UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

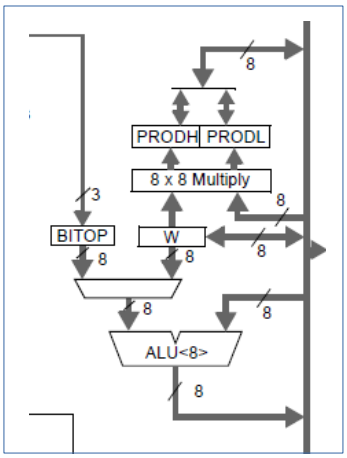
Annex. Recomanacions per la programació en C dels microcontroladors (e.g. família PIC18F)

1) Els microcontroladors no tenen (en general) un sistema operatiu per gestionar l'execució i sortida de programes, per tant hem de controlar nosaltres el seu acabament. En cas contrari el micro aniria executant instruccions de memòria de codi sense sentit. Això es fa típicament amb una estructura while():

```
// Inicialitzacions
while(1)
{
  // Codi a executar (per sempre)
}
```

```
// Inicialitzacions
// Codi a executar
  while(1);
        // esperarà aquí fins
        // que arribi un reset
```

2) Cal fixar-se en les capacitats de còmput del nostre microcontrolador (ALU i mòduls adjunts).



Mirem l'esquema del PIC18F de la dreta:
- Veiem que la ALU és de 8 bits.
- Veiem que hi ha mòdul de multiplicar però no per dividir.
- Veiem que el working register (W) està involucrat a les operacions de la ALU o el multiplicador.

Conseqüències:

2.1 De forma nativa no hi ha decimals: 3/2=1, 5/8=0 i els operadors no passaran de 2^8: 100+200=44

2.2 Cal però pensar bé el tipus de les variables que farem servir: BYTE, int, long segons les dades que han de contenir. La seva elecció afectarà el cost computacional:

La suma de 2 bytes, un cop passada a *assembler* implica:

```
MOVF    OP1,0,a     // copiem al W l'operand 1
ADDWF   OP2,0,a     // sumem al W
MOVWF   RES,a       // movem resultat
```

La suma de 2 ints (16b), un cop passada a *assembler* implica:

```
MOVF    LOW1,0,a    // copiem al W la part baixa 1
ADDWF   LOW2,0,a    // sumem parts baixes al W
MOVWF   RES_LOW,a   // movem part baixa del resultat
MOVF    HIGH1,0,a   // copiem al W la part alta 1
ADDWFC  HIGH2,0,a   // sumem les parts altes amb carry al W
MOVWF   RES_HIGH,a  // movem part alta del resultat
```

2.3 Al no tenir hardware per dividir, és millor fer les operacions de divisió per potències de 2 (s'implementen amb desplaçament a l'esquerra) i interpretar al final els resultats.

2.4 El tipus *float* no està suportat pel micro. Podem importar una llibreria que implementi el tipus a base d'un alt cost computacional. Pensar bé si és necessari o podem treballar amb un format de coma fixa.

Exemple: Si volem fer 3/32, primer fem VAR=3*1000, després RES= VAR/32 i sabem que el resultat representa mil·lèsimes.

3) Moltes vegades ens caldrà accedir a bits concrets dels registres; habitualment en llenguatge C es fa això mitjançant màscares per no afectar el registre sencer. Exemples:

```
REG = REG & 0xFE; // Posa a 0 el bit de menys pes (bit 0) del registre REG
REG = REG | 0x20;    // Posa a 1 el bit 5 del registre REG
REG = REG & 0x0F; // Posa a 0 els quatre bits de més pes del registre REG
REG = REG | 0x13;   // Posa a 1 els bits 0,1, i 4 del registre REG
REG = REG ^ 0x01;  // Nega el bit 0 del registre REG (XOR amb 1)
```

* No confondre els operadors bit a bit: | , & amb els booleans || i &&.

Però en el nostre cas, podem aprofitar l'existència d'instruccions en *assembler* d'accés a bits:

BIT-ORIENTED OPERATIONS						
BCF	f, b, a	Bit Clear f	1	1001 bbba ffff ffff	None	1, 2
BSF	f, b, a	Bit Set f	1	1000 bbba ffff ffff	None	1, 2
BTFSC	f, b, a	Bit Test f, Skip if Clear	1 (2 or 3)	1011 bbba ffff ffff	None	3, 4
BTFSS	f, b, a	Bit Test f, Skip if Set	1 (2 or 3)	1010 bbba ffff ffff	None	3, 4
BTG	f, d, a	Bit Toggle f	1	0111 bbba ffff ffff	None	1, 2

per exemple BCF, posa a 0 el bit b del registre f, és a dir, BCF PORTD , 2, 0 posa a 0 el bit 2 del PORTD.

4) Aprofitant l'anterior, el nostre compilador de C, permet crear estructures com la següent, que es troba en fitxer inclosos com ara xc8.h o P18F4550.h i en la que es permetrà accedir als bits individualment:

```
extern volatile near unsigned char    PORTD;
extern volatile near union {
struct {
  unsigned RD0:1; unsigned RD1:1;
  unsigned RD2:1; unsigned RD3:1;
  unsigned RD4:1; unsigned RD5:1;
  unsigned RD6:1; unsigned RD7:1;
};
} PORTDbits;
```

Llavors, si en el nostre codi fem una definició de l'estil:

```
#define LED PORTDbits.RD5     // Per indicar que hem connectat un LED aquí al bit 5 del Port D
#define APAGA 0
#define ENCEN 1
```

Podrem programar de manera més natural escrivint línies com:

```
LED = APAGA;          // El compilador la traduirà a BCF PORTD,5,0
LED = ENCEN;          // El compilador la traduirà a BSF PORTD,5,0
```

5) Donada la mida de la pila i de la memòria disponible, no plantejarem mai algorismes de forma recursiva. Segurament un microcontrolador no és la plataforma adequada pel problema plantejat que necessiti tal solució.

6) En algun lloc del programa hem d'establir la configuració del microcontrolador (selecció de l'oscil·lador, activació de watchdog, etc.). Habitualment això es delega en un fitxer .h (per exemple config.h) que dins seu tingui la definició d'aquests paràmetres.

```
// file config.h
#pragma config PLLDIV   = 2                 // (8 MHz crystal on EASYPIC6 board)
#pragma config CPUDIV   = OSC1_PLL2         // No dividim. Treballem a 8MHz
#pragma config USBDIV   = 2                 // Clock source from 96MHz PLL/2
...                                         // Resta de configuracions
```