTC<<1 ;
                    delay_ms(500) ;
                }
                for(i = 0 ; i<7 ; i++){
                    PORTC = PORTC>>1 ;
                    delay_ms(500) ;
                }
        }
}
```

Exercice 11

Ecrire un programme C permettant d'allumer la Led (RC2) selon le tableau suivant :

RB1	RB0	Led
0	0	Led ← 0
0	1	Led ← 1
1	0	Led clignote à une fréq. De 1 Hz
1	1	Led clignote à une fréq. De 0,5 Hz

```
void main()
{   TRISC = 0x00 ; PORTC = 0x01 ;
```

```
        while(1)
        {       if((RB1 == 0)&&(RB0 == 0))
                        RC2 = 0 ;
                if((RB1 == 0)&&(RB0 == 1))
                        RC2 = 1 ;
                if((RB1 == 1)&&(RB0 == 0)){
                        RC2 = !RC2 ;
                        delay_ms(1000) ;
                }
                if((RB1 == 1)&&(RB0 == 1)){
                        RC2 = !RC2 ;
                        delay_ms(2000) ;
                }
        }
}
```